Journal of Logical and Algebraic Methods in Programming 85 (2016) 425–446

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

www.elsevier.com/locate/jlamp

Relating two automata-based models of orchestration and choreography ☆

D. Basile [a,*], P. Degano [a,*], G.L. Ferrari [a,*], E. Tuosto [b,*]

[a] Dipartimento di Informatica, Università di Pisa, Italy
[b] Computer Science Department, University of Leicester, United Kingdom

ARTICLE INFO

Article history:
Received 15 October 2014
Received in revised form 16 September 2015
Accepted 24 September 2015
Available online 12 October 2015

ABSTRACT

We investigate the relations between two automata-based models for describing and studying distributed services, called contract automata and communicating machines. In the first model, distributed services are abstracted away as automata – oblivious of their partners – that coordinate with each other through an orchestrator. The second one is concerned with the interactions occurring between distributed services, that are represented by channel-based asynchronous communications; then services are coordinated through choreography.

We define a notion of strong agreement on contract automata; exhibit a natural mapping from this model to communicating machines with a synchronous semantics; and give conditions to ensure that strong agreement corresponds to well-formed choreography. Then these results are extended to a more liberal notion of agreement and to a fully asynchronous semantics of communicating machines.

1. Introduction

Among the most popular approaches to the design of distributed coordination, *orchestration* and *choreography* aim to describe the distributed workflow (in other words, how control and data exchanges are coordinated in distributed applications or systems). Intuitively, orchestration yields the description of a distributed workflow from "one party's perspective" [1], whereas choreography describes the behaviour of involved parties from a "*global viewpoint*" [2]. In an orchestrated model, the distributed computational components coordinate with each other by interacting with a special component, *the orchestrator*, which at run time dictates how the computation evolves. In a choreographed model, the distributed components autonomously execute and interact with each other on the basis of a local control flow expected to comply with their role as specified in the "global viewpoint". Here we investigate the relations between two models of distributed coordination: *contract automata* [3] and *communicating machines* [4].

The first model has been recently introduced as a *contract-based* coordination framework where contracts specify the expected behaviour of distributed components oblivious of their communicating partners. The underlying coordination mechanism of contract automata is *orchestration*. In this model a component, or *principal*, is assumed to communicate

☆ This work has been partially supported by the MIUR project *Security Horizons* and IST-FP7-FET open-IP project *ASCENS*, by the European Union Seventh Framework Programme under grant agreement No. 295261 *Mobility between Europe and Argentina applying Logics to Systems* (MEALS), and by the COST Action IC1201: *Behavioural Types for Reliable Large-Scale Software Systems* (BETTY).

* Corresponding authors.
E-mail addresses: basile@di.unipi.it (D. Basile), degano@di.unipi.it (P. Degano), giangi@di.unipi.it (G.L. Ferrari), emilio@le.ac.uk (E. Tuosto).

http://dx.doi.org/10.1016/j.jlamp.2015.09.011
2352-2208/

messages on some ports according to an automaton specifying its behavioural contract. These messages have to be thought of as directed to an orchestrator synthesised out of the components; the orchestrator directs the interactions in such a way that only executions that "are in agreement" actually happen. In this way, it is possible to transfer the approach of [5,6] to contract automata so to identify misbehaviour of components that do not realise their contract.

We illustrate this with the following simple example. Alice is willing to lend her aeroplane toy, Bob offers a bike toy in order to play with an aeroplane toy, while Carol wants to play with an aeroplane or a bike toy. Let $\bar{a}$ and $\bar{b}$ denote respectively the actions of offering an aeroplane or a bike toy and, dually, a and b denote the corresponding request actions. The contract automata for the principals Alice, Bob, and Carol correspond to the following regular expressions, used for conciseness from here onwards:

$$\texttt{Alice} = \bar{a} \qquad \texttt{Bob} = \bar{b}.a + a.\bar{b} \qquad \texttt{Carol} = a + b$$

If Alice exchanges her toy with Bob, then all contracts are fulfilled. Instead, if not coordinated, Alice, Bob, and Carol may share their toys in a way that does not fulfil their contracts. Indeed, Alice can give her aeroplane to Bob or Carol, while Carol can receive the bike from Alice or Bob. Now, if Alice gives her aeroplane to Carol the contracts of Alice and Carol are fulfilled, while Bob's is not. In the model of contract automata the coordinator acts as the mother of the three kids who takes their desires and suggests how to satisfy them (and reproaches those who do not act according to their declared contract).

The other model we consider here are *communicating machines*, a sort of automata that can exchange messages through asynchronous (FIFO) buffers. A set of machines constitutes a communicating system. This model has been introduced to study distributed communication protocols and to ensure the correctness of distributed components. Unlike contract automata, communicating machines require no orchestrator since they directly interact with each other through the buffers. Recently, a relation between communicating machines and distributed choreographies has been proved in [7].

We show that the above two models are related, in spite of the different problems they address and the different mechanisms they use for coordination. For this purpose, we first introduce the notion of *strong agreement* on contract automata, that requires the fulfilment of *all* offers *and* requests, differently than previously introduced notions [3], that we will deal with later on. We then introduce *strongly safe* contract automata, that only accept computations that are in strong agreement.

We consider *convergent* communicating systems, that is those exhibiting successfully terminating computations only. The machines of convergent systems respect their contracts, namely they accomplish their tasks and receive what they look for. Safety and convergence are key notions for semantically linking contract automata and communicating machines. We proceed as follows. We first define a mapping for translating (each principal of) a contract automaton in the corresponding (machine composing a) communication system, and we prove that the computations of the two correspond in a precise way. In the beginning we endow communicating systems with a synchronous semantics, while the general case is considered in Section 6.2.

Then, if a contract automaton is strongly safe the corresponding communicating system is convergent, and vice versa. It should be noted that a given contract automaton can always be transformed into one that is strongly safe, called its (most permissive) *controller*, by adapting standard constructions from the supervisory control theory for discrete event systems [8], as done in [3].

To be more precise, we first establish the above semantic connection by requiring the buffers of communicating machines to contain at most one message and contract automata to well-behave on branching constructs (a notion made precise below).

Later on, we recall the notion of *agreement* on contract automata [3], that only requires the fulfilment of *all* requests. In a sense, this notion allows a contract automaton to be compositionally placed in an unknown environment that may accept the unmatched offers of the automaton. Again, contract automata in agreement and well-behaving on branches are in bijection with the corresponding convergent communicating systems.

Finally, we consider the fully asynchronous semantics of communicating machines, and we prove a weaker result: a strongly safe contract automaton satisfying tighter constraints on branches represents a convergent communicating system.

This paper extends our contribution in [9]. Besides Section 2 on related work, the main new parts concern the notion of agreement just mentioned above, and the new correspondence results for communicating machines with the general semantics, both included in Section 6. Although most of the technicalities on contract automata are inherited from [3], the notion of strong agreement has been introduced here. Finally, we have also revised and simplified some proofs.

Structure of the paper. Section 2 briefly surveys the related work closer to ours. We recall contract automata and communicating finite-state machines in Section 3. Section 4 introduces our new notion of strong agreement on contract automata and the way their controllers are derived. The translation of contract automata into communicating machines is given in Section 5, where we also prove our main theorem of correspondence. In Section 6 we extend our results by relaxing the constraints put on both kinds of automata, i.e. we consider agreement on contract automata and the fully asynchronous semantics for communicating machines. Section 7 works out in full detail an example. Finally, Section 8 concludes, and discusses possible extensions of our results.

2. On choreography and orchestration: related work

To better cast our results within the literature, we first clarify what we mean by choreography and orchestration since, to the best of our knowledge, precise definitions of those concepts are still missing and only intuitive descriptions have been given so far.

There is common consensus that the distinguishing element of a choreographic model is the specification of a so-called *global viewpoint* detailing the interactions among distributed participants and offering a "contract" about their expected communication behaviour in terms of message exchanges. This intuition is best described in W3C words [2]:

> *Using the Web Services Choreography specification, a contract containing a global definition of the common ordering conditions and constraints under which messages are exchanged, is produced that describes, from a global viewpoint [...] observable behaviour [...]. Each party can then use the global definition to build and test solutions that conform to it. The global specification is in turn realised by combination of the resulting local systems [...]*

Noteworthy, the excerpt above points out that local behaviour should then be realised by conforming to the global viewpoint in a "top-down" fashion. Hence, the relations among the global and local specifications are paramount. These aspects are addressed in [10] through an analysis of the relations between the *interaction-oriented* choreographies (i.e., global specifications expressed as interactions) and the *process-oriented* ones (i.e., the local specifications expressed as process algebra terms). A different "bottom-up" approach has been recently introduced in [11] that synthesise choreographies from local specifications. This makes choreography models more flexible (for instance, choreographies have been exploited in [12] as a contract model for service composition). Note that – adapting the terminology of [10] – we use communicating machines as *automata-oriented* choreography, as in [11].

The concept of orchestration is more controversial. In this paper we adopt a widely accepted notion of orchestration [13–15] envisaging the distributed coordination of services as mediated by a distinguished participant that – besides acting as provider of some functionalities – regulates the control flow by exchanging messages with partner services according to their exposed communication interface. In Peltz's words [1]:

> *Orchestration refers to an executable business process that can interact with both internal and external Web services. The interactions occur at the message level. They include business logic and task execution order, and they can span applications and organisations to define a long-lived, transactional, multi-step process model. [...] Orchestration always represents control from one party's perspective.*

The "executable process" mentioned by Peltz is called *orchestrator* and specifies the workflow from the "one party's perspective" describing the interactions with other available services, so to yield a new composed service. This description accounts for a composition model enabling developers to combine existing and independently developed services. The orchestrator then "glues" them together in order to realise a new service, as done for instance in Orc [16]. This is a remarkable aspect since the services combined by an orchestrator are not supposed to have been specifically designed for the service provided by the orchestrator and can in fact be (re)used by other orchestrators for realising different services. Notice that this approach differs from the "bottom-up" one of [11], because synthesised choreographies do not correspond to executable orchestrators.

Other authors consider orchestration as the description of message exchanges among participants from the single participants' viewpoint *without assuming the presence of an orchestrator*. For instance, in [17,18] the local specifications of a choreography are considered the orchestration model of the choreography itself. We believe that this acceptation is too lax because any distributed application consists of parties that exchange information at will (no matter if realised with channel communication, remote method invocation, etc.). Considering each local specification of a choreography as an orchestration may obscure the matter; rather local specifications are tailored to (and dependent of) the corresponding party of the choreography instead of begin independently designed. In other words, for us such local specifications correspond to automata-oriented choreography adopted here and in [11], as well as to the process-oriented choreography of [10]. Instead, in an orchestration model, each participant defines and exposes its own communication pattern which is then (somehow) assembled in an orchestration. We model orchestrators as contract automata and, abstracting from technological aspects, we can describe our approach using Ross-Talbot's words [19]:

> *In the case of orchestration we use a service to broker the interactions between the services including the human agents and in the case of choreography we define the expected observable interactions between the services as peers as opposed to mandating any form of brokering.*

The dichotomy orchestration–choreography has been discussed in several papers (see e.g., [1]). The only formal results we are aware of that tightly link a choreography to an orchestration framework are in [20]; this paper uses a bisimulation to establish a conformance relation (or its absence) between choreographed and orchestrated computations. We note that the orchestration model of [20] – unlike ours – envisages systems as the parallel composition of orchestrators.

We instead devise conditions to correlate orchestrated computations with local specifications of choreographies so to ensure that the former well-behave and correspond to communication-safe choreographies. (We remark that the "liberal" execution of orchestrated computations not enjoying our conditions would in general be non-compliant with the corresponding choreographic model.) A practical outcome of our results is that contract automata enjoying (strong) agreement can execute without controller, if they are trusted. This yields the further advantage that contract automata are translated into communicating machines that run without any central control: disposing the orchestrator reduces the communication overhead.

The idea behind our notions of agreement is to require, in a composition of participants, the existence of possible "good" executions together with "bad" computations, that the orchestrator eventually cuts off, while the literature has also notions that require composition of participants to have "good" computations, only. Further research problems have been identified and investigated for distributed choreographies, for instance realizability [21,22,17,23], conformance [24–27], or enforcement [28]. Our work departs from the existing literature in that we does not require to start from a global description of the interactions. In fact, we simply assume that each service specifies the interactions it is involved in (oblivious of other partners). Then, we identify (i) the conditions to allow the coordination of a set of services to satisfy each local requirement, and (ii) the conditions for removing the central orchestrator.

3. Background

This section summarises the automata models we use in the paper. Both models envisage distributed computations as enacted by components that interact by exchanging messages. As we will see, in both cases components, abstracted away as automata, yield systems also formalised as automata.

3.1. Contract automata

Before recalling contract automata [3], we fix our notations and preliminary definitions.

Given a set X, as usual, $X^* \stackrel{\text{def}}{=} \bigcup_{n\geq 0} X^n$ is the set of finite *words* on X (ε is the empty word, ww' is the concatenation of words $w, w' \in X^*$, $w_{(i)}$ denotes the i-th symbol of w, and $|w|$ is the length of w); write x^n for the word obtained by n concatenations of $x \in X$ and x^* for a finite and arbitrarily long repetition of $x \in X$. It will also be useful to consider X^n as a set of tuples and let $\vec{x}$ to range over it. Sometimes, overloading notation (and terminology), we confound tuples on X with words on X (e.g., if $\vec{w} \in X^n$, then $|\vec{w}| = n$ is the length of w and $\vec{w}_{(i)}$ denotes the i-th element of w).

