Choreography automata

Choreography Automata*

Franco Barbanera[1], Ivan Lanese[2], and Emilio Tuosto[3]
barba@dmi.unict.it, ivan.lanese@gmail.com, emilio.tuosto@gssi.it

[1] Dept. of Mathematics and Computer Science, University of Catania (Italy)
[2] Focus Team, University of Bologna/INRIA (Italy)
[3] Gran Sasso Science Institute (Italy) and University of Leicester (UK)

Abstract. Automata models are well-established in many areas of computer science and are supported by a wealth of theoretical results including a wide range of algorithms and techniques to specify and analyse systems. We introduce *choreography automata* for the choreographic modelling of communicating systems. The projection of a choreography automaton yields a system of *communicating finite-state machines*. We consider both the standard asynchronous semantics of communicating systems and a synchronous variant of it. For both, the projections of well-formed automata are proved to be live as well as lock- and deadlock-free.

1 Introduction

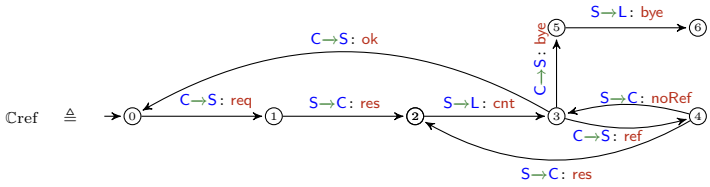
Choreographies are gaining momentum in the design and implementation of distributed applications also in the ICT industrial sector. This is witnessed by the effort of defining standards for specification languages such as WS-CDL [31] or BPMN [40] as well as the recognition of choreographies as suitable approaches to describe modern architectures such as microservices [12,2]. Choreographic approaches to the modelling, analysis, and programming of message-passing applications abound. For instance, in [34,5] abstract models have been applied to verify and debug BPMN specifications. Also, behavioural types have been proposed as suitable formalisations of choreographies [29] and for the analysis of properties such as liveness or deadlock freedom (e.g., [45,20] and the survey [30] to mention but few), while other approaches have considered syntax-free models [48]. At a programming level, choreographic programming has been explored in [35,39].

A distinguished trait of choreographies is the coexistence of two distinct but related views of a distributed system: the *global* and the *local* views. The former is an abstraction that yields a *holistic* description of the system. A global view indeed describes the coordination necessary among the various components of

* Research partly supported by the EU H2020 RISE programme under the Marie Skłodowska-Curie grant agreement No 778233, by the MIUR project PRIN 2017FTXR7S "IT-MaTTerS" (Methods and Tools for Trustworthy Smart Systems). and by the Piano Triennale Ricerca - UNICT 2016-19. The first and second authors have also been partially supported by INdAM as members of GNCS (Grup po Nazionale per il Calcolo Scientifico). The authors thanks the reviewers for their helpful comments and also M. Dezani for her support.

the system "altogether". In contrast, the local views specify the behaviour of the single components in "isolation".

In this paper we revisit the use of finite state automata to formally specify (and analyse) global views of message-passing systems, following an intuition similar to conversation protocols (CP) [26,16,27], a formalism where choreographies for asynchronous systems are described by means of Büchi automata. Our model, dubbed *choreography automata* (c-automata, for short), differs from CP in spite of some similarities. A comparison is given in § 6. The transitions of c-automata are labelled with *interactions*. As in most approaches, an interaction A→B: m states that *participant* A sends message m to participant B, which in turn receives it. For instance, consider the c-automaton



(1)

used to illustrate our model and as our working example through the paper. The c-automaton ℂref specifies the coordination among participants C, S, and L whereby a request req from client C is served by server S which replies with a message (of type) res and logs some meta-information cnt on a service L (e.g., for billing purposes). Client C may acknowledge a response of S (*i*) with an ok message to restart the protocol, or (*ii*) by requiring a refinement of the response with a ref message, or else (*iii*) by ending the protocol with a bye message which S forwards to L. In the second case, S sends C either a noRef message if no refinement is possible or another res (with the corresponding cnt to L).

Note that ℂref has nested as well as entangled loops. The support for entangled loops is a distinguished and expressive feature of c-automata, not present in many existing models of choreographies or multiparty session types (MST), and that we shall discuss in § 6.

We argue that c-automata provide a number of benefits. An advantage of c-automata is that finite state automata are well-known structures used both in theoretical and applied computer science. For instance, the c-automaton ℂref above can be easily understood by practitioners while retaining rigour. Another advantage is that c-automata are *syntax-independent*; they do not rely on complex linguistic constructs (such as the process algebraic constructs usually adopted in behavioural types). More crucially, we can re-use well-known results of the theory of automata and formal languages (e.g., we use determinisation and trace equivalence) as well as related algorithms. We discuss these advantages more extensively in § 6.

Choreographies enable a so called top-down approach whereby local views can be *projected* from the global view. Projections are expected to reflect the global specification without spoiling communication soundness (e.g., deadlock freedom, liveness, etc.). These results do not hold in general. In fact, global

views abstract away from "low level" peculiarities and projections may exhibit unintended behaviour.

The *realisability* of a global specification is a natural question to ask:

> Can global views such as $\mathbb{C}$ref be realised by distributed components C, S, and L coordinating with each other without intermediaries?

The answer to such question (obviously) depends on the communication infrastructure the distributed components use for the coordination. In fact, global views in general *abstractly* specify the coordination disregarding several details. For instance, the c-automaton $\mathbb{C}$ref in (1) is oblivious of the communication infrastructure used by the participants to coordinate with each other. Are the communications among C, S, and L synchronous or asynchronous? In the latter case, are messages received in their sending order? How is the sequencing reflected at the local level? For instance, should the messages that C sends from state 3 in (1) be sent after L receives the message cnt from S?

Tackling the realisability of global views is not a trivial endeavour. For instance, the recent analysis done in [45] highlights glitches in several projection operations of behavioural types. Also, some decidability results on the realisability of CPs [9], the only other automata-based choreographic setting that we are aware of, have been recently proved erroneous [24].

One would also like to understand whether the distributed components realising a choreography enjoy nice communication properties, e.g., will a component ready to engage in a communication eventually progress? Will a message sent by a participant eventually be received? We will consider such problems, showing that a set of conditions we define on c-automata do guarantee the choreography both to be realisable and to enjoy a number of relevant communication properties such as liveness and deadlock freedom.

Contributions & Structure. After a preliminary section (§ 2) recalling the main notions we deal with in the paper, in § 3 we formalise c-automata and their projections.We adopt *communicating systems* [13] (reviewed in § 2) for the local views of choreographies.

We consider both the case of synchronous and asynchronous communications for the local views. The projection from c-automata to communicating systems is defined in § 3 while in § 4 we define the class of *well-formed* c-automata for the synchronous case. There we show that, on well-formed c-automata, our notion of projection is correct (cf. Thm. 4.14) and guarantees liveness, lock- and deadlock-freedom in the synchronous semantics (cf. Thm. 4.15). In § 5 we generalise the above results to the case of asynchronous communications (cf. Thms. 5.6 and 5.7). Concluding remarks, related and future work are discussed in § 6. Additional material and complete proofs can be found in [7].

Some interesting technical points are worth noticing. Firstly, most of our constructions and results rely on basic notions of formal languages and automata theory. This greatly simplifies the presentation and the proofs. The generalisation from synchronous to asynchronous communications requires only a mild

strengthening of our notion of well-formedness and no changes to c-automata or their projection. These are further advantages of the use of finite-state automata.

2 Preliminaries

A *Labelled Transition System* (LTS) is a tuple $A = \langle \mathbb{S}, s_0, \mathcal{L}, \rightarrow \rangle$ where

- $\mathbb{S}$ is a set of states (ranged over by $s, q, \ldots$) and $s_0 \in \mathbb{S}$ is the *initial state*;
- $\mathcal{L}$ is a finite set of labels (ranged over by $l, \lambda, \ldots$);
- $\rightarrow \subseteq \mathbb{S} \times (\mathcal{L} \cup \{ \varepsilon \}) \times \mathbb{S}$ is a set of transitions where $\varepsilon \notin \mathcal{L}$ is a distinguished label.

We define a Finite-State Automaton (FSA) as an LTS where $\mathbb{S}$ is finite. We use the usual notation $s_1 \xrightarrow{\lambda} s_2$ for the transition $(s_1, \lambda, s_2) \in \rightarrow$, and $s_1 \rightarrow s_2$ when there exists λ such that $s_1 \xrightarrow{\lambda} s_2$, as well as $\rightarrow^*$ for the reflexive and transitive closure of $\rightarrow$. The set of *reachable states* of A is $\mathcal{R}(A) = \{ s \mid s_0 \rightarrow^* s \}$.

Remark 2.1. Our definition of FSA omits the set of *accepting* states since we consider only FSAs where each state is accepting (which is the normal case in LTSs). We discuss this point further at the end of the paper. ◇

We recall standard notions on LTSs.