A contract automaton (cf. Definition 2 below) represents the behaviour of a set of principals (possibly a singleton) capable of performing some *actions*; more precisely, as formalised in Definition 1, the actions of contract automata allow them to "advertise" offers, "make" requests, or "handshake" on simultaneous offer/request actions. Consequently, transitions of contract automata will be labelled with tuples of elements in the set $\mathbb{L} \stackrel{\text{def}}{=} \mathbb{R} \cup \overline{\mathbb{R}} \cup \{\Box\}$ where

- *requests* of principals will be built out of $\mathbb{R}$ while their *offers* will be built out of $\overline{\mathbb{R}}$,
- $\mathbb{R} \cap \overline{\mathbb{R}} = \emptyset$, and
- $\Box \notin \mathbb{R} \cup \overline{\mathbb{R}}$ is a distinguished label to represent components that stay idle.

We let $a, b, c, \ldots$ range over $\mathbb{L}$ and fix an involution $\bar{\cdot} : \mathbb{L} \to \mathbb{L}$ such that

$$\overline{\mathbb{R}} \subseteq \overline{\mathbb{R}}, \qquad \overline{\overline{\mathbb{R}}} \subseteq \mathbb{R}, \qquad \forall a \in \mathbb{R} \cup \overline{\mathbb{R}} : \bar{\bar{a}} = a, \qquad \text{and} \qquad \bar{\Box} = \Box$$

Definition 1 *(Actions).* A tuple $\vec{a}$ on $\mathbb{L}$ is

- a *request (action) on b* if and only if $\vec{a}$ is of the form $\Box^* b \Box^*$ with $b \in \mathbb{R}$
- an *offer (action) on b* if and only if $\vec{a}$ is of the form $\Box^* b \Box^*$ with $b \in \overline{\mathbb{R}}$
- a *match (action) on b* if and only if $\vec{a}$ is of the form $\Box^* b \Box^* \bar{b} \Box^*$ with $b \in \mathbb{R} \cup \overline{\mathbb{R}}$.

We define the relation $\bowtie \subseteq \mathbb{L}^* \times \mathbb{L}^*$ as the symmetric closure of $\dot{\bowtie} \subseteq \mathbb{L}^* \times \mathbb{L}^*$ where $\vec{a}_1 \mathrel{\dot{\bowtie}} \vec{a}_2$ if and only if

- $\vec{a}_1$ and $\vec{a}_2$ are actions of the same length and
- $\exists b \in \mathbb{R} \cup \overline{\mathbb{R}} : \vec{a}_1$ is an offer on $b \Longrightarrow \vec{a}_2$ is a request on b and
- $\exists b \in \mathbb{R} \cup \overline{\mathbb{R}} : \vec{a}_1$ is a request on $b \Longrightarrow \vec{a}_2$ is a offer on b.

We write $\vec{a}_1 \bowtie_b \vec{a}_2$ when there is $b \in \mathbb{R} \cup \overline{\mathbb{R}}$ such that $\vec{a}_1$ and $\vec{a}_2$ are actions on b and $\vec{a}_1 \bowtie \vec{a}_2$.

We are ready to introduce the notion of a contract automaton made of n principal, each of which represents an entity involved in the service.

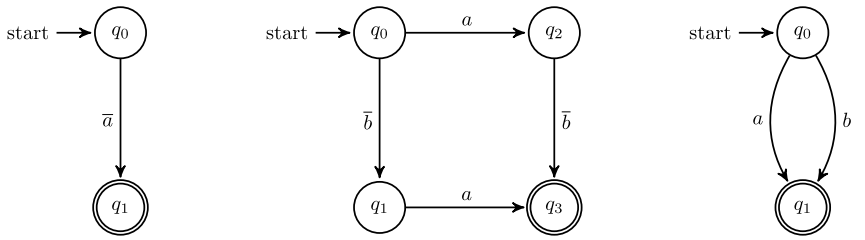

Fig. 1. The contract automata of Alice, Bob and Carol.

Definition 2 *(Contract Automata).* Let $\mathcal{Q}$ (ranged over by $q_1, q_2, \ldots$) be a finite set of states. A *contract automaton of rank n* is a (finite-state) automaton $\mathcal{A} = \langle \mathcal{Q}^n, \vec{q_0}, \mathbb{L}^n, T, F \rangle$, where

- $\vec{q_0} \in \mathcal{Q}^n$ is the *initial* state
- $F \subseteq \mathcal{Q}^n$ is the set of *accepting* states
- $T \subseteq \mathcal{Q}^n \times \mathbb{L}^n \times \mathcal{Q}^n$ is the set of *transitions* such that $(\vec{q}, \vec{a}, \vec{q'}) \in T$ if and only if
  - if $\vec{a}_{(i)} = \Box$ then $\vec{q}_{(i)} = \vec{q'}_{(i)}$ (i.e., the i-th principal stays idle) and
  - $\vec{a}$ is either a request, or an offer, or else a match action

A *principal* is a contract automaton $\mathcal{A}$ of rank 1 such that, for any two transitions (q_1, a_1, q_1') and (q_2, a_2, q_2') in $\mathcal{A}$, it is not the case that $a_1 \bowtie a_2$.

Note that a principal is not allowed to require what it offers.

Example 1. Consider again the simple example of the Introduction. The principal automata of Alice, Bob, and Carol are in Fig. 1, with the usual graphical conventions.

Given a contract automaton of rank n $\mathcal{A} = \langle \mathcal{Q}^n, \vec{q_0}, \mathbb{L}^n, T, F \rangle$, the standard definitions and constructions of finite-state automata apply. In particular,

- the configurations of $\mathcal{A}$ are pairs in $\mathcal{Q}^n \times (\mathbb{L}^n)^*$ of strings of n-tuples of labels and states of $\mathcal{A}$;
- $\mathcal{A}$ *moves from* $(\vec{q}, w)$ *to* $(\vec{q'}, w')$, written $(\vec{q}, w) \xrightarrow{\vec{a}} (\vec{q'}, w')$, if and only if $w = \vec{a}w'$ and $(\vec{q}, \vec{a}, \vec{q'}) \in T$; we write $(\vec{q}, w) \to (\vec{q'}, w')$ when $\vec{a}$ is immaterial and $\vec{q} \xrightarrow{\vec{a}} \vec{q'}$ when w is immaterial (note that transitions are triples, instead);
- the *language of* $\mathcal{A}$ is $\mathscr{L}(\mathcal{A}) = \{w \mid (\vec{q_0}, w) \to^* (\vec{q}, \varepsilon), \vec{q} \in F\}$ where $\to^*$ is the reflexive and transitive closure of $\to$. As usual, $\vec{q}_1 \xrightarrow{\vec{a}_1 \cdots \vec{a}_m} \vec{q}_{m+1}$ shortens $\vec{q}_1 \xrightarrow{\vec{a}_1} \vec{q}_2 \cdots \vec{q}_m \xrightarrow{\vec{a}_m} \vec{q}_{m+1}$ (for some $\vec{q}_2, \ldots, \vec{q}_m$) and we say that $\vec{q}_1$ is reachable in $\mathcal{A}$ if $\vec{q_0} \xrightarrow{w} \vec{q}_1$; finally $\vec{q} \not\to$ if and only if for no q' it is the case that $\vec{q} \to q'$.

We now borrow from [3] the product operation of contract automata, that is similar to the one introduced in [29]. Given a finite set of contract automata, this operation basically yields the contract automaton that interleaves all their transitions while forcing synchronisations when two contract automata are in states ready to "handshake" (i.e., they can fire complementary request/offer actions).

Definition 3 *(Product).* For $i \in \{1, \ldots, h\}$, let $\mathcal{A}_i = \langle \mathcal{Q}^{n_i}, \vec{q}_{0i}, \mathbb{L}^{n_i}, T_i, F_i \rangle$ be a contract automaton of rank n_i. The *product of* $\mathcal{A}_1, \ldots, \mathcal{A}_h$, denoted as $\bigotimes_{i \in \{1,\ldots,h\}} \mathcal{A}_i$, is the contract automaton $\langle \mathcal{Q}^n, \vec{q_0}, \mathbb{L}^n, T, F \rangle$ of rank $n = n_1 + \ldots + n_h$ where:

- $\vec{q_0} = \vec{q}_{01} \ldots \vec{q}_{0h}$
- $F = F_1 \times \ldots \times F_h$
- T is the smallest subset of $\mathcal{Q}^n \times \mathbb{L}^n \times \mathcal{Q}^n$ such that $(\vec{q}, \vec{c}, \vec{q'}) \in T$ if and only if, letting $\vec{q} = \vec{q}_1 \ldots \vec{q}_h \in \mathcal{Q}^n$, $\vec{c} = \vec{c}_1, \ldots, \vec{c}_h \in \mathbb{L}^n$,

  either there are $1 \le i < j \le h$ such that $(\vec{q}_i, \vec{a}_i, \vec{q'}_i) \in T_i$, $(\vec{q}_j, \vec{a}_j, \vec{q'}_j) \in T_j$, $\vec{a}_i \bowtie \vec{a}_j$, and

  $$\begin{cases} \vec{c}_i = \vec{a}_i,\ \vec{c}_j = \vec{a}_j, \text{ and } \vec{c}_l = \Box^{n_l} \text{ for } l \in \{1, \ldots, h\} \setminus \{i, j\} \\ \text{and} \\ \vec{q'} = \vec{q}_1 \ldots \vec{q}_{i-1}\, \vec{q'}_i\, \vec{q}_{i+1} \ldots\, \vec{q}_{j-1}\, \vec{q'}_j\, \vec{q}_{j+1} \ldots \vec{q}_h \end{cases}$$

  or for each $l \neq i \in \{1, \ldots, h\}$, if $(\vec{q}_i, \vec{a}_i, \vec{q'}_i) \in T_i$ then

  $$\vec{c}_i = \vec{a}_i, \qquad \vec{c}_l = \Box^{n_l}, \text{ and } \vec{q'} = \vec{q}_1 \ldots \vec{q}_{i-1} \vec{q'}_i \vec{q}_{i+1} \ldots \vec{q}_h$$

  also, for all $j \neq i$ and $(\vec{q}_j, \vec{a}_j, \vec{q'}_j) \in T_j$ it does not hold that $\vec{a}_i \bowtie \vec{a}_j$.

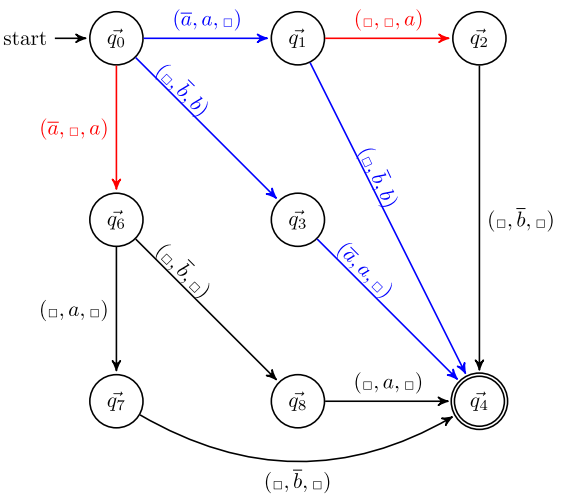

Fig. 2. Alice ⊗ Bob ⊗ Carol.

Example 2. The contract automaton in Fig. 2 is the product of the contract automata in Fig. 1. Notice that only match actions leave the states $\vec{q_0}$ and $\vec{q_3}$, indeed principals can handshake and so there are no outgoing offer and request transitions.

Remark 1. The product in Definition 3 is not associative: consider again Example 1. Then the composition the automaton for Carol with the product of those for Alice and Bob is different from the composition of the automaton of Alice with the product of the other two. An alternative definition of associative product is given in [3]. Briefly, before computing the product, all its arguments are projected to their principals. Aiming at a simpler presentation, we opted here for the non-associative product, but we stress that our results hold also with the associative one, because we are composing principals only.

Hereafter, we assume that all contract automata of rank $n > 1$ are the product of n principals. Also, we consider deterministic contract automata only. These assumptions can be relaxed at the cost of adding some technical intricacies.

3.2. Communicating machines

Communicating machines [4] are a simple automata-based model introduced to specify and analyse systems made of agents interacting via asynchronous message passing. We adapt the original definitions and notation from [4] and [30] to our needs; in particular, the only relevant difference with the original model is that we add the set of final states. Let $\mathcal{P}$ be a finite set of *participants* (ranged over by $\mathtt{p}$, $\mathtt{q}$, $\mathtt{r}$, $\mathtt{s}$, etc.) and $C \stackrel{\text{def}}{=} \{\mathtt{pq} \mid \mathtt{p}, \mathtt{q} \in \mathcal{P} \text{ and } \mathtt{p} \neq \mathtt{q}\}$ be the set of *channels*.