Definition 2.2 (Traces and Trace equivalence). *A* run *of an LTS* $A = \langle \mathbb{S}, s_0, \mathcal{L}, \rightarrow \rangle$ *is a (possibly empty) finite or infinite sequence of consecutive transitions starting at* s_0*. The* trace *(or* word*)* w *of a run* $(s_{i-1} \xrightarrow{\lambda_{i-1}} s_i)_{1 \leq i \leq n}$ *of* A *is the concatenation of the labels of the run (assume* $n = \infty$ *if the run is infinite), namely* $w = \lambda_0 \cdot \lambda_1 \cdots \lambda_n$*; label* ε*, as usual, denotes the identity element of concatenation; if the run is empty then* $w = \varepsilon$*.*

The language $\mathfrak{L}(A)$ *of* A *is the set of the traces of the runs of* A*. Two LTSs* A *and* B *are trace equivalent iff* $\mathfrak{L}(A) = \mathfrak{L}(B)$*. Also,* A accepts w *if* $w \in \mathfrak{L}(A)$*,* A accepts w from s *if* $w \in \mathfrak{L}(\langle \mathbb{S}, s, \mathcal{L}, \rightarrow \rangle)$*, and an* s*-run (resp.* s*-trace) of* A *is a run (resp. trace) of* $\langle \mathbb{S}, s, \mathcal{L}, \rightarrow \rangle$*.*

The notion of language in the definition above includes infinite words; this extends the standard notion of language accepted by an FSA. In particular, we consider an infinite word to be accepted by an FSA if each of its prefixes is accepted in the standard way. This is equivalent to look at an FSA both as a standard FSA and as a Büchi automaton where all the states are final.

Definition 2.3 (Deterministic LTSs). *An LTS* $A = \langle \mathbb{S}, s_0, \mathcal{L}, \rightarrow \rangle$ *is* deterministic *if*
– *it is* ε*-free, i.e. there is no transition of the form* $q \xrightarrow{\varepsilon} q'$*, and*
– *whenever* $q \xrightarrow{\lambda} q_1$ *and* $q \xrightarrow{\lambda} q_2$ *then* $q_1 = q_2$*.*
We denote the determinisation of A *(i.e. the translation of a nondeterministic LTS/FSA to a deterministic one) as* $\mathsf{det}(A)$ [4]*.*

[4] The result of $\mathsf{det}(A)$ may actually depend on the chosen algorithm, but that is irrelevant for our results.

We adopt communicating finite-state machines (CFSMs) [13] to model the local behaviour of systems of distributed components. The following definitions are borrowed from [13] and adapted to our context.

Let $\mathfrak{P}$ be a set of *participants* (or *roles*, ranged over by A, B, etc.) and $\mathcal{M}$ a set of *messages* (ranged over by m, n, etc.). We take $\mathfrak{P}$ and $\mathcal{M}$ disjoint.

Definition 2.4 (Communicating system). *A* communicating finite-state machine *(CFSM) is an FSA on the set*

$$\mathcal{L}_{act} = \{\mathsf{A}\,\mathsf{B}!\mathsf{m}, \mathsf{A}\,\mathsf{B}?\mathsf{m} \mid \mathsf{A}, \mathsf{B} \in \mathfrak{P}, \mathsf{m} \in \mathcal{M}\}$$

of actions. *The* subject *of an output (resp. input) action* $\mathsf{A}\,\mathsf{B}!\mathsf{m}$ *(resp.* $\mathsf{A}\,\mathsf{B}?\mathsf{m}$*) is* A *(resp.* B*). A CFSM is* A-local *if all its transitions have subject* A*.*

A (communicating) system *is a map* $\mathsf{S} = (M_\mathsf{A})_{\mathsf{A}\in\mathcal{P}}$ *assigning an* A*-local CFSM* M_A *to each participant* $\mathsf{A} \in \mathcal{P}$ *such that* $\mathcal{P} \subseteq \mathfrak{P}$ *is finite and any participant occurring in a transition of* M_A *is in* $\mathcal{P}$*.*

Note that CFSMs may contain ε-transitions. However, projection (see Def. 3.3 below) yields ε-free CFSMs.

Besides being a well-known and widely adopted model, CFSMs are equipped with both synchronous and asynchronous semantics. This enables a uniform treatment of both communication models. The use of CFMSs is also helpful to compare c-automata with other models which are projected on CFSMs as well, such as global graphs [37] and some versions of global types [23].

The synchronous semantics of communicating systems is an LTS where labels are *interactions*:

$$\mathcal{L}_{\text{int}} \;=\; \{\mathsf{A}{\to}\mathsf{B}\colon \mathsf{m} \mid \mathsf{A} \neq \mathsf{B} \in \mathfrak{P} \text{ and } \mathsf{m} \in \mathcal{M}\}$$

Definition 2.5 (Synchronous semantics). *Let* $\mathsf{S} = (M_\mathsf{A})_{\mathsf{A}\in\mathcal{P}}$ *be a communicating system where* $M_\mathsf{A} = \langle \mathbb{S}_\mathsf{A}, q_{0\mathsf{A}}, \mathcal{L}_{act}, \to_\mathsf{A}\rangle$ *for each participant* $\mathsf{A} \in \mathcal{P}$*. A* synchronous configuration of S *is a map* $s = (q_\mathsf{A})_{\mathsf{A}\in\mathcal{P}}$ *assigning a* local state $q_\mathsf{A} \in \mathbb{S}_\mathsf{A}$ *to each* $\mathsf{A} \in \mathcal{P}$*. We denote* q_A *by* $s(\mathsf{A})$ *and may denote* s *by* $\vec{q}$*.*

The synchronous semantics *of* S *is the transition system* $[\![\mathsf{S}]\!]^{\mathsf{s}} = \langle \mathbb{S}, \vec{q}_0, \mathcal{L}_{int}, \to\rangle$ *defined as follows*

- $\mathbb{S}$ *is the set of synchronous configurations of* S*, as defined above, and* $\vec{q}_0 = (q_{0\mathsf{A}})_{\mathsf{A}\in\mathcal{P}} \in \mathbb{S}$ *is the* initial *configuration*
- $\vec{q}_1 \xrightarrow{\mathsf{A}\to\mathsf{B}\colon \mathsf{m}} \vec{q}_2$ *if*
  1. $\vec{q}_1(\mathsf{A}) \xrightarrow{\mathsf{A}\,\mathsf{B}!\mathsf{m}}_\mathsf{A} \vec{q}_2(\mathsf{A})$ *and* $\vec{q}_1(\mathsf{B}) \xrightarrow{\mathsf{A}\,\mathsf{B}?\mathsf{m}}_\mathsf{B} \vec{q}_2(\mathsf{B})$*, and*
  2. *for all* $\mathsf{C} \neq \mathsf{A}, \mathsf{B}$*,* $\vec{q}_1(\mathsf{C}) = \vec{q}_2(\mathsf{C})$*.*

  In this case, we say that $\vec{q}_1(\mathsf{A}) \xrightarrow{\mathsf{A}\,\mathsf{B}!\mathsf{m}}_\mathsf{A} \vec{q}_2(\mathsf{A})$ *and* $\vec{q}_1(\mathsf{B}) \xrightarrow{\mathsf{A}\,\mathsf{B}?\mathsf{m}}_\mathsf{B} \vec{q}_2(\mathsf{B})$ *are* component transitions of $\vec{q}_1 \xrightarrow{\mathsf{A}\to\mathsf{B}\colon \mathsf{m}} \vec{q}_2$*.*
- $\vec{q}_1 \xrightarrow{\varepsilon} \vec{q}_2$ *if* $\vec{q}_1(\mathsf{A}) \xrightarrow{\varepsilon}_\mathsf{A} \vec{q}_2(\mathsf{A})$*, and for all* $\mathsf{B} \neq \mathsf{A}$*,* $\vec{q}_1(\mathsf{B}) = \vec{q}_2(\mathsf{B})$*.*

Note that ε-transitions in the semantics of a communicating system are induced by those of the constituent CFSMs. Also, $[\![\mathsf{S}]\!]^{\mathsf{s}}$ is finite; in fact, it is in general a non-deterministic automaton on the alphabet $\mathcal{L}_{\text{int}}$.

As one would expect, the notion of synchronous semantics is invariant under language equivalence of CFSMs.

Proposition 2.6. *Let* $\mathsf{S} = (M_\mathsf{A})_{\mathsf{A}\in\mathcal{P}}$ *and* $\mathsf{S}' = (M'_\mathsf{A})_{\mathsf{A}\in\mathcal{P}}$ *be two communicating systems. If* $\mathfrak{L}(M_\mathsf{A}) = \mathfrak{L}(M'_\mathsf{A})$ *for all* $\mathsf{A} \in \mathcal{P}$ *then* $\mathfrak{L}([\![\mathsf{S}]\!]^{\mathsf{s}}) = \mathfrak{L}([\![\mathsf{S}']\!]^{\mathsf{s}})$.

The asynchronous semantics of systems is defined in terms of transition systems which keep track of both the state of each machine and the content of unbounded FIFO queues b_{AB} which are associated to each channel $(\mathsf{A},\mathsf{B}) \in \mathcal{C}$, where $\mathcal{C} = \mathcal{P}\times\mathcal{P}\setminus\{(\mathsf{A},\mathsf{A}) \mid \mathsf{A}\in\mathcal{P}\}$. The queue b_{AB} is where M_A puts the messages to M_B and from which M_B consumes the messages from M_A. To avoid cumbersome parenthesis, we write $\mathsf{A}\,\mathsf{B} \in \mathcal{C}$ for $(\mathsf{A},\mathsf{B}) \in \mathcal{C}$.

Definition 2.7 (Asynchronous semantics). *Let* $\mathsf{S} = (M_\mathsf{A})_{\mathsf{A}\in\mathcal{P}}$ *be a communicating system where* $M_\mathsf{A} = \langle \mathbb{S}_\mathsf{A}, q_{0\mathsf{A}}, \mathcal{L}_{act}, \to_\mathsf{A}\rangle$ *for each participant* $\mathsf{A}\in\mathcal{P}$. *An* asynchronous configuration *of* S *is a pair* $s = \langle \vec{q}\,;\,\vec{b}\rangle$ *where* $\vec{q} = (q_\mathsf{A})_{\mathsf{A}\in\mathcal{P}}$ *with* $q_\mathsf{A}\in Q_\mathsf{A}$ *and* $\vec{b} = (b_{\mathsf{AB}})_{\mathsf{AB}\in\mathcal{C}}$ *with* $b_{\mathsf{AB}}\in\mathcal{M}^*$; *we write* $s(\mathsf{A})$ *for* $\vec{q}(\mathsf{A})$ *and denote by* ε *the empty queue. The* asynchronous semantics *of* S *is the transition system* $[\![\mathsf{S}]\!]^{\mathsf{a}} = \langle \mathbb{S}, s_0, \mathcal{L}_{act}, \to\rangle$ *defined as follows*

- $\mathbb{S}$ *is the set of asynchronous configurations of* S *and* $s_0 = \langle \vec{q_0}\,;\,\vec{b}\rangle$ *is the* initial *configuration where* $\vec{q_0} = (q_{0\mathsf{A}})_{\mathsf{A}\in\mathcal{P}}$ *and all the queues are empty.*
- $s \xrightarrow{l} s'$ *if* $s = \langle \vec{q}\,;\,\vec{b}\rangle$, $s' = \langle \vec{q'}\,;\,\vec{b'}\rangle$ *and either (1) or (2) below holds:*

1. $l = \mathsf{A}\,\mathsf{B}!\mathsf{m}$ *and* $s(\mathsf{A}) \xrightarrow{l}_\mathsf{A} s'(\mathsf{A})$ *and*
   a. $s(\mathsf{C})' = s(\mathsf{C})$ $\forall \mathsf{C}\neq\mathsf{A}\in\mathcal{P}$ *and*
   b. $b'_{\mathsf{A}\,\mathsf{B}} = b_{\mathsf{A}\,\mathsf{B}}.\mathsf{m}$ *and*
   c. $b'_{\mathsf{C}\,\mathsf{D}} = b_{\mathsf{C}\,\mathsf{D}}$ *for all* $\mathsf{C}\,\mathsf{D}\in\mathcal{C}, \mathsf{C}\,\mathsf{D}\neq\mathsf{A}\,\mathsf{B}$
2. $l = \mathsf{A}\,\mathsf{B}?\mathsf{m}$ *and* $s(\mathsf{B}) \xrightarrow{l}_\mathsf{B} s'(\mathsf{B})$ *and*
   a. $s'(\mathsf{C}) = s(\mathsf{C})$ $\forall \mathsf{C}\neq\mathsf{B}\in\mathcal{P}$ *and*
   b. $b_{\mathsf{A}\,\mathsf{B}} = \mathsf{m}.b'_{\mathsf{A}\,\mathsf{B}}$ *and*
   c. $b'_{\mathsf{C}\,\mathsf{D}} = b_{\mathsf{C}\,\mathsf{D}}$ *for all* $\mathsf{C}\,\mathsf{D}\in\mathcal{C}, \mathsf{C}\,\mathsf{D}\neq\mathsf{A}\,\mathsf{B}$

  In the first (resp. second) case we say that $s(\mathsf{A}) \xrightarrow{\mathsf{A}\,\mathsf{B}!\mathsf{m}}_\mathsf{A} s'(\mathsf{A})$ *(resp.* $s(\mathsf{B}) \xrightarrow{\mathsf{A}\,\mathsf{B}?\mathsf{m}}_\mathsf{B} s'(\mathsf{B})$*) is a component transition of* $s \xrightarrow{\mathsf{A}\to\mathsf{B}:\,\mathsf{m}} s'$.
- $\langle \vec{q}\,;\,\vec{b}\rangle \xrightarrow{\varepsilon} \langle \vec{q'}\,;\,\vec{b'}\rangle$ *if* $\vec{q}(\mathsf{A}) \xrightarrow{\varepsilon} \vec{q'}(\mathsf{A})$ *for some* $\mathsf{A}\in\mathcal{P}$ *and for all* $\mathsf{B}\neq\mathsf{A}$, $\vec{q}(\mathsf{B}) = \vec{q'}(\mathsf{B})$, *and* $\vec{b} = \vec{b'}$.

State q_A keeps track of the state of the machine M_A and buffer b_{AB} keeps track of the messages sent from A to B (and not yet received by B). In a transition $s \xrightarrow{\mathsf{A}\,\mathsf{B}!\mathsf{m}} s'$, participant A adds message m in the queue of the channel $\mathsf{A}\,\mathsf{B}$ and symmetrically, in a transition $s \xrightarrow{\mathsf{A}\,\mathsf{B}?\mathsf{m}} s'$, participant B consumes message m from the top of the queue of the channel $\mathsf{A}\,\mathsf{B}$. In both cases, any machine or queue not involved in the transition is left unchanged.

The asynchronous semantics is also invariant under equivalence of CFSMs.

Proposition 2.8. *Let* $\mathsf{S} = (M_\mathsf{A})_{\mathsf{A}\in\mathcal{P}}$ *and* $\mathsf{S}' = (M'_\mathsf{A})_{\mathsf{A}\in\mathcal{P}}$ *be two communicating systems. If* $\mathfrak{L}(M_\mathsf{A}) = \mathfrak{L}(M'_\mathsf{A})$ *for all* $\mathsf{A}\in\mathcal{P}$ *then* $\mathfrak{L}([\![\mathsf{S}]\!]^{\mathsf{a}}) = \mathfrak{L}([\![\mathsf{S}']\!]^{\mathsf{a}})$.

For both the synchronous and the asynchronous semantics we restrict the attention to *fair runs*. An infinite run is fair if each transition which is continuously enabled is taken in a finite number of steps. A finite run is always fair.

We are interested in standard properties of communicating systems which we now recall. Definitions are alike in the synchronous and asynchronous semantics, hence, to avoid repetitions, below $[\![\_]\!]$ stands for $[\![\_]\!]^{\mathsf{a}}$ or $[\![\_]\!]^{\mathsf{s}}$.

Definition 2.9 (Communication properties). *Let* $\mathsf{S} = (M_{\mathsf{A}})_{\mathsf{A}\in\mathcal{P}}$ *be a communicating system.*

i) **Liveness***:* S *is* live *if for each configuration* $s \in \mathcal{R}(\llbracket \mathsf{S} \rrbracket)$*, each* $\mathsf{A} \in \mathcal{P}$ *and each outgoing transition* t *from* $s(\mathsf{A})$ *in* M_{A} *there exists a run of* $\llbracket \mathsf{S} \rrbracket$ *from* s *including a transition which has* t *as component transition.*

ii) **Lock freedom***: A configuration* $s \in \mathcal{R}(\llbracket \mathsf{S} \rrbracket)$ *is a* lock *if*
- *there is* $\mathsf{A} \in \mathcal{P}$ *with an outgoing transition* t *from* $s(\mathsf{A})$ *in* M_{A} *and*
- *there exists a run of* $\llbracket \mathsf{S} \rrbracket$ *starting from* s*, maximal w.r.t. prefix order, and containing no transition* t' *involving* A*.*

System S *is* lock-free *if for each* $s \in \mathcal{R}(\llbracket \mathsf{S} \rrbracket)$*,* s *is not a lock.*

iii) **Deadlock freedom***: A configuration* $s \in \mathcal{R}(\llbracket \mathsf{S} \rrbracket)$ *is a* deadlock *if*
- s *has no outgoing transitions in* $\llbracket \mathsf{S} \rrbracket$ *and*
- *there exists* $\mathsf{A} \in \mathcal{P}$ *such that* $s(\mathsf{A})$ *has an outgoing transition in* M_{A}*.*

System S *is* deadlock-free *if for each* $s \in \mathcal{R}(\llbracket \mathsf{S} \rrbracket)$*,* s *is not a deadlock.*

Liveness, as in [41], establishes the progress of communicating systems we are interested in. Lock freedom casts in our framework the idea that, similarly to [33,32], certain communications happen, whereas deadlock freedom extends the definition of deadlock in [19] to a setting which can be synchronous or asynchronous (as done also in [37,48]).

3 Choreography Automata

We introduce *choreography automata* (c-automata) as an expressive and flexible model of global specifications, following the styles of conversation protocols [27], choreographies [14,31,40], global graphs [48] and multiparty session types [17,28,30]. As customary in choreographic frameworks, we show how to project c-automata on local specifications. As anticipated, our projection yields a system of CFSMs formalising the local behaviour of the participants of a choreography.

C-automata (ranged over by $\mathbb{C}\mathrm{A}$, $\mathbb{C}\mathrm{B}$, etc.) are FSAs with labels in $\mathcal{L}_{\mathrm{int}}$.

Definition 3.1 (Choreography automata). *A* choreography automaton *(c-automaton) is an FSA on the alphabet* $\mathcal{L}_{int}$*. Elements of* $\mathcal{L}^*_{int}$ *are* choreography words*, subsets of* $\mathcal{L}^*_{int}$ *are* choreography languages*.*

Remark 3.2. Def. 3.1 admits non-deterministic c-automata. This does not increase the expressiveness of our framework. In fact, (*i*) the notions that we use for our results rely on traces and (*ii*) our projection operation (cf. Def. 3.3) is insensitive to non-determinism (cf. Prop. 3.6). Non-deterministic specifications are however desirable since they are easier to attain for the designer. ◇

Given a c-automaton, our projection operation builds the corresponding communicating system consisting of the set of projections of the c-automaton on each participant, each projection yelding a CFSM. Hereafter, $\mathcal{P} \subseteq \mathfrak{P}$ is the set of participants of c-automata; note that $\mathcal{P}$ is necessarily finite.