The set of *actions* is $\mathsf{Act} \stackrel{\text{def}}{=} C \times (\mathbb{R} \cup \mathbb{O})$ and it is ranged over by ℓ; we abbreviate $(\mathtt{sr}, \overline{a})$ with $\overline{a}@\mathtt{sr}$ when $\overline{a} \in \mathbb{O}$, representing the action of *sending* a from machine $\mathtt{s}$ to $\mathtt{r}$. Similarly, we shorten $(\mathtt{sr}, a)$ with $a@\mathtt{sr}$ when $a \in \mathbb{R}$, representing the *reception* of a by $\mathtt{r}$.

Definition 4 *(CFSM).* Given a finite set of *states* Q, a *communicating finite state machine* is an automaton $M = (Q, q_0, \mathsf{Act}, \delta, F)$ where

- $q_0 \in Q$ is the *initial* state,
- $\delta \subseteq Q \times \mathsf{Act} \times Q$ is the set of *transitions*,
- $F \subseteq Q$ is the set of final, accepting states.

We say that M is *deterministic* if and only if for all states $q \in Q$ and all actions $\ell \in \mathsf{Act}$, if $(q, \ell, q'), (q, \ell, q'') \in \delta$ then $q' = q''$.

Finally, we write $\mathcal{L}(M) \subseteq \mathsf{Act}^*$ for the language on Act accepted by the automaton, i.e. the machine M.

The notion of deterministic communicating finite state machines adopted here differs from the standard one (e.g., the one in [30]) which requires that, for any state q, if $(q, \overline{a}@\mathtt{sr}, q') \in \delta$ and $(q, \overline{b}@\mathtt{sr}, q'') \in \delta$ then $a = b$ and $q' = q''$. We use our variant because it reflects the semantics of contract automata better. Indeed, hereafter, we will only consider deterministic communicating finite state machines.

The communication model of communicating finite state machines (cf. Definitions 5 and 6) is based on (unbounded) FIFO buffers, that actually are the elements in C. They are to be intended as the channels that the participants use to exchange

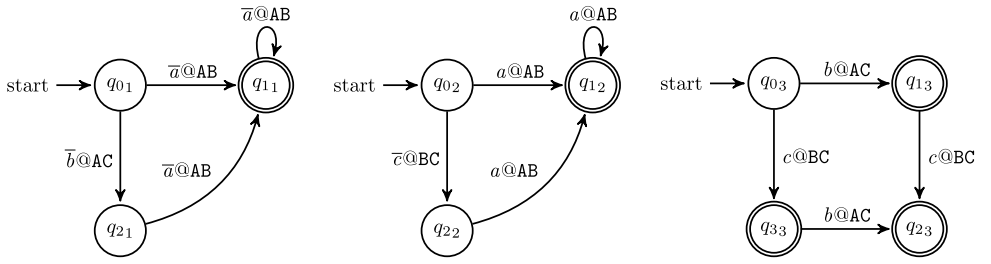

Fig. 3. Three communicating machines.

messages. To spare another syntactic category and cumbersome definitions, we draw the messages appearing in the buffers of communicating finite state machines from the set of requests $\mathbb{R}$. Recall that the set of participants $\mathcal{P}$ is finite.

Definition 5 *(Communicating systems).* Given a communicating finite state machine $M_{\mathtt{p}} = (Q_{\mathtt{p}}, q_{0\mathtt{p}}, \mathsf{Act}, \delta_{\mathtt{p}}, F_{\mathtt{p}})$ for each $\mathtt{p} \in \mathcal{P}$, the tuple $S = (M_{\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$ is a *communicating system*, belonging to the set *CS*.

A *configuration* of S is a pair $s = (\vec{q}; \vec{u})$ where $\vec{q} = (q_{\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$ with $q_{\mathtt{p}} \in Q_{\mathtt{p}}$ and where $\vec{u} = (u_{\mathtt{pq}})_{\mathtt{pq}\in C}$ with $u_{\mathtt{pq}} \in \mathbb{R}^*$; the component $\vec{q}$ is the *control state* and $q_{\mathtt{p}} \in Q_{\mathtt{p}}$ is the *local state* of machine $M_{\mathtt{p}}$, while $\vec{u}$ represents the contents of the (FIFO buffers of the) channels.

The *initial configuration* of S is $s_0 = (\vec{q_0}; \vec{\varepsilon})$ with $\vec{q_0} = (q_{0\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$ and $\vec{\varepsilon}$ is the vector of empty channels.

Hereafter, we fix a machine $M_{\mathtt{p}} = (Q_{\mathtt{p}}, q_{0\mathtt{p}}, \mathsf{Act}, \delta_{\mathtt{p}}, F_{\mathtt{p}})$ for each participant $\mathtt{p} \in \mathcal{P}$ and we let $S = (M_{\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$ be the corresponding system. The definition below formalises a computation step of a communicating system: if a machine $M_{\mathtt{s}}$ sends a message a to a machine $M_{\mathtt{r}}$ then a is inserted on (the FIFO buffer of) the channel $\mathtt{sr}$ connecting the two, rendered by $\mathtt{sr} \cdot a$ (condition 1 below). The element is then read from the channel, if this is in the form $a \cdot \mathtt{sr}$ being a the top of the channel (condition 2 below). In both cases, no other machine is affected in the step.

Definition 6 *(Reachable state).* A configuration $s' = (\vec{q'}; \vec{u'})$ is *reachable* from another configuration $s = (\vec{q}; \vec{u})$ by *firing* $\ell \in \mathsf{Act}$, written $s \xrightarrow{\ell} s'$, if there exists $a \in \mathbb{R}$ such that:

1. if $\ell = \bar{a}@\mathtt{sr}$ then $(\vec{q}_{(\mathtt{s})}, \ell, \vec{q'}_{(\mathtt{s})}) \in \delta_{\mathtt{s}}$ and for all $\mathtt{p} \neq \mathtt{s}$, $\vec{q'}_{(\mathtt{p})} = \vec{q}_{(\mathtt{p})} \wedge \vec{u'}_{(\mathtt{sr})} = \vec{u}_{(\mathtt{sr})} \cdot a$;
2. if $\ell = a@\mathtt{sr}$ then $(\vec{q}_{(\mathtt{r})}, \ell, \vec{q'}_{(\mathtt{r})}) \in \delta_{\mathtt{r}}$ and for all $\mathtt{p} \neq \mathtt{r}$, $\vec{q'}_{(\mathtt{p})} = \vec{q}_{(\mathtt{p})} \wedge \vec{u}_{(\mathtt{sr})} = a \cdot \vec{u'}_{(\mathtt{sr})}$;
3. and, in both (1) and (2) above, for all $\mathtt{pq} \neq \mathtt{sr}$, $\vec{u'}_{(\mathtt{pq})} = \vec{u}_{(\mathtt{pq})}$.

As usual, the *computation* $s_1 \xrightarrow{\ell_1 \cdots \ell_m} s_{m+1}$ shortens $s_1 \xrightarrow{\ell_1} s_2 \cdots s_m \xrightarrow{\ell_m} s_{m+1}$ (for some $s_2, \ldots, s_m$).
The set of *reachable configurations of* S is $RS(S) = \{s \mid s_0 \to^* s\}$.

Example 3. Fig. 3 shows a graphical representation of a communicating system made of three communicating machines, used later on to illustrate the mapping from contract automata to communicating machines.

Similarly to finite state automata, communicating machines have one initial state and a set of final states. A transition has a channel as label, e.g. $\bar{c}@\mathtt{BC}$, indicating that the action c is sent by participant $\mathtt{B}$ to the buffer of the participant $\mathtt{C}$; similarly for a receive action. Note that channels of communicating machines specify for each action the sender and the receiver, which is not the case for contract automata.

3.3. Notational synopsis

To avoid cumbersome repetitions, through the paper we assume fixed a contract automaton $\mathcal{A} = \langle \mathcal{Q}^n, \vec{q_0}, \mathbb{L}^n, T, F \rangle$ of rank n. For readability, we summarise the notations introduced so far in Table 1.

Finally, we assume that the states of any automaton/machine are built out of a fixed universe $\mathcal{Q}$ (of states).

4. Enforcing agreement

This section introduces the original notion of *strong agreement* on contract automata, that elaborates the notions of *agreement* and *weak agreement* introduced in [3]. The three notions differ on the conditions for the fulfilment of an interaction between different principals, and share the basic requirement that all the requests of principals are satisfied. Briefly, an *agreement* exists if all the requests, but not necessarily all the offers, are satisfied synchronously. Intuitively, this means that the orchestrator guarantees two principals that their requests are synchronously matched by complementary actions.

Table 1
Notational synopsis.

X^*	set of finite words w on an alphabet X;
	ε is the empty word;
	juxtaposition of words is denoted by $\_\cdot\_$
$\lvert w\rvert$, $w_{(i)}$	the length of w and its i-th symbol, respectively
w^n (w^*, resp.)	w concatenated n-times
	(arbitrarily many, resp.) with itself
$\vec{z}$ or $(z_i)_{1\leq i\leq n}$	indexed tuples of states, labels, channel contents, ...
$\mathbb{L}$	labels
$\mathbb{R}$	request labels, ranged over by $\bar{a}, \bar{b}, \ldots$
$\mathbb{O}$	offer labels, ranged over by $a, b, \ldots$
$\square \notin \mathbb{R} \cup \mathbb{O}$	idle label
$\mathcal{A}$	a contract automaton of rank n the language of which is $\mathscr{L}(\mathcal{A})$
$\mathcal{P}$	set of participants
	(ranged over by $\mathtt{p}$, $\mathtt{q}$, $\mathtt{i}$, $\mathtt{j}$, $\mathtt{A}$, $\mathtt{B}$, $\mathtt{C}$, etc.)
C	set of channels, i.e. pairs of participants;
	(ranged over by $\mathtt{pq}$)
$u \in \mathbb{R}^*$	the FIFO buffer of a channel
	(top on left, bottom on right)
$M_{\mathtt{p}}$	communicating machine of participant $\mathtt{p}$
S	a system of communicating machines
$\mathsf{Act} = C \times (\mathbb{R} \cup \mathbb{O})$	actions ℓ, either $\bar{a}@\mathtt{pq}$ (sending offer $\bar{a}$)
	or $a@\mathtt{pq}$ (receiving request a)
$s = (\vec{q}; \vec{u})$	a configuration
$RS(S)$	the set of the reachable configurations of S

Instead, a *weak agreement* exists when request actions can be performed "on credit". In other words, a computation yields weak agreement when the fulfilment of a request action can happen after the action has been taken. Intuitively, this corresponds to an asynchronous communication admitting inputs taken on credit, provided that obligations will be honoured later on.

Here, we focus on *strongly safe* contract automata, i.e. those enjoying the property of *strong agreement*. This notion is a tighter version of the ones sketched above: it requires *synchronous* fulfilment of all offers *and* requests. In Section 5 we will show how this condition corresponds to interactions between communicating machines that always succeed.

Definition 7 *(Strong Agreement and Strong Safety).* A *strong agreement on* $\mathbb{L}$ is a finite sequence of match actions. We let $\mathfrak{Z}$ denote the set of all strong agreements on $\mathbb{L}$.

A contract automaton $\mathcal{A}$ is *strongly safe* if $\mathscr{L}(\mathcal{A}) \subseteq \mathfrak{Z}$. We say that $\mathcal{A}$ *admits strong agreement* when $\mathscr{L}(\mathcal{A}) \cap \mathfrak{Z} \neq \emptyset$.

We can compose a set of principals so to obtain a strongly safe contract automaton adopting an approach borrowed from the supervisory control theory for discrete event systems [8]. In this theory, discrete event systems are basically automata where accepting states represent successful termination. Instead, *forbidden states* are those that may lead to failures, and thus should not be traversed in "good" computations. The purpose is then to synthesise a controller that enforces a given system to never pass through a forbidden state. The supervisory control theory distinguishes between *controllable* events (those events that the controller can disable) and *uncontrollable* events (those that are always enabled). Moreover, the theory partitions events in *observable* and *unobservable*; the latter being a subset of uncontrollable events. It is known that if all events are observable then a maximally permissive controller exists that never blocks a good computation [8].

We assume that the orchestrator can prescribe any kind of action to be forbidden, so all the events will be controllable. Since the behaviours that we want to enforce in $\mathcal{A}$ are exactly those traces labelled by words in $\mathfrak{Z} \cap \mathscr{L}(\mathcal{A})$, we then specialise the notions of supervisory control theory by defining

- observable events to be all offer, request, and match actions;
- forbidden events to be non-match actions.

Definition 8 *(Controller).* A *(strong) controller* of $\mathcal{A}$ is a contract automaton $KS_{\mathcal{A}}$ such that $\mathscr{L}(KS_{\mathcal{A}}) \subseteq \mathfrak{Z} \cap \mathscr{L}(\mathcal{A})$.

The *most permissive (strong) controller of* $\mathcal{A}$ is the controller $KS_{\mathcal{A}}$ such that $\mathscr{L}(KS'_{\mathcal{A}}) \subseteq \mathscr{L}(KS_{\mathcal{A}})$ for all $KS'_{\mathcal{A}}$ controllers of $\mathcal{A}$.

It is immediate that the most permissive controller is unique up-to language equivalence.