Definition 3.3 (Automata Projection). *The* projection on A *of a transition* $t = q \xrightarrow{\lambda} q'$ *of a c-automaton, written* $t{\downarrow_{\mathsf{A}}}$*, is defined by:*

$$t{\downarrow_{\mathsf{A}}} = \begin{cases} q \xrightarrow{\mathsf{A}\,\mathsf{C}!\mathsf{m}} q' & \text{if } \lambda = \mathsf{B}{\to}\mathsf{C}\colon \mathsf{m} \quad \text{and} \quad \mathsf{B} = \mathsf{A} \\ q \xrightarrow{\mathsf{B}\,\mathsf{A}?\mathsf{m}} q' & \text{if } \lambda = \mathsf{B}{\to}\mathsf{C}\colon \mathsf{m} \quad \text{and} \quad \mathsf{C} = \mathsf{A} \\ q \xrightarrow{\varepsilon} q' & \text{otherwise} \end{cases}$$

The projection *of a c-automaton* $\mathbb{C}\mathrm{A} = \langle \mathbb{S}, q_0, \mathcal{L}_{int}, \to \rangle$ *on a participant* $\mathsf{A} \in \mathcal{P}$*, denoted* $\mathbb{C}\mathrm{A}\downarrow_{\mathsf{A}}$*, is obtained by determinising and minimising up-to-language equivalence the* intermediate *automaton*

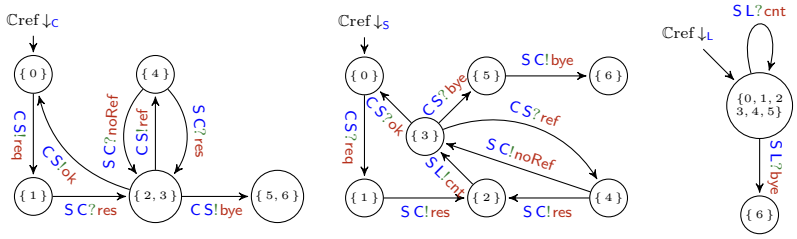
$$A_{\mathsf{A}} = \langle \mathbb{S}, q_0, \mathcal{L}_{act} \cup \{\,\varepsilon\,\}, \{\,(q \xrightarrow{\lambda} q'){\downarrow_{\mathsf{A}}} \mid q \xrightarrow{\lambda} q'\,\} \rangle$$

The projection of $\mathbb{C}\mathrm{A}$*, written* $\mathbb{C}\mathrm{A}\downarrow$*, is the communicating system* $(\mathbb{C}\mathrm{A}\downarrow_{\mathsf{A}})_{\mathsf{A}\in\mathcal{P}}$*. The projection function trivially extends to choreography words and languages.*

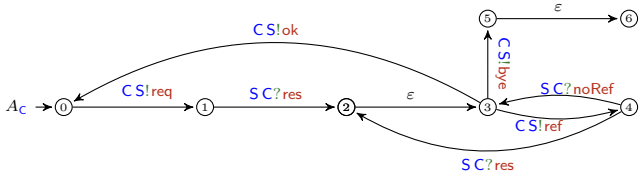
The projection defined above, apart for determinisation and minimisation, is essentially homomorphic, as most of the projections in the literature. Other approaches such as [43,25] add hidden communications to be able to deal with larger classes of choreographies. We prefer the former approach for its simplicity. Hidden communications can however be added directly at the choreographic level as proposed in [36].

It is a simple observation that the projection on A of $\mathbb{C}\mathrm{A}$ is A-local, deterministic and hence ε-free. Thanks to the properties of determinisation and minimisation (as, e.g., in the partition refinement algorithm [42]), the states of $\mathbb{C}\mathrm{A}\downarrow_{\mathsf{A}}$ are sets of states of $\mathbb{C}\mathrm{A}$.

Example 3.4 (Projections of $\mathbb{C}\mathrm{ref}$*).* The projections of our working example are



For instance, $\mathbb{C}\mathrm{ref}\downarrow_{\mathsf{C}}$ is obtained by determinising (minimisation is the identity in this case) the following intermediate automaton obtained as described in Def. 3.3.



◇

The following proposition relates the language of the projection with the language of the original automaton.

Proposition 3.5. *For all c-automata* $\mathbb{C}\mathrm{A}$ *and* $\mathsf{A} \in \mathcal{P}$, $\mathfrak{L}(\mathbb{C}\mathrm{A}{\downarrow_{\mathsf{A}}}) = \mathfrak{L}(\mathbb{C}\mathrm{A}){\downarrow_{\mathsf{A}}}$.

The projection operation is well-behaved with respect to trace equivalence.

Proposition 3.6. *If* $\mathbb{C}\mathrm{A}$ *and* $\mathbb{C}\mathrm{A}'$ *are trace-equivalent c-automata then* $\mathbb{C}\mathrm{A}{\downarrow_{\mathsf{A}}}$ *and* $\mathbb{C}\mathrm{A}'{\downarrow_{\mathsf{A}}}$ *are isomorphic for each participant* $\mathsf{A} \in \mathcal{P}$.

4 Well-formed Choreography Automata

To ensure that the communicating system obtained by projection of a c-automaton is well-behaved, some conditions are necessary. Since the conditions depend on the used communication infrastructure, we consider first synchronous communication, leaving to § 5 the case of asynchronous communication.

Definition 4.1 (Concurrent transitions). *Let* $\mathbb{C}\mathrm{A} = \langle \mathbb{S}, q_0, \mathcal{L}, \rightarrow \rangle$. *Two transitions* $q \xrightarrow{l_1} q_1$ *and* $q \xrightarrow{l_2} q_2$ *are concurrent iff there is a state* $q' \in \mathbb{S}$ *and transitions* $q_1 \xrightarrow{l_2} q'$ *and* $q_2 \xrightarrow{l_1} q'$.

Well-branchedness (cf. Def. 4.6) is a key notion which intuitively states that each participant is aware of the choices made in the choreography when its behaviour depends on those choices. The awareness of choice is checked on *spans*, namely pairs of runs that may constitute alternative branches of choices. Spans are formalised building on the notion of *candidate branch* which, in turn, is defined in terms of *pre-candidate branch*.

Definition 4.2 (Candidate q-branch). *Let* q *be a state of a c-automaton* $\mathbb{C}\mathrm{A}$. *A* pre-candidate q-branch *of* $\mathbb{C}\mathrm{A}$ *is a* q*-run of* $\mathbb{C}\mathrm{A}$ *such that each cycle has at most one occurrence within the whole run (i.e. any subsequence of the form* $q \rightarrow \ldots \rightarrow q$*, where* q *occurs only at the beginning and at the end of the subsequence, is not present more than once in the run). A* candidate q-branch *is a maximal pre-candidate* q*-branch with respect to prefix order.*

We often refer to a (pre-)candidate q-branch simply as "(pre-)candidate of q". Due to the condition about cycles in Def. 4.2, the following holds trivially.

Fact 1. *Given a state* q *of a c-automaton* $\mathbb{C}\mathrm{A}$, *the set of its pre-canditates is finite, and so is, a fortiori, that of its candidates.*

Example 4.3 ((Pre-)candidate branches in $\mathbb{C}\mathrm{ref}$*).* The sequences in Fig. 1 are runs of the c-automaton of our working example. They are all pre-candidates of either 3 or 4, but run π_e, which is not a pre-candidate of 4 since the cycle $4 - 3 - 4$ occurs twice. Runs π_b and π_d are also candidates of 3, being maximal pre-candidates with respect to prefix order. ◇

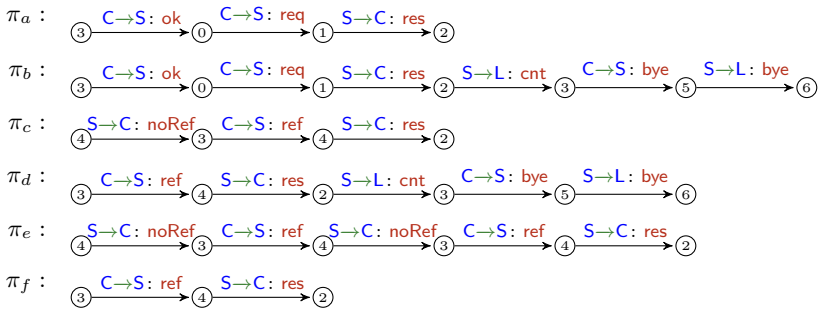


Fig. 1. Runs of $\mathbb{C}$ref.

Definition 4.4 (q-span). *Given a state q of a c-automaton $\mathbb{C}\mathrm{A}$, a pair (σ, σ') of pre-candidate q-branches of $\mathbb{C}\mathrm{A}$ is a* q-span *if σ and σ' are*

- *either cofinal, with no common node but q and the last one;*
- *or candidate q-branches with no common node but q;*
- *or a candidate q-branch and a loop on q with no other common nodes.*

A participant $\mathsf{A} \in \mathcal{P}$ chooses *at a q-span (σ, σ') if the first transition of both σ and σ' has A as sender.*

Example 4.5 (Spans of $\mathbb{C}$ref*).* The states with spans of our working example are 3 and 4. A span from 3 is (π_a, π_f), where π_a and π_f are as in Fig. 1. Indeed, π_a and π_f are cofinal (in 2) pre-candidates of 3 with no common states but 3 and 2. Participant C chooses at (π_a, π_f). The pair (π_b, π_d), instead, is not a span from 3, since π_b and π_d are maximal, but share other nodes than 3. ◇

Intuitively, a choice is well-branched when the participants other than the one opting for alternative runs either behave uniformly in each branch, or can ascertain which branch has been chosen from the messages they receive.

Definition 4.6 (Well-branchedness). *A c-automaton $\mathbb{C}\mathrm{A}$ is* well-branched *if for each state q in $\mathsf{det}(\mathbb{C}\mathrm{A})$ and $\mathsf{A} \in \mathcal{P}$ sender in a transition from q, all of the following conditions must hold:*

(1) all transitions from q involving A, have sender A;
(2) for each transition t from q whose sender is not A and each transition t' from q whose sender is A, t and t' are concurrent
(3) for each q-span (σ, σ') where A chooses at and each participant $\mathsf{B} \neq \mathsf{A} \in \mathcal{P}$, the first pair of different labels on the runs $\sigma\!\downarrow_{\mathsf{B}}$ and $\sigma'\!\downarrow_{\mathsf{B}}$ (if any) is of the form $(\mathsf{C}\,\mathsf{B}?\mathsf{m}, \mathsf{D}\,\mathsf{B}?\mathsf{n})$ with $\mathsf{C} \neq \mathsf{D}$ or $\mathsf{m} \neq \mathsf{n}$.

We dub A a selector at q.

In the above definition loops are taken into account in item (3) since the notion of span is defined in terms of candidate branch. The latter is a maximal run where cycles can be considered at most once, as shown in Example 4.3.

In case of a nondeterministic c-automaton, the conditions of Def. 4.6 are checked after the c-automaton has been determinised. In fact, recalling Remark 3.2, we consider properties of languages of c-automata, and determinisation, as well as minimisation, of FSA preserve languages. Also, both operations preserve the system resulting from projection (cf. Prop. 3.6). (Observe that here we exploit classical results of automata theory.) Also, by Fact 1 and the obvious decidability of the conditions of Defs. 4.4 and 4.6 we get

Fact 2. *Well-branchedness is a decidable property.*

Example 4.7 (Well-branchedness of $\mathbb{C}$ref*).* All the states of $\mathbb{C}$ref satisfy the conditions of Def. 4.6; the only non-trivial cases are states 3 and 4. Condition (1) holds for C, which is the selector of the choice at 3, and for S, which is the selector of the choice at 4; condition (2) holds vacuously, and condition (3) holds for both S and L in all the spans from 3 and from 4. For instance, in the span (π_a, π_f) from 3, described in Example 4.5, the first actions of S on π_a and π_f are the inputs from C which have different messages, whereas, for what concerns L, the condition holds vacuously. As a matter of fact, since π_a and π_f are cofinal in 2, the well-branchedness conditions on state 2 do guarantee L to behave properly afterwards, independently on whether π_a or π_f have been followed before. ◇

Condition (2), vacuously true in our working example, is needed when multiple participants act as sender in the same state: this ensures that the only possibility is that actions of different participants are concurrent so that possible choices at a state are not affected by independent behaviour.

We add a further condition to rule out c-automata having consecutive transitions involving disjoint participants and not actually concurrent.

Definition 4.8 (Well-sequencedness). *A c-automaton* $\mathbb{C}$A *is* well-sequenced *if for each two consecutive transitions* $q \xrightarrow{\mathsf{A}\longrightarrow\mathsf{B}:\,\mathsf{m}} q' \xrightarrow{\mathsf{C}\longrightarrow\mathsf{D}:\,\mathsf{n}} q''$ *either*

- *they share a participant, that is* $\{\mathsf{A},\mathsf{B}\} \cap \{\mathsf{C},\mathsf{D}\} \neq \emptyset$*, or*
- *they are concurrent, i.e. there is* q''' *such that* $q \xrightarrow{\mathsf{C}\longrightarrow\mathsf{D}:\,\mathsf{n}} q''' \xrightarrow{\mathsf{A}\longrightarrow\mathsf{B}:\,\mathsf{m}} q''$*.*

Notice that, by finiteness of the transition relation of c-automata, we get

Fact 3. *Well-sequencedness is a decidable property.*

Notation. For the sake of readability, a well-sequenced c-automaton can be represented by omitting, for each diamond, two of its consecutive transitions. We call such representation *compact*. Notice that, given a compact representation, it is always possible to recover the original c-automaton. So far and hereafter we assume that all c-automata are compactly represented.

Example 4.9 (Well-sequencedness of $\mathbb{C}$ref*).* It is not difficult to check that $\mathbb{C}$ref is well-sequenced because the first condition of Def. 4.8 holds for any pair of consecutive transitions in $\mathbb{C}$ref. ◇

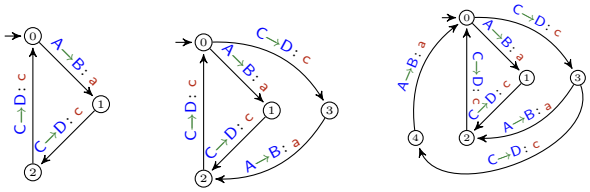


Fig. 2. Failure of well-sequencedness completion.

Well-sequencedness is necessary to establish a precise correspondence between the language of a c-automaton and of its projection (cf. Thm. 4.14 and the discussion following it).

Remark 4.10. We show that not all c-automata can be "completed" to well-sequenced ones. Consider the c-automaton of Fig. 2(a), which is not well-sequenced because of the transitions from state 0 to state 1 and from state 1 to 2. By "completing the diamond" for such transitions (i.e., by adding the new state 3 and the transitions $0 \xrightarrow{\mathsf{C}\to\mathsf{D}:\,\mathsf{c}} 3$ and $3 \xrightarrow{\mathsf{A}\to\mathsf{B}:\,\mathsf{a}} 2$) we obtain the c-automaton of Fig. 2(b). This is still not well sequenced, because of the transitions $3 \xrightarrow{\mathsf{A}\to\mathsf{B}:\,\mathsf{a}} 2$ and $2 \xrightarrow{\mathsf{C}\to\mathsf{D}:\,\mathsf{c}} 0$. So we try to make it well-sequenced by completing the diamond once again and obtain the c-automaton of Fig. 2(c). The resulting c-automaton is still not well-sequenced, because of the transitions $4 \xrightarrow{\mathsf{A}\to\mathsf{B}:\,\mathsf{a}} 0$ and $0 \xrightarrow{\mathsf{C}\to\mathsf{D}:\,\mathsf{c}} 3$. Again a vain attempt, because of the transitions $5 \xrightarrow{\mathsf{A}\to\mathsf{B}:\,\mathsf{a}} 3$ and $3 \xrightarrow{\mathsf{C}\to\mathsf{D}:\,\mathsf{c}} 4$. It is immediate to check that we could go on indefinitely.

It is impossible to complete the initial c-automaton since the intended completed automaton should generate a non-regular language (since it should generate strings with a number of $\mathsf{C}\to\mathsf{D}\colon \mathsf{c}$ interactions which is, roughly, double of the number of $\mathsf{A}\to\mathsf{B}\colon \mathsf{a}$ interactions). It would hence be interesting to know whether, in case the expected completed interaction language of a c-automaton is regular and prefix-closed, it is possible to generate it also by means of a well-sequenced c-automaton. It would be also interesting to establish a condition on cycles (if any) that guarantees the effectiveness of the completion of a c-automaton. We leave these questions for future work. ◇

We show a closure property of the languages of well-sequenced c-automata.

Definition 4.11 (Concurrency closure). *The* swap *relation on choreography words is the smallest equivalence relation* $\_ \sim \_$ *satisfying*

$$w(\mathsf{A}\to\mathsf{B}\colon \mathsf{m})(\mathsf{C}\to\mathsf{D}\colon \mathsf{n})w' \sim w(\mathsf{C}\to\mathsf{D}\colon \mathsf{n})(\mathsf{A}\to\mathsf{B}\colon \mathsf{m})w'$$

where $\{\mathsf{A},\mathsf{B}\} \cap \{\mathsf{C},\mathsf{D}\} = \emptyset$. *Given a choreography language* $\mathcal{L}$

$$close(\mathcal{L}) = \{\, w \in \mathcal{L}_{int} \mid \exists w' \in \mathcal{L}.\ w \sim w' \,\}$$

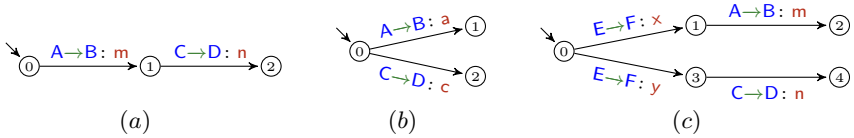


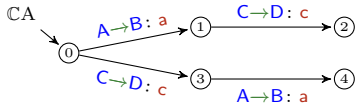
Fig. 3. Well-formedness is necessary for Thm. 4.14.

is the concurrency closure *of* $\mathcal{L}$.

The above relation is reminiscent of the *swapping* relation introduced in [18], with similar aims.

Proposition 4.12. *Let* $\mathbb{C}\mathrm{A}$ *be a well-sequenced c-automaton. Then* $\mathfrak{L}(\mathbb{C}\mathrm{A})$ *is concurrency closed, i.e.* $\mathfrak{L}(\mathbb{C}\mathrm{A}) = \mathit{close}(\mathfrak{L}(\mathbb{C}\mathrm{A}))$.

Notice that the converse of the above proposition does not hold in general. In fact, consider the following c-automaton



we can check that $\mathfrak{L}(\mathbb{C}\mathrm{A}) = \mathit{close}(\mathfrak{L}(\mathbb{C}\mathrm{A}))$ but $\mathbb{C}\mathrm{A}$ is not well-sequenced.