Example 4. The most permissive controller of the contract automaton in Fig. 2 has of the states $\vec{q}_0$, $\vec{q}_1$, $\vec{q}_3$, and $\vec{q}_4$ with transitions $(\vec{q}_0, (\bar{a}, a, \square), \vec{q}_1)$, $(\vec{q}_3, (\bar{a}, a, \square), \vec{q}_4)$, $(\vec{q}_0, (\square, \bar{b}, b), \vec{q}_3)$, and $(\vec{q}_1, (\square, \bar{b}, b), \vec{q}_4)$. Such controller is easily obtained by applying the construction of Lemma 1 below.

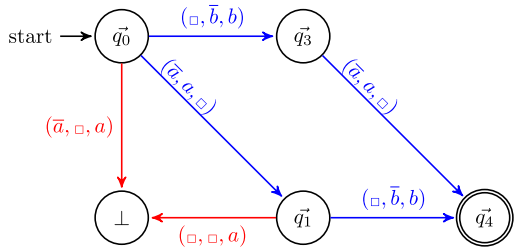

Fig. 4. The controlled system of the contract automaton in Fig. 2.

Proposition 1. *If $KS_{\mathcal{A}}$ is the most permissive controller of $\mathcal{A}$ then $\mathscr{L}(KS_{\mathcal{A}}) = \mathfrak{Z} \cap \mathscr{L}(\mathcal{A})$.*

Proof. By contradiction, assume $\mathscr{L}(KS_{\mathcal{A}}) \subset \mathfrak{Z} \cap \mathscr{L}(\mathcal{A})$. Since $\mathfrak{Z} \cap \mathscr{L}(\mathcal{A})$ is the intersection of two regular languages and all actions are controllable, there exists a contract automaton $KS'_{\mathcal{A}}$ accepting it (cf. [8]). By definition, $KS'_{\mathcal{A}}$ is a controller of $\mathcal{A}$ strictly containing $\mathscr{L}(KS_{\mathcal{A}})$, contradicting the hypothesis that $\mathscr{L}(KS_{\mathcal{A}})$ is the most permissive controller. □

It is convenient to introduce the following definition.

Definition 9. A state $\vec{q}$ is *redundant in* the contract automaton $\mathcal{A}$ if, and only if, no accepting state of $\mathcal{A}$ can be reached from $\vec{q}$.

Lemma 1 *(MPC). A contract automaton is the most permissive controller of $\mathcal{A}$ if and only if it is language-equivalent to*

$$KS_{\mathcal{A}} \stackrel{\text{def}}{=} \langle \mathcal{Q}^n, \vec{q_0}, \mathbb{L}^n, T_K \setminus \{(\vec{q}, a, \vec{q'}) \mid \vec{q} \text{ or } \vec{q'} \text{ is redundant in } K\}, F\rangle$$

where $K = \langle \mathcal{Q}^n, \vec{q_0}, \mathbb{L}^n, T_K = \{t \in T \mid t \text{ is a match transition}\}, F\rangle$ is the sub-automaton of $\mathcal{A}$ the transition relation of which consists of the match transitions of $\mathcal{A}$ only.

Proof. By definition K is a sub-automaton of $\mathcal{A}$ and, by construction, the transitions of $KS_{\mathcal{A}}$ are a subset of the transitions of $\mathcal{A}$ and K contains only match transition, hence $\mathscr{L}(KS_{\mathcal{A}}) \subseteq \mathfrak{Z} \cap \mathscr{L}(\mathcal{A})$. Therefore $KS_{\mathcal{A}}$ is a controller of $\mathcal{A}$ and we have just to prove that $\mathscr{L}(KS_{\mathcal{A}}) = \mathfrak{Z} \cap \mathscr{L}(\mathcal{A})$. We proceed by contradiction.

Let $w \in (\mathfrak{Z} \cap \mathscr{L}(\mathcal{A})) \setminus \mathscr{L}(KS_{\mathcal{A}})$. Since if $\varepsilon \in \mathscr{L}(\mathcal{A})$ then $\varepsilon \in \mathscr{L}(KS_{\mathcal{A}})$, we have $w \neq \varepsilon$ and there must be a transition $t = (\vec{q}, \vec{a}, \vec{q'})$ of $\mathcal{A}$ not in $KS_{\mathcal{A}}$ in the accepting path of w (which is unique since we consider deterministic contract automata only), otherwise $w \in \mathscr{L}(KS_{\mathcal{A}})$. We know that $\vec{a}$ is a match action because $w \in \mathfrak{Z}$, and $\vec{q}$,$\vec{q'}$ are not redundant states of $\mathcal{A}$ because the transition belongs to an accepting path. Hence there must be $t \in KS_{\mathcal{A}}$, since by construction match transitions between non-redundant states are in $KS_{\mathcal{A}}$. □

From now onwards, we consider as most permissive controller the one obtained by the construction of Lemma 1. We then define the *controlled system* of a contract automaton $\mathcal{A}$, using such controller. The purpose of this new automaton is to identify the match transitions of $\mathcal{A}$ together with those that lead "outside" of the controller; in its definition we use the distinguished state $\perp \notin \mathcal{Q}^n$ (for any n).

Definition 10 *(Controlled system).* Let $KS_{\mathcal{A}} = \langle \mathcal{Q}^n, \vec{q_0}, \mathbb{L}^n, T' \subseteq T, F\rangle$ be the most permissive controller of $\mathcal{A}$. The *controlled system of $\mathcal{A}$ under $KS_{\mathcal{A}}$* is the automaton

$$KS_{\mathcal{A}}/\mathcal{A} = \langle \mathcal{Q}^n \cup \{\perp\}, \vec{q_0}, \mathbb{L}^n, T'', F\rangle$$

such that

$$T'' = T' \cup \{(\vec{q}, \vec{a}, \perp) \mid \vec{q} \text{ reachable in } KS_{\mathcal{A}} \text{ and } \exists \vec{q'} \in \mathcal{Q}^n : (\vec{q}, \vec{a}, \vec{q'}) \in T \setminus T'\}$$

Example 5. The controlled system of the contract automaton in Fig. 2 is obtained by adding the transitions $(\vec{q}_1, (\Box, \Box, a), \perp)$, $(\vec{q}_0, (\overline{a}, \Box, a), \perp)$ to the most permissive controller of Example 4. It is displayed in Fig. 4.

It is worth remarking that the transitions reaching $\perp$ in the controlled system of $\mathcal{A}$ identify the start of the computations in $\mathcal{A}$ which lead to violations of strong agreement.

In the next definition, we introduce a notion of strong liability, to single out the principals that are potentially responsible of the divergence from the expected behaviour.

Definition 11 *(Strong Liability).* Given a controlled system $KS_{\mathcal{A}}/\mathcal{A}$ of rank n, the set of *liable* principals on a trace $w \in \mathscr{L}(\mathcal{A})$ is given by:

$$Liable(KS_{\mathcal{A}}/\mathcal{A}, w) = \{\mathtt{i} \leq n \mid (\vec{q_0}, w) \to^* (\vec{q}, \vec{a}\,w') \to (\bot, w'') \text{ in } KS_{\mathcal{A}}/\mathcal{A}, \vec{a}_{(\mathtt{i})} \neq \Box\}$$

The *potentially liable principals* in $KS_{\mathcal{A}}/\mathcal{A}$ are:

$$Liable(KS_{\mathcal{A}}/\mathcal{A}) \stackrel{\text{def}}{=} \bigcup_{w \in \mathscr{L}(\mathcal{A})} Liable(KS_{\mathcal{A}}/\mathcal{A}, w)$$

Finally, let $TLiable(KS_{\mathcal{A}}/\mathcal{A})$ denote the set of transitions of $\mathcal{A}$ that make principals liable:

$$TLiable(KS_{\mathcal{A}}/\mathcal{A}) \stackrel{\text{def}}{=} \{(\vec{q}, \vec{a}, \bot) \in T_{KS_{\mathcal{A}}/\mathcal{A}} \mid \vec{q} \in \mathcal{Q}^n, \vec{a} \in (\mathbb{L}^*)^n\}$$

Note that the transition labelled by $\vec{a}$ in Definition 11 is the first which diverges from the expected path (since, by Definition 10, state $\bot$ has no outgoing transitions). Indeed a liable principal may fire an action taking the computation away from agreement.

Example 6. The liable indexes of the contract automaton in Fig. 4 are 1 and 3, corresponding to Alice and Carol respectively; the transitions that make them liable are respectively $(\vec{q}_0,\ (\bar{a}, \Box, a),\ \bot)$ and $(\vec{q}_1,\ (\Box, \Box, a),\ \bot)$. The former liable transition is a match that leads to a non-match transition. So, by inspecting the labels, we easily identify (the indexes of) the liable principals.

5. From contract automata to communicating machines

The translation of a principal into a communicating machine is conceptually straightforward as the two automata are almost isomorphic, apart from their labels. Recall that the principals in a contract automaton can fire transitions not matched by other principals. To account for this kind of "openness," the Definition 12 below uses the new "$-$" symbol to represent a special, "anonymous" participant, distinguished from those composing the contract automaton in hand, and playing the role of the environment. For this reason, we will assume from now onwards that actions in Act are built on $C \stackrel{\text{def}}{=} \{\mathtt{pq} \mid \mathtt{p}, \mathtt{q} \in \mathcal{P} \cup \{-\} \text{ and } \mathtt{p} \neq \mathtt{q}\}$.

Definition 12 *(Translation).* For a participant $\mathtt{p} \in \mathcal{P}$, let $[\![\_]\!]_{\mathtt{p}} : \mathbb{L}^n \to \mathsf{Act}$ be defined as:

$$[\![\vec{a}]\!]_{\mathtt{p}} = \begin{cases} \bar{a}@\mathtt{ij} & \text{if } \vec{a} \text{ is a match action and } \mathtt{i} \text{ and } \mathtt{j} \text{ are such that} \\ & \vec{a}_{(\mathtt{i})} \in \mathbb{O} \text{ and } \vec{a}_{(\mathtt{j})} \in \mathbb{R} \text{ and } \mathtt{p} = \mathtt{i} \\ a@\mathtt{ij} & \text{if } \vec{a} \text{ is a match action and } \mathtt{i} \text{ and } \mathtt{j} \text{ are such that} \\ & \vec{a}_{(\mathtt{i})} \in \mathbb{O} \text{ and } \vec{a}_{(j)} \in \mathbb{R} \text{ and } \mathtt{p} = \mathtt{j} \\ \bar{a}@\mathtt{i}- & \text{if } \vec{a} \text{ is an offer action and } \mathtt{i} \text{ is such that } \vec{a}_{(\mathtt{i})} \in \mathbb{O} \text{ and } \mathtt{p} = \mathtt{i} \\ a@-\mathtt{j} & \text{if } \vec{a} \text{ is a request action and } \mathtt{j} \text{ is such that } \vec{a}_{(\mathtt{j})} \in \mathbb{R} \text{ and } \mathtt{p} = \mathtt{j} \\ \varepsilon & \text{otherwise} \end{cases}$$

The *translation of $\mathcal{A}$ to a communicating finite state machine* is given by the map

$$[\![\mathcal{A}]\!]_{\mathtt{p}} \stackrel{\text{def}}{=} \langle \mathcal{Q}, \vec{q_0}_{(p)}, \mathsf{Act}, \{(\vec{q}_{(p)}, [\![\vec{a}]\!]_{\mathtt{p}}, \vec{q'}_{(p)}) \mid (\vec{q}, \vec{a}, \vec{q'}) \in T \text{ and } [\![\vec{a}]\!]_{\mathtt{p}} \neq \varepsilon\}, F\rangle$$

The communicating system *corresponding* to the contract automaton $\mathcal{A}$ is $S(\mathcal{A}) = ([\![\mathcal{A}]\!]_{\mathtt{p}})_{\mathtt{p} \in \{1,\ldots,n\}}$.

The following example illustrates the composition of three contract automata, its most permissive controller, and its translation into a system of three communicating machines.

Example 7. Fig. 5 shows three contract automata A, B, and C (top), their product $\mathtt{A} \otimes \mathtt{B} \otimes \mathtt{C}$ (middle), with initial state $\vec{q}_0 = \langle q_{01}, q_{02}, q_{03}\rangle$, and the corresponding most permissive controller $KS_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}}$ (bottom). The translations $[\![KS_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}}]\!]_{\mathtt{A}}$, $[\![KS_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}}]\!]_{\mathtt{B}}$, and $[\![KS_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}}]\!]_{\mathtt{C}}$ as per Definition 12 yields the three communicating machines in Fig. 3.

We constrain the behaviour of communicating machines, so to make it easier reflecting in this model the notion of strong agreement defined on contract automata. Intuitively, the new semantics, called *1-buffer* semantics, only allows a machine $M_{\mathtt{p}}$ to send a message $\bar{a}$ to a partner $M_{\mathtt{q}}$ if in the communicating system S all the channels are empty. To specify the *1-buffer* semantics, it suffices to constrain the component $\vec{u}_{(\mathtt{pq})}$ of the reachable configurations $RS(S)$ of S, and only keeping the transitions between the selected configurations. We also define *convergent* communicating systems that always reach a final configuration and so they are *deadlock-free*.

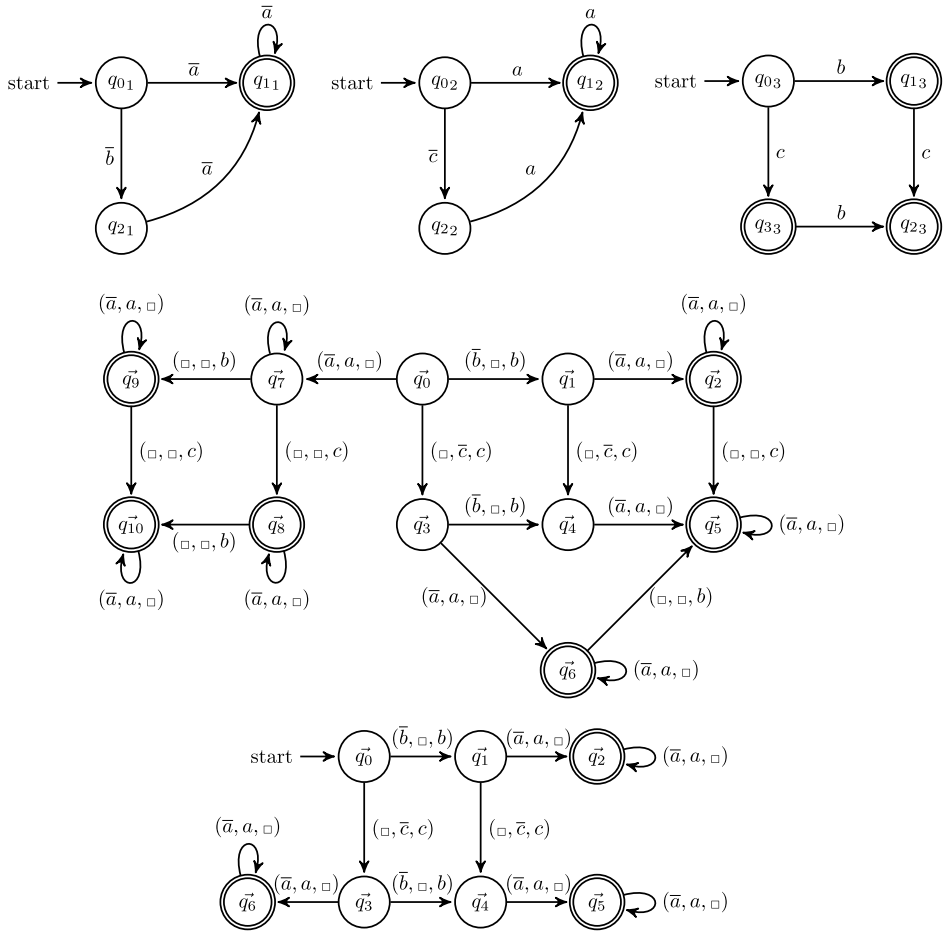

Fig. 5. From left to right, top line: the contract automata $\mathtt{A}, \mathtt{B}, \mathtt{C}$; second line: the contract automaton $\mathtt{A} \otimes \mathtt{B} \otimes \mathtt{C}$, the controlled automaton of which $KS_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}}$ is in the third line; the communicating machines $[\![KS_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}}]\!]_{\mathtt{A}}$, $[\![KS_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}}]\!]_{\mathtt{B}}$, $[\![KS_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}}]\!]_{\mathtt{C}}$ corresponding to the contract automata in the first line are shown in Fig. 3.

Definition 13 *(1-buffer, convergence, deadlock).* Let $S = (M_{\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$ be a communicating system. A configuration $s = (\vec{q}; \vec{u})$ of S is *stable* if and only if $\vec{u} = \vec{\varepsilon}$; additionally, s is *final* if it is stable and $\vec{q} \in (F_{\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$.

The transition relation of the *1-buffer semantics of* S is the following

$$\twoheadrightarrow \stackrel{\text{def}}{=} \rightarrow \cap (RS_{\leq 1}(S) \times \mathsf{Act} \times RS_{\leq 1}(S))$$

where $\rightarrow$ is the relation introduced in Definition 6 and

$$RS_{\leq 1}(S) \stackrel{\text{def}}{=} \{(\vec{q}; \vec{u}) \in RS(S) \mid (\vec{q}; \vec{u}) \text{ is stable or}$$

$$\exists \mathtt{pq} \in C : \exists a \in \mathbb{R} : \vec{u}_{(\mathtt{pq})} = a \wedge \forall \mathtt{sr} \neq \mathtt{pq}. \vec{u}_{(\mathtt{sr})} = \varepsilon\}$$

We say that the system S is *convergent* (with the 1-buffer semantics) if and only if for every reachable configuration $(\vec{q}; \vec{u}) \in RS_{\leq 1}(S)$, there exists a final configuration s such that

$$(\vec{q}; \vec{u}) \twoheadrightarrow^* s.$$

Moreover a configuration $(\vec{q}; \vec{u})$ is a *deadlock* if and only if it is not final and $(\vec{q}; \vec{u}) \not\twoheadrightarrow$.

In order to relate the computations of contract automata and those of communicating systems, it is convenient to define a translation from the first to the second ones, as follows.

Definition 14. Given a sequence of n-tuples of actions $\varphi \in (\mathbb{L}^n)^*$, we define

$$[\![\varphi]\!] \stackrel{\text{def}}{=} \begin{cases} \bar{a}@\mathtt{i}\,a@\mathtt{i}[\![\varphi']\!] & \text{if } \varphi = \vec{a}\varphi' \text{ and } \vec{a} \text{ is a match action on } a \\ & \text{with } \vec{a}_{(\mathtt{i})} \in \mathbb{O} \text{ and } \vec{a}_{(\mathtt{j})} \in \mathbb{R} \\ \bar{a}@\mathtt{i}-[\![\varphi']\!] & \text{if } \varphi = \vec{a}\varphi' \text{ and } \vec{a} \text{ is an offer action on } a \\ & \text{with } \vec{a}_{(\mathtt{i})} \in \mathbb{O} \\ a@-\mathtt{j}[\![\varphi']\!] & \text{if } \varphi = \vec{a}\varphi' \text{ and } \vec{a} \text{ is a request action on } a \\ & \text{with } \vec{a}_{(\mathtt{j})} \in \mathbb{R} \\ \varepsilon & \text{if } \varphi = \varepsilon \\ \text{undefined} & \text{otherwise} \end{cases}$$

Now we are ready to establish a first property showing how a sequence of actions labelling a computation of the corresponding communicating system, is mapped into a trace in strong agreement, i.e. a string of matches.

Property 2. *Let $S(KS_{\mathcal{A}})$ be the communicating system corresponding to $KS_{\mathcal{A}}$, and let s_0 be its initial configuration. If $s_0 \stackrel{f}{\twoheadrightarrow} s$, then*

- *$f \in \mathsf{Act}^*$ is a sequence of pairs $(\bar{a}@\mathtt{i}\,a@\mathtt{i}\mathtt{j})$ possibly followed by $\overline{a'}@\mathtt{i}'\mathtt{j}'$, for some $a, a', \mathtt{i}, \mathtt{j}, \mathtt{i}', \mathtt{j}'$, and*
- *there exists a strong agreement φ such that either $f = [\![\varphi]\!]$ or $f = [\![\varphi]\!]\overline{a'}@\mathtt{i}'\mathtt{j}'$.*

Proof. The first item follows from Definition 13, and the second is then immediate by Definition 14. □

Before establishing our main results, we introduce a notion of well-formedness of contract automata. We require that if an output of a principal $\mathtt{i}$ is enabled in two different states, it can be taken in both, giving raise to the same match, regardless of the (projection of these) states on the other principals $\mathtt{j} \neq \mathtt{i}$ in the product automaton.

In what follows, it is convenient to highlight the principal of a contract automaton that makes an offer or a request. For that, we define $snd(\vec{a}) \stackrel{\text{def}}{=} \mathtt{i}$ when $\vec{a}$ is a match action or an offer action and $\vec{a}_{(\mathtt{i})} \in \mathbb{O}$; similarly, let $rcv(\vec{a}) \stackrel{\text{def}}{=} \mathtt{j}$ when $\vec{a}$ is a match or a request action and $\vec{a}_{(\mathtt{j})} \in \mathbb{R}$. Also, we will omit the target configuration of a transition when immaterial, and simply write, e.g. $\vec{q} \xrightarrow{\vec{a}}$ or $s \xrightarrow{\ell}$.

Definition 15 *(Branching Condition).* A contract automaton $\mathcal{A}$ has the *branching condition* if and only if the following holds for each $\vec{q}_1, \vec{q}_2$ reachable in $\mathcal{A}$

$$\forall \vec{a} \text{ match actions } .(\vec{q}_1 \xrightarrow{\vec{a}} \wedge\, snd(\vec{a}) = \mathtt{i} \wedge \vec{q}_{1\,(i)} = \vec{q}_{2(i)}) \text{ implies } \vec{q}_2 \xrightarrow{\vec{a}}$$

Example 8. The product automaton $\mathtt{A} \otimes \mathtt{B} \otimes \mathtt{C}$ of Example 7 enjoys the branching condition. Indeed, consider the match action $(\bar{a}, a, \Box)$ and the transition $\vec{q}_0 \xrightarrow{(\bar{a},a,\Box)}$. We have that $\vec{q}_{0(1)} = \vec{q}_{3(1)} = q_{01}$ and there also exists the transition $\vec{q}_3 \xrightarrow{(\bar{a},a,\Box)}$. The same happens for:

$$\vec{q}_1 \xrightarrow{(\bar{a},a,\Box)}, \vec{q}_4 \xrightarrow{(\bar{a},a,\Box)} \qquad \vec{q}_2 \xrightarrow{(\bar{a},a,\Box)}, \vec{q}_5 \xrightarrow{(\bar{a},a,\Box)} \qquad \vec{q}_6 \xrightarrow{(\bar{a},a,\Box)}, \vec{q}_5 \xrightarrow{(\bar{a},a,\Box)}$$

$$\vec{q}_7 \xrightarrow{(\bar{a},a,\Box)} \vec{q}_9 \xrightarrow{(\bar{a},a,\Box)} \qquad \vec{q}_7 \xrightarrow{(\bar{a},a,\Box)}, \vec{q}_8 \xrightarrow{(\bar{a},a,\Box)} \qquad \vec{q}_9 \xrightarrow{(\bar{a},a,\Box)}, \vec{q}_{10} \xrightarrow{(\bar{a},a,\Box)}$$

$$\vec{q}_8 \xrightarrow{(\bar{a},a,\Box)}, \vec{q}_{10} \xrightarrow{(\bar{a},a,\Box)} \qquad \vec{q}_0 \xrightarrow{(\bar{b},\Box,b)}, \vec{q}_3 \xrightarrow{(\bar{b},\Box,b)} \qquad \vec{q}_0 \xrightarrow{(\Box,\bar{c},c)}, \vec{q}_1 \xrightarrow{(\Box,\bar{c},c)}$$

Instead, the most permissive controller of $\mathtt{A} \otimes \mathtt{B} \otimes \mathtt{C}$ does not enjoy the branching condition. Consider again the match action $(\bar{a}, a, \Box)$: while $\vec{q}_{0(1)} = \vec{q}_{3(1)} = q_{01}$ and $\vec{q}_3 \xrightarrow{(\bar{a},a,\Box)}$, no transition labelled by $(\bar{a}, a, \Box)$ exits from $\vec{q}_0$.

Theorem 1 characterises the relations between a contract automaton $\mathcal{A}$, its most permissive controller $KS_{\mathcal{A}}$ and the corresponding communicating system $S(KS_{\mathcal{A}})$. It states that $S(KS_{\mathcal{A}})$ is capable of performing all the moves of the controller (item 1), while $\mathcal{A}$, but possibly *not* $KS_{\mathcal{A}}$, can perform all the traces of $S(KS_{\mathcal{A}})$ in strong agreement (item 2a). Moreover the runs of $S(KS_{\mathcal{A}})$ leading to a configuration from which no final configuration is reachable correspond to runs in $\mathcal{A}$ that traverse liable transitions (item 2b). The following example illustrates the relations between convergence and deadlock freedom discussed above on the automata in Fig. 5.

Example 9. Consider the contract automata and communicating machines of Fig. 5. A trace of the system $S(KS_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}})$ is:

$$((q_{01}, q_{02}, q_{03}); (\varepsilon, \varepsilon, \varepsilon)) \xrightarrow{\bar{a}@\mathtt{AB}}$$
$$((q_{11}, q_{02}, q_{03}); (a, \varepsilon, \varepsilon)) \xrightarrow{a@\mathtt{AB}}$$
$$((q_{11}, q_{12}, q_{03}); (\varepsilon, \varepsilon, \varepsilon)) = s$$

We have $[\![(\bar{a}, a, \Box)]\!] = \bar{a}@\mathtt{AB}\ a@\mathtt{AB}$, and the contract automaton $\mathtt{A} \otimes \mathtt{B} \otimes \mathtt{C}$ is capable of performing the transition $(\vec{q_0}, (\bar{a}, a, \Box), \vec{q_7})$, while this is not true for the controller $KS_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}}$.