The notion of well-formedness below sums up the requirements needed in order for a c-automaton to be projected to a well-behaved communicating system.

Definition 4.13 (Well-formedness). *A c-automaton is* well-formed *if it is both well-branched and well-sequenced.*

The next result in Thm. 4.14 establishes that the language of a well-formed c-automaton coincides with the language of the communicating system obtained by projection. This provides a correctness criterion for our projection operation.

Theorem 4.14. $\mathfrak{L}(\mathbb{C}\mathrm{A}) = \mathfrak{L}(\llbracket \mathbb{C}\mathrm{A}\!\downarrow \rrbracket^{\mathsf{s}})$ *for any well-formed c-automaton* $\mathbb{C}\mathrm{A}$.

Notice that well-formedness is a necessary condition for the theorem above. It is in fact easy to check that

$$(\mathsf{C}\to\mathsf{D}\colon \mathsf{m})(\mathsf{A}\to\mathsf{B}\colon \mathsf{n}) \in \mathfrak{L}(\llbracket \mathbb{C}\mathrm{A}\!\downarrow \rrbracket^{\mathsf{s}}) \quad \text{and} \quad (\mathsf{C}\to\mathsf{D}\colon \mathsf{m})(\mathsf{A}\to\mathsf{B}\colon \mathsf{n}) \notin \mathfrak{L}(\mathbb{C}\mathrm{A})$$

when $\mathbb{C}\mathrm{A}$ is one of the c-automata (a), (b) or (c) of Fig. 3. In particular, (a) is not well-sequenced whereas (b) and (c) are not well-branched: for (b), item (2) of well-branchedness (Def. 4.6) does not hold; (c) instead violates item (3).

We can now show that the projections of well-formed choreography automata enjoy the communication properties of Def. 2.9.

Theorem 4.15. *Given a well-formed c-automaton* $\mathbb{C}\mathrm{A}$*, its projection* $(\mathbb{C}\mathrm{A}\!\downarrow_{\mathsf{A}})_{\mathsf{A}\in\mathcal{P}}$ *is live, lock-free, and deadlock-free with respect to the synchronous semantics.*

5 Asynchronous Communications

We now transfer the results of the previous sections to the asynchronous semantics of communicating systems (Def. 2.7). Remarkably, the semantics does not affect the definition of c-automata (and of projections) since it is independent of the communication model. Hence, any result depending only on the definition of c-automata still holds. Well-sequencedness instead needs updating.

Definition 5.1 (Asynchronous well-sequencedness). *A c-automaton is* asynchronously well-sequenced *if for each two consecutive transitions* $q \xrightarrow{\mathsf{A}\to\mathsf{B}\colon \mathsf{m}} q' \xrightarrow{\mathsf{C}\to\mathsf{D}\colon \mathsf{n}} q''$ *either*

- *the sender of the second transition occurs in the first one, that is* $\mathsf{C} \in \{\mathsf{A}, \mathsf{B}\}$,
- *or they are concurrent, i.e. there is* q''' *such that* $q \xrightarrow{\mathsf{C}\to\mathsf{D}\colon \mathsf{n}} q''' \xrightarrow{\mathsf{A}\to\mathsf{B}\colon \mathsf{m}} q''$.

Asynchronous well-sequencedness (Def. 4.8) implies the synchronous one. Indeed, asynchronous well-sequencedness requires either two transitions to be concurrent or that the sender of the second transition occurs in the first one. The latter condition is weaker than having disjoint participants as required in the synchronous case.

Note that our working example is well-sequenced but not asynchronously well-sequenced (because e.g., of transitions $2 \xrightarrow{\mathsf{S}\to\mathsf{L}\colon \mathsf{cnt}} 3 \xrightarrow{\mathsf{C}\to\mathsf{S}\colon \mathsf{bye}} 5$). Thus, we now consider it as the *compact* representation of the actual c-automaton according to Notation on page 11.

Unlike well-sequencedness, the notion of well-branchedness has not to be changed in case asynchronous communications are considered. So, in the asyncronous setting, we define *asynchronous well-formedness* as the conjunction of asynchronous well-sequencedness (Def. 5.1) and well-branchedness (Def. 4.6).

The correspondence result between the semantics of a c-automaton and of its projection requires to decide which actions to observe on the projection. Indeed, in a c-automaton, each interaction is seen as an atomic event, while in the asynchronous semantics of communicating systems each interaction corresponds to two events: a sending event and a receving event. We opt to observe sending events only because (internal) choices are determined by sending events. This decision also plays well with the notion of well-branchedness, where most of the conditions concern sender participants. Other possible options are discussed in [35], in a process algebraic setting. This idea is formalised by *sender traces*.

Definition 5.2 (Sender traces). *The* sender traces $\mathfrak{L}^!(\mathsf{S})$ *of a communicating system* S *are obtained from its asynchronous traces by replacing each output label* $\mathsf{A}\,\mathsf{B}!\mathsf{m}$ *with* $\mathsf{A}\to\mathsf{B}\colon \mathsf{m}$ *and each input label* $\mathsf{A}\,\mathsf{B}?\mathsf{m}$ *with* ε.

The modification of well-sequencedness for the asynchronous case does imply that we need to "update" the definition of concurrency closure as well.

Definition 5.3 (Asynchronous concurrency closure). *The* asynchronous swap *relation on choreography words is the smallest pre-order* $\_ \leq \_$ *satisfying*

$$w(\mathsf{A}{\to}\mathsf{B}\colon \mathsf{m})(\mathsf{C}{\to}\mathsf{D}\colon \mathsf{n})w' \leq w(\mathsf{C}{\to}\mathsf{D}\colon \mathsf{n})(\mathsf{A}{\to}\mathsf{B}\colon \mathsf{m})w' \qquad \text{where } \mathsf{C} \notin \{\mathsf{A},\mathsf{B}\}.$$

The downward closure of a choreography language $\mathcal{L}$ *with respect to* $\_ \leq \_$

$$close_{\mathrm{a}}(\mathcal{L}) = \{\, w \in \mathcal{L}_{int} \mid \exists w' \in \mathcal{L}.\ w \leq w' \,\}$$

is the asynchronous concurrency closure *of* $\mathcal{L}$.

The condition for asynchronous concurrency closure is weaker than the one in the synchronous case. This is due to the fact that sender-traces must be closed under asynchronous concurrency (cf. Lemma 5.4 below), so to guarantee that the traces of an automaton do coincide with the sender-traces of its projection (Thm. 5.6 below). We discuss such a necessity with an example after Thm. 5.6.

Lemma 5.4. *Let* $\mathbb{C}\mathrm{A}$ *be a c-automaton. Then* $\mathfrak{L}^{!}(\llbracket \mathbb{C}\mathrm{A}{\downarrow} \rrbracket^{\mathrm{a}}) = close_{\mathrm{a}}(\mathfrak{L}^{!}(\llbracket \mathbb{C}\mathrm{A}{\downarrow} \rrbracket^{\mathrm{a}}))$.

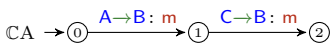
We now proceed to prove the correctness of projection for asynchronous systems. We will reduce it to the corresponding result for synchronous systems (Thm. 4.14). This is done by showing that all asynchronous runs are pairable (see below), that is they can be put in a suitable normal form which directly corresponds to a synchronous run. Notably, such a result is false for c-automata which are not asynchronously well-formed.

Definition 5.5 (Pairable runs). *Let* $\mathbb{C}\mathrm{A}$ *be a c-automaton. A run* σ *in* $\llbracket \mathbb{C}\mathrm{A}{\downarrow} \rrbracket^{\mathrm{a}}$ *is paired into a run* σ' *in* $\llbracket \mathbb{C}\mathrm{A}{\downarrow} \rrbracket^{\mathrm{a}}$ *iff they are coinitial, produce the same sender trace, and each output* $\mathsf{A}\,\mathsf{B}!\mathsf{m}$ *in* σ' *is immediately followed by the corresponding input* $\mathsf{A}\,\mathsf{B}?\mathsf{m}$. *A run* σ *is* pairable *if it is paired into a run* σ'.

Theorem 5.6. *Let* $\mathbb{C}\mathrm{A}$ *be an asynchronously well-formed c-automaton.*

$$\mathfrak{L}^{!}(\llbracket \mathbb{C}\mathrm{A}{\downarrow} \rrbracket^{\mathrm{a}}) = \mathfrak{L}(\mathbb{C}\mathrm{A})$$

Similarly to Thm. 4.14, asynchronous well-formedness is a necessary condition for Thm. 5.6. Examples (*b*) and (*c*) of Fig. 3 work the same also for the asynchronous case, since we do not changed the definition of well-branchedness. We changed instead the definition of well-sequencedness to a stricter version and the c-automaton (*a*) of Fig. 3 is hence not enough to show the necessity of asyncronous well-sequencedness; this can however be easily done using the following c-automaton which is well-sequenced but not asynchronouly well-sequenced



Since outputs of asynchronous CFSMs can always be fired, there is a run of the projected system beginning with $\mathsf{C}\,\mathsf{B}!\mathsf{m}$ and producing the sender trace $(\mathsf{C}{\to}\mathsf{B}\colon \mathsf{m})(\mathsf{A}{\to}\mathsf{B}\colon \mathsf{m}) \in \mathfrak{L}^{!}(\llbracket \mathbb{C}\mathrm{A} \rrbracket^{\mathrm{a}})$ which trivially does not belong to $\mathfrak{L}(\mathbb{C}\mathrm{A})$ because the interactions cannot be swapped (cf. Def. 5.3).