Note that from the configuration s it is not possible to reach a final configuration, since the participant $\mathtt{C}$ is prevented from reaching a final state, and thus the transition $(\vec{q_0}, (\bar{a}, a, \Box), \vec{q_7})$ of $\mathtt{A} \otimes \mathtt{B} \otimes \mathtt{C}$ is *liable*.

Moreover s is not a deadlock, indeed it is always possible to perform the loop:

$$((q_{11}, q_{12}, q_{03}); (\varepsilon, \varepsilon, \varepsilon)) \xrightarrow{\bar{a}@\mathtt{AB}}$$
$$((q_{11}, q_{12}, q_{03}); (a, \varepsilon, \varepsilon)) \xrightarrow{a@\mathtt{AB}}$$
$$((q_{11}, q_{12}, q_{03}); (\varepsilon, \varepsilon, \varepsilon))$$

As a matter of fact $S(KS_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}})$ is deadlock-free but not convergent.

It is convenient to introduce an *equivalence* between the states of a contract automaton and those of a configuration of a communicating system (assuming them ordered by their indexes).

Definition 16. Let $(\vec{q}; \vec{u})$ be a reachable configuration of a communicating system, and let $\vec{q'}$ be a state of a contract automaton $\mathcal{A}$. Then we let $\vec{q} \sim \vec{q'}$ iff $|\vec{q}| = |\vec{q'}| = n$ and $\forall i \in 1 \ldots n. \vec{q}_{(i)} = \vec{q'}_{(i)}$.

Theorem 1. *Let $\mathcal{A}$ be a contract automaton with initial state $\vec{q}_0$; let $KS_{\mathcal{A}}$ be its most permissive controller with initial state $\vec{q}_{mpc}$; let $S(KS_{\mathcal{A}})$ be the corresponding communicating system with initial configuration s_0. Then, given a strong agreement $\varphi \in \mathfrak{Z}$, the following hold:*

1. *if $\vec{q}_{mpc} \xrightarrow{\varphi} \vec{q'}_{mpc}$, then there exists a configuration s such that $s_0 \overset{[\![\varphi]\!]}{\twoheadrightarrow} s = (\vec{q^s}; \vec{u})$ and $\vec{q'}_{mpc} \sim \vec{q^s}$.*
2. *if $s_0 \overset{f}{\twoheadrightarrow} s = (\vec{q^s}; \vec{u})$, then*
   (a) *if $f = [\![\varphi]\!]$, then there exists $\vec{q'}$ such that $\vec{q}_0 \xrightarrow{\varphi} \vec{q'}$ and $\vec{q^s} \sim \vec{q'}$*
   (b) *if no final configuration is reachable from s, with either $f = [\![\varphi]\!]$ or $f = [\![\varphi]\!]\bar{a}@\mathtt{ij}$, then the run $\vec{q}_0 \xrightarrow{\hat{\varphi}} \vec{q'}$ has traversed a transition in TLiable($KS_{\mathcal{A}}/\mathcal{A}$) with either $\hat{\varphi} = \varphi$ or $\hat{\varphi} = \varphi\vec{a}$, where $\vec{a}$ is a match on a and $snd(\vec{a}) = \mathtt{i}$, $rcv(\vec{a}) = \mathtt{j}$.*

Proof. Through the proof assume that $\vec{q'}$, $\vec{q'}_{mpc}$ and s are such that $\vec{q}_0 \xrightarrow{\varphi} \vec{q'}$, $\vec{q}_{mpc} \xrightarrow{\varphi} \vec{q'}_{mpc}$ and $s_0 \overset{[\![\varphi]\!]}{\twoheadrightarrow} s = (\vec{q}_s; \vec{u})$. Also, let $\vec{a}$ be a match with $snd(\vec{a}) = \mathtt{i}$ and $rcv(\vec{a}) = \mathtt{j}$.

1. By induction on the length of φ.
   The base case is when $\vec{q}_{mpc} \xrightarrow{\vec{a}} \vec{q'}_{mpc}$. Let $\vec{q}_{mpc(\mathtt{i})}$ and $\vec{q}_{mpc(\mathtt{j})}$ be the initial states of participants $\mathtt{i}$ and $\mathtt{j}$ in $S(KS_{\mathcal{A}})$. By Definition 12, we have that
   $$(\vec{q}_{mpc(\mathtt{i})}, \bar{a}@\mathtt{ij}, \vec{q'}_{mpc(\mathtt{i})}) \qquad \text{and} \qquad (\vec{q}_{mpc(\mathtt{j})}, a@\mathtt{ij}, \vec{q'}_{mpc(\mathtt{j})})$$
   are transitions of participants $\mathtt{i}$ and $\mathtt{j}$, respectively. We have $s_0 \overset{\bar{a}@\mathtt{ij}}{\twoheadrightarrow} \overset{a@\mathtt{ij}}{\twoheadrightarrow} s$ since after the first transition participant $\mathtt{j}$ remains in its initial state. Moreover by Definition 12 and Definition 3 we have $\vec{q'}_{mpc(\mathtt{i})} = \vec{q}_{s(\mathtt{i})}$ and $\vec{q'}_{mpc(\mathtt{j})} = \vec{q}_{s(\mathtt{j})}$, all the other components of the states remain in their initial state, and thus $\vec{q'}_{mpc} \sim \vec{q^s}$.
   For the inductive case we have $\varphi = \vec{a}\,\varphi'$, and the run $\vec{q}_{mpc} \xrightarrow{\vec{a}} \vec{q''}_{mpc} \xrightarrow{\varphi} \vec{q'}_{mpc}$ for some $\vec{q''}_{mpc}$. The same argument used above guarantees that there exists $s'' = (\vec{q^{s''}}; \vec{u''})$ and $\vec{q^{s''}} \sim \vec{q''}_{mpc}$, such that $s_0 \overset{\bar{a}@\mathtt{ij}}{\twoheadrightarrow} \overset{a@\mathtt{ij}}{\twoheadrightarrow} s'' \overset{[\![\varphi']\!]}{\twoheadrightarrow} s$ and the induction hypothesis suffices.
2. The proof of the statement 2a is by induction on the length of f and the proof of 2b is by contradiction.
   (a) The base case is when $s_0 \overset{\bar{a}@\mathtt{ij}}{\twoheadrightarrow} \overset{a@\mathtt{ij}}{\twoheadrightarrow} s$, in other words $s_0 \overset{[\![\vec{a}]\!]}{\twoheadrightarrow} s$. Now, in order to obtain $S(KS_{\mathcal{A}})$ through Definition 12 there must exist two automata such that $\vec{q}_{(\mathtt{i})} = \vec{q}_{s(\mathtt{i})}$ and $\vec{q}_{(\mathtt{j})} = \vec{q}_{s(\mathtt{j})}$. Consequently, the product automaton $\mathcal{A}$ has the transition $(\vec{q_0}, \vec{a}, \vec{q})$ where $\forall \mathtt{k} \neq \mathtt{i}, \mathtt{j}.\ \vec{q}_{0_{(\mathtt{k})}} = \vec{q}_{(\mathtt{k})}$. Thus $\vec{q} \sim \vec{q^s}$.
   For the inductive case we have $\varphi = \vec{a}\,\varphi'$ and the computation $s_0 \overset{[\![\vec{a}]\!]}{\twoheadrightarrow} s'' \overset{[\![\varphi']\!]}{\twoheadrightarrow} s$ where $s'' = (\vec{q^{s''}}; \vec{u''})$. The same argument used above guarantees that there exists a transition $(\vec{q_0}, \vec{a}, \vec{q''})$ and $\vec{q''} \sim \vec{q^{s''}}$ and we conclude by applying the induction hypothesis.
   (b) Assume by contradiction that $\vec{q}_0 \xrightarrow{\hat{\varphi}} \vec{q'}$ has traversed no liable transitions. Then by Definition 11 there exists φ' such that $\vec{q'} \xrightarrow{\varphi'} \vec{q''}$ and $\vec{q''}$ is a final configuration. By Lemma 1 we must have $\vec{q}_{mpc} \xrightarrow{\hat{\varphi}\varphi'} \vec{q'}_{mpc}$ where $\vec{q'}_{mpc}$ is a final configuration. Hence by applying the first item of Theorem 1 we have $s \overset{f'}{\twoheadrightarrow} s''$ where s'' is a final configuration and $f' = [\![\varphi']\!]$ or $f' = a@\mathtt{ij}[\![\varphi']\!]$, obtaining a contradiction. □

As a side comment to item 2b, we can also prove that if s is a deadlock configuration (and either $f = [\![\varphi]\!]$ or $f = [\![\varphi]\!]\bar{a}@\mathtt{ij}$ for some $\varphi \in \mathfrak{Z}$) and $\mathcal{A}$ enjoys the branching condition, then there exists a run $\vec{q}_0 \xrightarrow{\hat{\varphi}} \vec{q}'$ traversing a transition in $TLiable(KS_{\mathcal{A}}/\mathcal{A})$ with either $\hat{\varphi} = \varphi$ or $\hat{\varphi} = \varphi\vec{a}$, where $\vec{a}$ is a match on a and $snd(\vec{a}) = \mathtt{i}, rcv(\vec{a}) = \mathtt{j}$. Note also that, if a contract automaton $\mathcal{A}$ fires through a liable transition, it can be that $S(KS_{\mathcal{A}})$ never reaches a deadlock configuration, as shown by the Example 9.

We are now ready to state a main result of ours: the controller of a contract automaton has the branching condition if and only if the corresponding communicating system is convergent.

Theorem 2. *Let $\mathcal{A}$ be a contract automaton, $KS_{\mathcal{A}}$ be its most permissive controller, and $S(KS_{\mathcal{A}})$ be the corresponding communicating system. Then*

$S(KS_{\mathcal{A}})$ is convergent if and only if $KS_{\mathcal{A}}$ satisfies the branching condition.

Proof. Let $\vec{q}_0$, $\vec{q}_{mpc}$, and s_0 be the initial configurations of $\mathcal{A}$, $KS_{\mathcal{A}}$, and $S(KS_{\mathcal{A}})$, respectively.

(*Only if-part*) Assume by contradiction that $S(KS_{\mathcal{A}})$ is not convergent, i.e. $s_0 \overset{f}{\twoheadrightarrow} s = (\vec{q^s}; \vec{u})$ and no final configurations are reachable from s. By Property 2, f is either $f = [\![\varphi]\!]$ or $f = [\![\varphi]\!]\vec{a'}@\mathtt{i'j'}$. Hence by applying item 2b of Theorem 1 we have that $\vec{q}_0 \xrightarrow{\hat{\varphi}} \vec{q'}$ has traversed a transition in $TLiable(KS_{\mathcal{A}}/\mathcal{A})$ with either $\hat{\varphi} = \varphi$ or $\hat{\varphi} = \varphi\vec{a'}$. Hence $\hat{\varphi} = \varphi_1\vec{a}\varphi''$ for some $\varphi_1, \vec{a}, \varphi''$ such that $\vec{q_0} \xrightarrow{\varphi_1} \vec{q_1}$ is a run of $\mathcal{A}$ and $(\vec{q_1}, \vec{a}, \vec{q_1'}) \in TLiable(KS_{\mathcal{A}}/\mathcal{A})$, with $snd(\vec{a}) = \mathtt{i}$, $rcv(\vec{a}) = \mathtt{j}$ for some $\mathtt{i}, \mathtt{j} \in \mathcal{P}$.

There exists then a (sub-)computation $s_0 \overset{[\![\varphi_1]\!]}{\twoheadrightarrow} s' = (\vec{q^{s'}}; \vec{u'}) \overset{\bar{a}@\mathtt{ij}}{\twoheadrightarrow}$, and by item 2a of Theorem 1 we have $\vec{q^{s'}} \sim \vec{q}_1$. Since we obtained $S(KS_{\mathcal{A}})$ through Definition 12, the transition $s' \overset{\bar{a}@\mathtt{ij}}{\twoheadrightarrow}$ has been originated by a transition $(\vec{q_2}, \vec{a}, \vec{q_3})$ in $KS_{\mathcal{A}}$, for some $\vec{q_2}$ such that $\vec{q_{2(i)}} = \vec{q_{1(i)}}$. We have that $\vec{q}_1 \neq \vec{q}_2$ because $(\vec{q_1}, \vec{a}, \vec{q_1'}) \in TLiable(KS_{\mathcal{A}}/\mathcal{A})$. Hence there must be $\varphi_1 = \varphi_1^1\varphi_1^2$ such that $\vec{q_0} \xrightarrow{\varphi_1^1} \vec{q}_2 \xrightarrow{\varphi_1^2} \vec{q}_1$ and the principal (corresponding to participant) $\mathtt{i}$ stays idle in each step of φ_1^2, since $\vec{q_{2(i)}} = \vec{q_{1(i)}}$. The transitions of $KS_{\mathcal{A}}$ include $\vec{q}_2 \xrightarrow{\bar{a}}$, but not $\vec{q}_1 \xrightarrow{\bar{a}}$ and since $\vec{q_{2(i)}} = \vec{q_{1(i)}}$, $KS_{\mathcal{A}}$ violates the branching condition, against our hypothesis.

(*If-part*) By contradiction assume that the branching condition does not hold in $KS_{\mathcal{A}}$, i.e. in $KS_{\mathcal{A}}$ $\vec{q}_1 \xrightarrow{\vec{a}} \vec{q''}$, $\vec{q}_2 \not\xrightarrow{\vec{a}}$ where $\vec{a}$ is a match on a with $snd(\vec{a}) = \mathtt{i}$, $rcv(\vec{a}) = \mathtt{j}$ for some $\mathtt{i}, \mathtt{j} \in \mathcal{P}$ and $\vec{q_{1(i)}} = \vec{q_{2(i)}}$. Suppose that φ and φ' are such that $\vec{q}_{mpc} \xrightarrow{\varphi} \vec{q}_1$ and $\vec{q}_{mpc} \xrightarrow{\varphi'} \vec{q}_2$.

By the first item of Theorem 1 we have $s_0 \xrightarrow{[\![\varphi]\!]} s \xrightarrow{\bar{a}@\mathtt{ij}} \hat{s} \xrightarrow{a@\mathtt{ij}} s'' = (\vec{q^s}; \vec{u})$ with $\vec{q^s} \sim \vec{q''}$ and $s_0 \xrightarrow{[\![\varphi']\!]} s'$. Moreover, since by hypothesis $\vec{q_{1(i)}} = \vec{q_{2(i)}}$, we also have $s' \xrightarrow{\bar{a}@\mathtt{ij}} \bar{s}$. The computation $\bar{s} \xrightarrow{a@\mathtt{ij}[\![\varphi_2]\!]} s_f$, for any φ_2 and s_f final, is not possible, because otherwise by item 2a of Theorem 1 we would have $\vec{q}_2 \xrightarrow{\vec{a}\varphi_2} \vec{q}_f$, with $\vec{q}_f$ final state of the automaton $\mathcal{A}$, as well of $KS_{\mathcal{A}}$. This would be a contradiction, because $\vec{q}_2 \not\xrightarrow{\vec{a}}$ by hypothesis. Hence $S(KS_{\mathcal{A}})$ is not convergent, since from $\bar{s}$ it is not possible to reach a final configuration. □

A consequence of Theorem 2 is that a *strongly safe* contract automaton has the branching condition if and only if its corresponding communicating system is convergent.

Corollary 3. *Let $\mathcal{A}$ be a contract automaton, then*

$\mathcal{A}$ is strongly safe and satisfies the branching condition if and only if $S(\mathcal{A})$ is convergent.

Proof. The statement follows by applying Theorem 2, because if $\mathcal{A}$ is strongly safe then $\mathcal{A} = KS_{\mathcal{A}}$, hence $KS_{\mathcal{A}}$ has the branching condition. □

Example 10. Consider the automaton $\mathcal{A} = \mathtt{A} \otimes \mathtt{B} \otimes \mathtt{C} \otimes \mathtt{D}$ depicted in Fig. 6. Both principals $\mathtt{A}$ and $\mathtt{B}$ only offer $\bar{a}$ and then stop, while the others principals $\mathtt{B}$ and $\mathtt{C}$ only perform the complementary request a.

The contract automaton $\mathcal{A}$ is strongly safe, but it does not enjoy the branching condition. As an example of violation, notice that the state of $\mathtt{B}$ in both $\vec{q}_1, \vec{q}_3$ is the same, i.e. $\vec{q_{1(2)}} = \vec{q_{3(2)}}$. But the match transition $(\square, \bar{a}, \square, a)$ is possibly from state $\vec{q}_1$, and not from the state $\vec{q}_3$; also from state $\vec{q}_3$ we can fire the match transition $(\square, \bar{a}, a, \square)$, which is not available in state $\vec{q}_1$.

The translation of $\mathcal{A}$ yields the communicating machines:

$$[\![KS_{\mathcal{A}}]\!]_{\mathtt{A}} = \bar{a}@\mathtt{AC} + \bar{a}@\mathtt{AD} \qquad [\![KS_{\mathcal{A}}]\!]_{\mathtt{B}} = \bar{a}@\mathtt{BC} + \bar{a}@\mathtt{BD}$$

$$[\![KS_{\mathcal{A}}]\!]_{\mathtt{C}} = a@\mathtt{AC} + a@\mathtt{BC} \qquad [\![KS_{\mathcal{A}}]\!]_{\mathtt{D}} = a@\mathtt{AD} + a@\mathtt{BD}$$

A deadlock configuration is generated by the trace $\bar{a}@\mathtt{AC}.a@\mathtt{AC}.\bar{a}@\mathtt{BC}$.

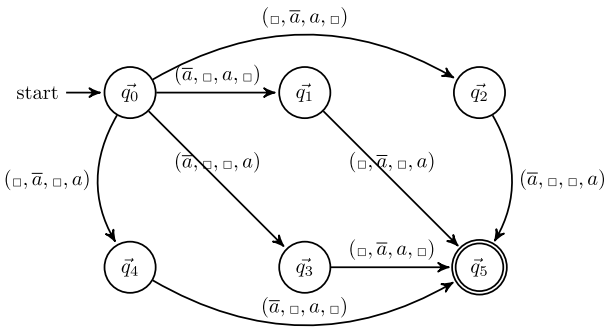

Fig. 6. $KS_{\mathcal{A}}$.

6. Extensions: agreement and asynchrony

We now extend our proposal along two lines. First we consider the more permissive notion of agreement on contract automata introduced in [3]. Then, we drop the constraints on the number of messages that a buffer can contain, and we consider the fully asynchronous semantics of communicating machines of Definition 6.

6.1. Agreement

The notion of *agreement* of [3] considers as forbidden actions only the non-matching requests while for strong agreement (Definition 7) both non-matching requests and offers are forbidden.

Below, we extend Theorem 2 and establish a correspondence between contract automata enjoying the property of *agreement* [3] and the convergent communicating system with the one buffer semantics. To do that, we introduce a special kind of contract automaton, called *environment*. Then, we extend the branching condition of Definition 15.

We borrow from [3] the definition of agreement and of most permissive controller in this case.

Definition 17 *(Agreement and Controller).* (See [3].) An *agreement on* $\mathbb{L}$ is a finite sequence of match or offer actions. We let $\mathfrak{A}$ to denote the set of all agreements on $\mathbb{L}$. A contract automaton $\mathcal{A}$ is *safe* if $\mathscr{L}(\mathcal{A}) \subseteq \mathfrak{A}$. We say that $\mathcal{A}$ *admits agreement* when $\mathscr{L}(\mathcal{A}) \cap \mathfrak{A} \neq \emptyset$.

A *controller* of $\mathcal{A}$ is a contract automaton $\mathcal{K}_{\mathcal{A}}$ such that $\mathscr{L}(\mathcal{K}_{\mathcal{A}}) \subseteq \mathfrak{A} \cap \mathscr{L}(\mathcal{A})$. The *most permissive controller of* $\mathcal{A}$ is the controller $\mathcal{K}_{\mathcal{A}}$ such that $\mathscr{L}(\mathcal{K}'_{\mathcal{A}}) \subseteq \mathscr{L}(\mathcal{K}_{\mathcal{A}})$ for all $\mathcal{K}'_{\mathcal{A}}$ controllers of $\mathcal{A}$.

The construction of the most permissive controller $\mathcal{K}_{\mathcal{A}}$ is basically the one of Lemma 1 (see for more details [3]), except that only request transitions are removed.

The above notion of agreement does not prevent offer actions to occur in isolation, and the following definition accordingly extends the branching condition of Definition 15.

Definition 18 *(Extended Branching Condition).* A contract automaton $\mathcal{A}$ has the *extended branching condition* if and only if it has the *branching condition* and for each $\vec{q}_1, \vec{q}_2$ reachable in $\mathcal{A}$ the following holds

$$\forall \vec{a} \text{ offer action } .(\vec{q}_1 \xrightarrow{\vec{a}} \wedge\, snd(\vec{a}) = \mathtt{i} \wedge \vec{q}_{1(i)} = \vec{q}_{2(i)}) \text{ implies } \vec{q}_2 \xrightarrow{\vec{a}}$$

To easily handle the offer actions that are now admitted, we introduce a special contract automaton, called *environment*, that "captures" them all. In this way we can re-use the notion of strong agreement. The environment has a single state, both initial and final and a request loop transition for each possible offer.

Definition 19 *(Environment).* The *environment* is the contract automaton

$$\mathtt{E} = \langle \{q_e\}, q_e, \mathbb{L}, \{(q_e, a, q_e) \mid a \in \mathbb{R}\}, \{q_e\} \rangle$$

Note that by composing a contract automaton $\mathcal{A}$ with the environment $\mathtt{E}$, that is $\mathcal{A} \otimes \mathtt{E}$, all the offer actions of $\mathcal{A}$ are turned into match actions with the environment.

The next theorem shows how the controller of a contract automaton $\mathcal{A}$ and the controller of $\mathcal{A} \otimes \mathtt{E}$ are related by the two branching conditions of Definition 15 and Definition 18.

Theorem 3. *Given the most permissive controller $\mathcal{K}_{\mathcal{A}}$ for the contract automaton $\mathcal{A}$, then $\mathcal{K}_{\mathcal{A}}$ has the extended branching condition if and only if $\mathcal{K}_{\mathcal{A}\otimes\mathtt{E}}$ has the branching condition.*

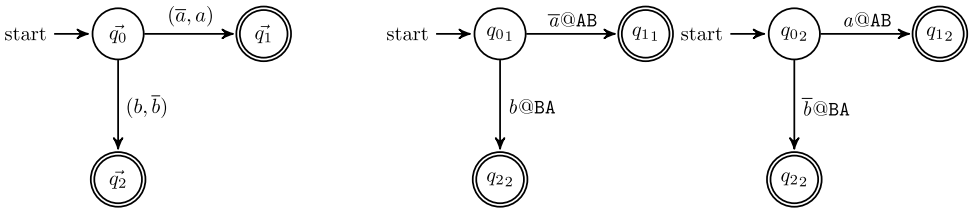

Fig. 7. From left to right: the contract automaton (with mixed choices) $\mathtt{A} \otimes \mathtt{B}$ (where $\mathtt{A} = \bar{a} + b$ and $\mathtt{B} = \bar{b} + a$), and its corresponding communicating machines $[\![\mathcal{K}_{\mathtt{A}\otimes\mathtt{B}}]\!]_{\mathtt{A}}$, $[\![\mathcal{K}_{\mathtt{A}\otimes\mathtt{B}}]\!]_{\mathtt{B}}$.

Proof.

(if) By hypothesis $\mathcal{K}_{\mathcal{A}}$ has the extended branching condition. By Definition 3 all offers of $\mathcal{K}_{\mathcal{A}}$ are turned into matches with the environment in $\mathcal{K}_{\mathcal{A}\otimes\mathtt{E}}$. By contradiction assume that $\mathcal{K}_{\mathcal{A}\otimes\mathtt{E}}$ does not hold the branching condition.

Hence there must be two states $\vec{q}_1, \vec{q}_2$ and a principal $\mathtt{p}$ such that $\vec{q}_1 \xrightarrow{\vec{a}}, \vec{q}_2 \not\xrightarrow{\vec{a}}$ with $\vec{q_1}_{(\mathtt{p})} = \vec{q_2}_{(\mathtt{p})}$ where $\vec{a}$ is a match with $snd(\vec{a}) = \mathtt{p}$.

Moreover it must be that $rcv(\vec{a}) = \mathtt{E}$ (by abuse of notation $\mathtt{E}$ is the principal corresponding to the environment); otherwise the match transition would also be in $\mathcal{K}_{\mathcal{A}}$, obtaining a contradiction. By Definition 3 the match $\vec{a}$ in $\mathcal{K}_{\mathcal{A}\otimes\mathtt{E}}$ is turned into an offer action $\vec{a}'$ in $\mathcal{K}_{\mathcal{A}}$, and we have that $\mathcal{K}_{\mathcal{A}}$ does not hold the extended branching condition since $\vec{q}_1 \xrightarrow{\vec{a}'}, \vec{q}_2 \not\xrightarrow{\vec{a}'}$.

(only if) By hypothesis we have that $\mathcal{K}_{\mathcal{A}\otimes\mathtt{E}}$ has the branching condition. By Definition 3 we know that in the contract automaton $\mathcal{K}_{\mathcal{A}\otimes\mathtt{E}}$ all the matches involving the environment are coupled with offers of $\mathcal{K}_{\mathcal{A}}$.

By contradiction assume that $\mathcal{K}_{\mathcal{A}}$ does not hold the extended branching condition. Hence there must be two states $\vec{q}_1, \vec{q}_2$ and a principal $\mathtt{p}$ such that $\vec{q}_1 \xrightarrow{\vec{a}}, \vec{q}_2 \not\xrightarrow{\vec{a}}$ with $\vec{q_1}_{(\mathtt{p})} = \vec{q_2}_{(\mathtt{p})}$ and $snd(\vec{a}) = \mathtt{p}$.