The communication properties for projected systems can also be obtained.

Theorem 5.7. *Given an asynchronous well-formed c-automaton* $\mathbb{C}\mathrm{A}$, *its projection* $(\mathbb{C}\mathrm{A}{\downarrow_{\mathsf{A}}})_{\mathsf{A}\in\mathcal{P}}$ *is live, lock-free, and deadlock-free with respect to the asyncronous semantics.*

6 Conclusion, Related Work and Future Work

We introduced a model of choreographies based on FSAs whose transitions are labelled by interactions. We showed relevant results both for a synchronous and an asynchronous underlying communication infrastructure. We established a correspondence between the language of an automaton and the one of its projection, as well as proofs of liveness, lock, and deadlock freedom for the latter.

The adoption of an automata-based model brings in two main benefits. Firstly, the constructions that we provided are based on set-theoretic notions and are *syntax-independent*. This contrasts with syntax-driven models (such as behavioural type systems [30]) where expressiveness may be limited and definitions may be more complex due to syntactic reasons. E.g., the example in § 1 cannot be modelled in many behavioural type systems since entangled loops cannot be represented using a recursion operator. Secondly, we can re-use well-known results of the theory of automata (e.g., we used notions of trace equivalence and determinisation) and related tools.

Related work Automata-based models for specifying the local behaviour of distributed compontents are commonplace in the literature (see e.g., [13,21]). Less so is for the global specifications of choreographies: to the best of our knowledge, the conversation protocols (CP) in [26,27,9] (and references therein) are the only such model in the literature. The realisability of CP has been first studied in [27]; this is indeed the work closest to ours. Conversation protocols are non-deterministic Büchi automata whose labels resemble our interactions (barred the fact that, contrarily to our formalism, in [27] the sender and the receiver of each message are determined by its content). Our c-automata are basically finite-state automata where infinite words can be taken into account by looking at them as Büchi automata where all states are actually final. It is not immediate to provide a detailed comparison between conversation protocols and c-automata because their semantics and underlying communication models differ. As for the communication model, conversation protocols are realised in a subclass of CFSMs (cf. Section 5 of [27]), whereas we consider the unrestricted model of CFSMs, as well as a synchronous version of it. Concerning the semantics, Definition 4 (item 3(b)) of [27] restricts the runs to those where all messages in queues are eventually consumed, that is they require by definition a form of liveness. Instead, one of our goals is to identify conditions that guarantee relevant liveness properties. We prove them in Theorem 5.7, and in Prop. D.1 in [7] we prove the exact property assumed in [27]. The realisability conditions of conversation protocols are *lossless join*, *synchronous compatibility*, and *autonomy*. Those conditions cannot easily be compared with well-formedness, due to the differences in the models and in the semantics. Furthermore, the style of the conditions is very different, and it also induces very different proof strategies in many cases. In particular,

- our well-sequencedness is checked on pairs of consecutive transitions and well-branchedness on pairs of coinitial paths;
- lossless join is a global property, that is a condition on the automaton consisting of the product of the languages of the local projections;

– synchronous compatibility is defined in terms of pairs of traces in the projection but verified with an algorithm that checks a global property of an automata costruction, and the same holds for autonomy.

Thus, while the conditions capture similar intuitions, a detailed comparison is very hard. When restricting to the common part of the two models, well-branchedness implies autonomy while the opposite does not hold. Indeed, by well-branchedness the selector is output-ready (according to the terminology in [27]), while any other participant either behaves uniformly in each branch (and is thus either input-ready or output-ready or termination-ready) or it is made aware of the choice by distinct inputs (that is it is input-ready). In all the cases autonomy is satisfied. In the other direction, a choice between traces $(\mathsf{A}{\to}\mathsf{B}\colon \mathsf{l})(\mathsf{B}{\to}\mathsf{C}\colon \mathsf{n})(\mathsf{C}{\to}\mathsf{D}\colon \mathsf{w})$ and $(\mathsf{A}{\to}\mathsf{B}\colon \mathsf{r})(\mathsf{B}{\to}\mathsf{C}\colon \mathsf{n})(\mathsf{C}{\to}\mathsf{D}\colon \mathsf{z})$ satisfies autonomy but not well-branchedness.

As for lossless join, we do not assume it. Actually, it is equivalent to one of our results, namely the correctness of the projection in the synchronous case (Thm. 4.14). Such a result is also used in the asynchronous case (Thm. 5.6), which is proved by reduction to the synchronous one via paired runs. We leave a detailed comparison of the two sets of constraints, in a common setting, for future work. Later works on CP (see, e.g., [9]) changed the approach and relied on model checking to show realisability instead of well-formedness conditions. Unfortunately, some of their main decidability results were flawed [24].

Conditions similar to well-branchedness and well-sequencedness do naturally arise in investigations related to choreoghraphies and their realisability. A unique sender driving a choice is a condition present in several multiparty session types formalisms ([28] and [20] to cite just a couple of them), global graphs formalisms [48], choreography languages in general (for instance see the notion of *dominant role* in [44]). Conditions related to item (3) of Def. 4.6 can also be found in multiparty session types formalisms [46] or in conversation protocols, as discussed above. Also, notions close to well-sequencedness turn out to arise quite naturally in "well-behaved" choreographies (see for instance the notion of *well-informedness* of [15] in the context of collaboration diagrams).

Similarly to what discussed in Remark 4.10, some approaches propose techniques to fix choreographies which are not well-behaved. This issue is considered in some multiparty session types [11,10], in algebraic and automata-based frameworks for choreographies [36,8] as well as in the choreographic middleware ChoreOS [3,4]. While they consider different conditions than ours, trying to adapt their approaches to our setting is an interesting item for future work.

As said, most approaches are not based on automata. For instance, [44,35,22] use algebraic operators to build larger choreographies from smaller ones, and give conditions on such operations ensuring that the resulting choreography is "well-behaved". This technique is not applicable in our case, since, like most works on automata, we do not consider an algebra to build automata.

While the main aim of c-automata is to provide a choreography model based on FSAs, we remark here that it is rather expressive and complements existing models of choreographies or multiparty session types (MST). In particular, the expressive power of c-automata is not comparable with the one of the MST in [45],

which subsumes most systems in the literature. More precisely, the c-automaton $\mathbb{C}$ref in § 1 cannot be syntactically written in [45] due to the two entangled loops. That example cannot be expressed in global graphs [48] either, again due to the intersecting loops. We note that the infinite unfolding of the c-automaton is regular and therefore it would fit in the session type system considered in [47]. However, this type system has not been conceived for choreographies (it is a binary session type system) and does not allow non-determinism.

On the other side, examples such as [45, Ex. 2, Fig. 4] cannot be written in our model (since we expect the same roles to occur in branches which are coinitial, branches inside loops require that all participants in a loop are notified when the loop ends). We conjecture that a refinement of well-branchedness is possible to address this limitation. Global graphs are another model of global specifications. Their advantage is that they feature parallel composition, which c-automata lack. We note however that one could use the classical product of automata on c-automata to model parallel composition in the case where the two branches have disjoint sets of participants (as typically assumed in MST with parallel composition). Mapping global graphs without parallel composition into c-automata is trivial. The same considerations apply to choreography languages where possible behaviours are defined by a suitable process algebra with parallel composition such as [35,14].

Future work One of the main motivations to develop a choreography model based on automata was to lift the compositional mechanism discovered in [6] on CFSMs to global specifications, in such a way that composition of global specifications preserves well-formedness. This is the problem we are currently addressing.

An interesting future development is also to adopt Büchi automata as c-automata. This extension is technically straightforward (just add accepting states to Def. 3.1 and define ω-languages accordingly), but it probably impacts greatly the underlying theory. An interesting yet not trivial effort is the identification of well-formedness conditions on this generalised class of c-automata that guarantee a precise correspondence with the ω-languages of the projections.

The interplay between FSAs and formal languages could lead to a theory of projection of choreographies based on languages instead of automata. For instance, one could try to characterise the languages accepted by well-formed c-automata, similarly to what done in [1,38,48]. In those approaches global specifications are rendered as partial orders and the distributed realisability is characterised in terms of closure properties of languages.

A final direction for future work concerns the implementation of tool support for the approach. We are currently working in this direction. A very preliminary and partial implementation by Simone Orlando and Ivan Lanese is available at `https://github.com/simoneorlando/Corinne`.

References

1. Rajeev Alur, Kousha Etessami, and Mihalis Yannakakis. Inference of Message Sequence Charts. *IEEE Trans. Software Eng.*, 29(7):623–633, 2003.

2. W. Ariola and C. Dunlop. Testing in the API Economy. Top 5 Myths. `https://api2cart.com/api-technology/api-testing-myths-infographic/`.
3. Marco Autili, Paola Inverardi, and Massimo Tivoli. Choreography realizability enforcement through the automatic synthesis of distributed coordination delegates. *Sci. Comput. Program.*, 160:3–29, 2018.
4. Marco Autili, Davide Di Ruscio, Amleto Di Salle, and Alexander Perucci. Choreosynt: enforcing choreography realizability in the future internet. In Shing-Chi Cheung, Alessandro Orso, and Margaret-Anne D. Storey, editors, *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, (FSE-22), Hong Kong, China, November 16 - 22, 2014*, pages 723–726. ACM, 2014.
5. Marco Autili, Amleto Di Salle, Francesco Gallo, Claudio Pompilio, and Massimo Tivoli. Chorevolution: Automating the realization of highly-collaborative distributed applications. In *COORDINATION*, pages 92–108, 2019.
6. Franco Barbanera, Ugo de'Liguoro, and Rolf Hennicker. Connecting open systems of communicating finite state machines. *J. Log. Algebraic Methods Program.*, 109, 2019.
7. Franco Barbanera, Ivan Lanese, and Emilio Tuosto. Choreography automata. `http://www.cs.unibo.it/~lanese/work/TR/choreography_automata.pdf`, April 2020. Full version.
8. Samik Basu and Tevfik Bultan. Automated choreography repair. In Perdita Stevens and Andrzej Wasowski, editors, *FASE*, volume 9633 of *Lecture Notes in Computer Science*, pages 13–30. Springer, 2016.
9. Samik Basu, Tevfik Bultan, and Meriem Ouederni. Deciding choreography realizability. In *Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012, Philadelphia, Pennsylvania, USA, January 22-28, 2012*, pages 191–202, 2012.
10. Laura Bocchi, Julien Lange, and Emilio Tuosto. Amending contracts for choreographies. In *ICE*, volume 59 of *EPTCS*, pages 111–129, 2011.
11. Laura Bocchi, Julien Lange, and Emilio Tuosto. Three algorithms and a methodology for amending contracts for choreographies. *Sci. Ann. Comp. Sci.*, 22(1):61–104, 2012.
12. Jonas Bonér. *Reactive Microsystems - The Evolution Of Microservices At Scale.* O'Reilly, 2018.
13. Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983.
14. Mario Bravetti and Gianluigi Zavattaro. Towards a unifying theory for choreography conformance and contract compliance. In Markus Lumpe and Wim Vanderperren, editors, *Software Composition, 6th International Symposium, SC 2007, Braga, Portugal, March 24-25, 2007, Revised Selected Papers*, volume 4829 of *Lecture Notes in Computer Science*, pages 34–50. Springer, 2007.
15. Tevfik Bultan and Xiang Fu. Specification of realizable service conversations using collaboration diagrams. *Service Oriented Computing and Applications*, 2(1):27–39, 2008.
16. Tevfik Bultan, Xiang Fu, Richard Hull, and Jianwen Su. Conversation specification: a new approach to design and analysis of e-service composition. In Gusztáv Hencsey, Bebo White, Yih-Farn Robin Chen, László Kovács, and Steve Lawrence, editors, *Proceedings of the Twelfth International World Wide Web Conference, WWW 2003, Budapest, Hungary, May 20-24, 2003*, pages 403–410. ACM, 2003.

17. Marco Carbone, Kohei Honda, and Nobuko Yoshida. Structured communication-centred programming for web services. In Rocco De Nicola, editor, *ESOP*, volume 4421 of *Lecture Notes in Computer Science*, pages 2–17. Springer, 2007.
18. Marco Carbone and Fabrizio Montesi. Deadlock-freedom-by-design: multiparty asynchronous global programming. In Roberto Giacobazzi and Radhia Cousot, editors, *POPL*, pages 263–274. ACM, 2013.
19. Gérard Cécé and Alain Finkel. Verification of programs with half-duplex communication. *I&C*, 202(2):166–190, 2005.
20. Mario Coppo, Mariangiola Dezani-Ciancaglini, Nobuko Yoshida, and Luca Padovani. Global progress for dynamically interleaved multiparty sessions. *Mathematical Structures in Computer Science*, 26(2):238–302, 2016.
21. Luca de Alfaro and Thomas A. Henzinger. Interface automata. In *Proceedings of the 8th European Software Engineering Conference held jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering 2001, Vienna, Austria, September 10-14, 2001*, pages 109–120, 2001.
22. Pierre-Malo Deniélou and Nobuko Yoshida. Dynamic multirole session types. In Thomas Ball and Mooly Sagiv, editors, *POPL*, pages 435–446. ACM, 2011.
23. Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty session types meet communicating automata. In Helmut Seidl, editor, *Programming Languages and Systems - 21st European Symposium on Programming, ESOP 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, volume 7211 of *Lecture Notes in Computer Science*, pages 194–213. Springer, 2012.
24. Alain Finkel and Étienne Lozes. Synchronizability of communicating finite state machines is not decidable. In *ICALP*, pages 122:1–122:14, 2017.
25. Adrian Francalanza, Claudio Antares Mezzina, and Emilio Tuosto. Reversible choreographies via monitoring in erlang. In Silvia Bonomi and Etienne Rivière, editors, *Distributed Applications and Interoperable Systems - 18th IFIP WG 6.1 International Conference, DAIS 2018, Held as Part of the 13th International Federated Conference on Distributed Computing Techniques, DisCoTec 2018, Madrid, Spain, June 18-21, 2018, Proceedings*, volume 10853 of *Lecture Notes in Computer Science*, pages 75–92. Springer, 2018.
26. Xiang Fu, Tevfik Bultan, and Jianwen Su. Conversation protocols: A formalism for specification and verification of reactive electronic services. In Oscar H. Ibarra and Zhe Dang, editors, *Implementation and Application of Automata, 8th International Conference, CIAA 2003, Santa Barbara, California, USA, July 16-18, 2003, Proceedings*, volume 2759 of *Lecture Notes in Computer Science*, pages 188–200. Springer, 2003.
27. Xiang Fu, Tevfik Bultan, and Jianwen Su. Conversation protocols: a formalism for specification and verification of reactive electronic services. *Theor. Comput. Sci.*, 328(1-2):19–37, 2004.
28. Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In George C. Necula and Philip Wadler, editors, *POPL*, pages 273–284. ACM Press, 2008.
29. Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. *Journal of the ACM*, 63(1):9:1–9:67, 2016. Extended version of a paper presented at POPL08.
30. Hans Hüttel, Ivan Lanese, Vasco T. Vasconcelos, Luís Caires, Marco Carbone, Pierre-Malo Deniélou, Dimitris Mostrous, Luca Padovani, António Ravara, Emilio Tuosto, Hugo Torres Vieira, and Gianluigi Zavattaro. Foundations of session types and behavioural contracts. *ACM Comput. Surv.*, 49(1):3:1–3:36, 2016.

31. Nickolas Kavantzas, David Burdett, Gregory Ritzinger, Tony Fletcher, Yves Lafon, and Charlton Barreto. Web services choreography description language version 1.0. Technical report, W3C, 2005. `http://www.w3.org/TR/ws-cdl-10/`.
32. Naoki Kobayashi. A Partially Deadlock-Free Typed Process Calculus. *ACM TOPLAS*, 20(2):436–482, 1998.
33. Naoki Kobayashi. Type-Based Information Flow Analysis for the Pi-Calculus. *Acta Informatica*, 42(4–5):291–347, 2005.
34. Ajay Krishna, Pascal Poizat, and Gwen Salaün. Checking business process evolution. *Sci. Comput. Program.*, 170:1–26, 2019.
35. Ivan Lanese, Claudio Guidi, Fabrizio Montesi, and Gianluigi Zavattaro. Bridging the gap between interaction- and process-oriented choreographies. In *Software Engineering and Formal Methods, SEFM 2008*, pages 323–332, 2008.
36. Ivan Lanese, Fabrizio Montesi, and Gianluigi Zavattaro. Amending choreographies. In António Ravara and Josep Silva, editors, *WWV*, volume 123 of *EPTCS*, pages 34–48, 2013.
37. Julien Lange, Emilio Tuosto, and Nobuko Yoshida. From communicating machines to graphical choreographies. In Sriram K. Rajamani and David Walker, editors, *POPL*, pages 221–232. ACM, 2015.
38. Markus Lohrey. Safe Realizability of High-Level Message Sequence Charts. In Luboš Brim, Mojmír Křetínský, Antonín Kučera, and Petr Jančar, editors, *CONCUR*, LNCS, pages 177–192. Springer, 2002.
39. Fabrizio Montesi. *Choreographic Programming*. PhD thesis, University of Copenhagen, 2013.
40. OMG. *Business Process Model and Notation (BPMN), Version 2.0*, January 2011. `https://www.omg.org/spec/BPMN`.
41. Luca Padovani, Vasco Thudichum Vasconcelos, and Hugo Torres Vieira. Typing liveness in multiparty communicating systems. In *COORDINATION*, volume 8459 of *LNCS*, pages 147–162. Springer, 2014.
42. Robert Paige and Robert Tarjan. Three Partition Refinement Algorithms. *SIAM Journal on Computing*, 16(6):973–989, December 1987.
43. Mila Dalla Preda, Maurizio Gabbrielli, Saverio Giallorenzo, Ivan Lanese, and Jacopo Mauro. Dynamic choreographies: Theory and implementation. *Logical Methods in Computer Science*, 13(2), 2017.
44. Zongyan Qiu, Xiangpeng Zhao, Chao Cai, and Hongli Yang. Towards the theoretical foundation of choreography. In Carey L. Williamson, Mary Ellen Zurko, Peter F. Patel-Schneider, and Prashant J. Shenoy, editors, *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007*, pages 973–982. ACM, 2007.
45. Alceste Scalas and Nobuko Yoshida. Less is more: multiparty session types revisited. *PACMPL*, 3(POPL):30:1–30:29, 2019.
46. Paula Severi and Mariangiola Dezani-Ciancaglini. Observational Equivalence for Multiparty Sessions. *Fundamenta Informaticae*, 170:267–305, 2019.
47. Paula Severi, Luca Padovani, Emilio Tuosto, and Mariangiola Dezani-Ciancaglini. On sessions and infinite data. *Logical Methods in Computer Science*, 13(2), 2017.
48. Emilio Tuosto and Roberto Guanciale. Semantics of global view of choreographies. *J. Log. Algebr. Meth. Program.*, 95:17–40, 2018.