We distinguish two cases:

- $\vec{a}$ is a match action: then by Definition 3 the match is present also in $\mathcal{K}_{\mathcal{A}\otimes\mathtt{E}}$, obtaining a contradiction since $\mathcal{K}_{\mathcal{A}\otimes\mathtt{E}}$ has the branching condition.
- $\vec{a}$ is an offer action: then by Definition 3 $\vec{a}$ is turned into a match $\vec{a}'$ with $\mathtt{E}$ in $\mathcal{K}_{\mathcal{A}\otimes\mathtt{E}}$. Since $\vec{q}_1 \xrightarrow{\vec{a}'}, \vec{q}_2 \not\xrightarrow{\vec{a}'}$ we have that $\mathcal{K}_{\mathcal{A}\otimes\mathtt{E}}$ does not hold the branching condition, obtaining a contradiction. □

Finally we relate a controller $\mathcal{K}_{\mathcal{A}}$ of a contract automaton $\mathcal{A}$ with the corresponding system $S(\mathcal{K}_{\mathcal{A}\otimes\mathtt{E}})$, obtained from the controller composed with the environment. Indeed as a direct consequence of Theorem 2 and Theorem 3 we have that $\mathcal{K}_{\mathcal{A}}$ has the extended branching condition if and only if $S(\mathcal{K}_{\mathcal{A}\otimes\mathtt{E}})$ is convergent.

Corollary 4. *Given the most permissive controller $\mathcal{K}_{\mathcal{A}}$ for the contract automaton $\mathcal{A}$, then*
$\mathcal{K}_{\mathcal{A}}$ has the extended branching condition iff $S(\mathcal{K}_{\mathcal{A}\otimes\mathtt{E}})$ is convergent.

Proof. *(if)* By hypothesis $\mathcal{K}_{\mathcal{A}}$ has the extended branching condition, by Theorem 3 we have that $\mathcal{K}_{\mathcal{A}\otimes\mathtt{E}}$ has the branching condition. Finally by Theorem 2 we have that $S(\mathcal{K}_{\mathcal{A}\otimes\mathtt{E}})$ is convergent.

(only if) By hypothesis $S(\mathcal{K}_{\mathcal{A}\otimes\mathtt{E}})$ is convergent, by applying Theorem 2 we have that $\mathcal{K}_{\mathcal{A}\otimes\mathtt{E}}$ has the branching condition. Finally by Theorem 3 we have that $\mathcal{K}_{\mathcal{A}}$ has the extended branching condition. □

6.2. Asynchronous semantics of communicating systems

We now discuss the relations between contract automata and communicating systems, with the semantics of Definition 6, i.e. when the buffers of the communicating machines can contain more then one message. From now onward, the notions of *convergence* and *deadlock* introduced in Definition 13 are considered using the semantics of Definition 6. The following example shows the difficulties with the unrestricted semantics.

Example 11. Consider the contract automaton $\mathtt{A} \otimes \mathtt{B}$ and the corresponding communicating machines of Fig. 7. This contract automaton is strongly safe and has the branching condition. However the translated system is *not* convergent. Indeed a possible deadlock in $S(\mathcal{K}_{\mathtt{A}\otimes\mathtt{B}})$ occurs if $\mathtt{A}$ performs the action $\bar{a}$@AB and then $\mathtt{B}$ performs the action $\bar{b}$@BA. This is because participant $\mathtt{B}$ can ignore the message received by the participant $\mathtt{A}$ and follow the other branch of the contract automaton. These behaviours are not permitted by the 1-buffer semantics.

In order to guarantee that a system of communicating machines corresponding to a contract automaton is convergent, we constrain the contract automata as follows. Intuitively, we will discard those contract automata that have a request and an offer transition of a principal outgoing from the same node, like the automata $\mathtt{A}$ and $\mathtt{B}$ of Example 11. More precisely:

Definition 20. Let $\mathcal{A}$ be a contract automaton, then $\mathcal{A}$ has a *mixed choice* if and only if there exists a reachable state $\vec{q}$ with two outgoing transitions $\vec{q} \xrightarrow{\vec{a}_1}, \vec{q} \xrightarrow{\vec{a}_2}$ such that $snd(\vec{a}_1) = rcv(\vec{a}_2)$.

In the following we will prove that if the strong controller of a contract automaton $\mathcal{A}$ has the branching condition and no mixed choices then the corresponding system is convergent.

It is convenient to define how to project a trace of a communicating system into its offer actions, and to prove an auxiliary property.

Definition 21. Given $f \in \mathsf{Act}^*$, its projection on its offers is defined as follows:

$$f\lfloor_{\mathbb{O}} \stackrel{\text{def}}{=} \begin{cases} \bar{a}@\mathtt{ij}\,(f'\lfloor_{\mathbb{O}}) & \text{if } f = \bar{a}@\mathtt{ij}\,f' \text{ for some } \mathtt{i},\mathtt{j} \in \mathcal{P}, \bar{a} \in \mathbb{O} \\ f'\lfloor_{\mathbb{O}} & \text{if } f = a@\mathtt{ij}\,f' \text{ for some } \mathtt{i},\mathtt{j} \in \mathcal{P}, a \in \mathbb{R} \\ \varepsilon & \text{if } f = \varepsilon \end{cases}$$

By abuse of notation, we say that a communicating machine has no mixed choices if it is never the case that both offer and request transitions leave one of its states.

Property 5. *Let $KS_{\mathcal{A}}$ be the most permissive controller of $\mathcal{A}$ and $S(KS_{\mathcal{A}})$ be the corresponding communicating system.*

If $KS_{\mathcal{A}}$ has the branching condition and no mixed choices then all the communicating machines of the participants in $S(KS_{\mathcal{A}})$ have no mixed choices.

Proof. By contradiction assume that there is a participant $\mathtt{p}$ with two transitions $(q_1, \bar{a}@\mathtt{pj}, q_2), (q_1, b@\mathtt{ip}, q_3)$.

By Definition 12 there must be two transitions $(\vec{q}, \vec{a}, \vec{q_1}), (\vec{q_2}, \vec{b}, \vec{q_3})$ where $\vec{q}_{(\mathtt{p})} = \vec{q_2}_{(\mathtt{p})}$, and $\vec{a}, b$ are match actions on a and b, respectively, with $snd(\vec{a}) = \mathtt{p}$, $rcv(\vec{a}) = \mathtt{j}$, $snd(\vec{b}) = \mathtt{i}$, $rcv(\vec{b}) = \mathtt{p}$.

There are the following two cases:

- $\vec{q} = \vec{q_2}$ we obtain a contradiction since we have a mixed choice;
- $\vec{q} \neq \vec{q_2}$ then since $\vec{q}_{(p)} = \vec{q_2}_{(p)}$ by the branching condition there are also two transitions $(\vec{q_2}, \vec{a}, \vec{q_4}), (\vec{q}, \vec{b}, \vec{q_5})$, hence both $\vec{q}, \vec{q_2}$ are mixed choices, obtaining a contradiction. □

The following theorem relates the traces of a communicating system with the ones of the strong controller $KS_{\mathcal{A}}$ it comes from, provided that $KS_{\mathcal{A}}$ has the branching condition and no mixed choices. Under the above conditions a trace of a communicating system only differs from the corresponding trace of the $KS_{\mathcal{A}}$ in the order in which the request actions are fired in the system. Indeed by considering only offer actions the two traces are *equal*.

Theorem 4. *Let $KS_{\mathcal{A}}$ be the most permissive controller of $\mathcal{A}$, with the branching condition and no mixed choices, and let Φ be the set of its non-empty traces; let $S(KS_{\mathcal{A}})$ be the communicating system obtained from $KS_{\mathcal{A}}$ and let F be the set of its non-empty traces. Then*

$$\forall f \in F \text{ there exists } \varphi \in \Phi \text{ such that } f\lfloor_{\mathbb{O}} = [\![\varphi]\!]\lfloor_{\mathbb{O}} .$$

Proof. Let $\vec{q}_{mpc}$ and s be the initial configurations of $KS_{\mathcal{A}}$ and $S(KS_{\mathcal{A}})$, respectively. Assume by contradiction that there exists a f with $s \xrightarrow{f}$ such that for all φ with $\vec{q}_{mpc} \xrightarrow{\varphi}$ we have $f\lfloor_{\mathbb{O}} \neq [\![\varphi]\!]\lfloor_{\mathbb{O}}$.

Since $f \neq \varepsilon$, by Definition 6 it must be that $f = \overline{a_1}@\mathtt{i}_1\mathtt{j}_1 f_1'$ for some $\mathtt{i}_1, \mathtt{j}_1, \bar{a}_1, f_1'$, and by Definition 12 there must be a transition $(\vec{q}_{mpc}, \vec{a'}, \vec{q'}_{mpc})$ in $KS_{\mathcal{A}}$, and therefore there is a $\varphi = \vec{a'}\varphi_1'$, for some $\vec{q'}_{mpc}$ and φ_1', where $\vec{a'}$ is a match on a_1 with $snd(\vec{a'}) = \mathtt{i}_1, rcv(\vec{a'}) = \mathtt{j}_1$. Moreover by hypothesis it must be that $f_1'\lfloor_{\mathbb{O}} \neq [\![\varphi']\!]\lfloor_{\mathbb{O}}$.

Now split f as $f_1 f_2$, and select a trace $\varphi = \varphi_1\varphi_2$ such that f_1 is the longest prefix of f with $[\![\varphi_1]\!]\lfloor_{\mathbb{O}} = f_1\lfloor_{\mathbb{O}}$ ($\neq \varepsilon$ because of the above). Then we have $f_2 = \bar{a}@\mathtt{ij}f_3$ for some $\mathtt{i}, \mathtt{j}, \bar{a}$, and $\vec{q}_{mpc} \xrightarrow{\varphi_1} \vec{q}_{1_{mpc}} \xrightarrow{\vec{a}}\!\!\!\!\!\!/\;$ where $\vec{a}$ is a match with $snd(\vec{a}) = \mathtt{i}, rcv(\vec{a}) = \mathtt{j}$. Then, there must exist a transition $(\vec{q}_{2_{mpc}}, \vec{a}, \vec{q}_{2'_{mpc}})$, for some $\vec{q}_{2_{mpc}}, \vec{q}_{2'_{mpc}}$. Assuming $s \xrightarrow{f_1} (\vec{q}_r, \vec{u})$, we have then that $\vec{q}_{r(\mathtt{i})} = \vec{q}_{2_{mpc_{(\mathtt{i})}}}$.

Since the branching condition holds, it turns out that $\vec{q}_{1_{mpc_{(\mathtt{i})}}} \neq \vec{q}_{2_{mpc_{(\mathtt{i})}}}$. If the principal (corresponding to participant) $\mathtt{i}$ stays idle in φ_1 then it must be $\vec{q}_{mpc_{(\mathtt{i})}} = \vec{q}_{1_{mpc_{(\mathtt{i})}}} = \vec{q}_{2_{mpc_{(\mathtt{i})}}}$ obtaining a contradiction. Thus, the principal $\mathtt{i}$ has performed some steps in φ_1, more precisely some requests, otherwise we would obtain $\vec{q}_{1_{mpc_{(\mathtt{i})}}} = \vec{q}_{2_{mpc_{(\mathtt{i})}}}$, because $[\![\varphi_1]\!]\lfloor_{\mathbb{O}} = f_1\lfloor_{\mathbb{O}}$.

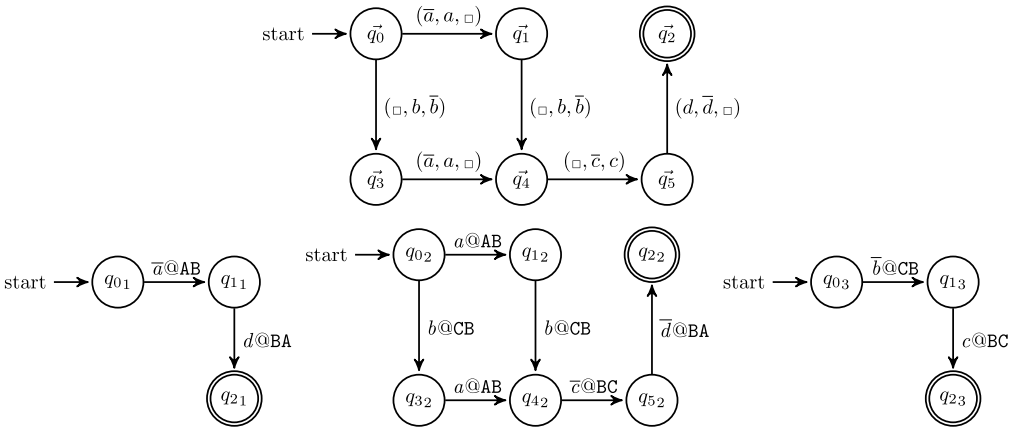

Fig. 8. Top: $KS_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}}$. Bottom: $[\![KS_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}}]\!]_{\mathtt{A}}$, $[\![KS_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}}]\!]_{\mathtt{B}}$, $[\![KS_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}}]\!]_{\mathtt{C}}$.

We now introduce an auxiliary notion of projection on the offers of a participant $\mathtt{p}$:

$$\hat{f}\lfloor_{\mathbb{O},\mathtt{p}} \stackrel{\text{def}}{=} \begin{cases} \bar{a}@\mathtt{p}\mathtt{j}\ \hat{f}'\lfloor_{\mathbb{O},\mathtt{p}} & \text{if } \hat{f} = \bar{a}@\mathtt{p}\mathtt{j}\hat{f}' \text{ for some } \mathtt{j} \in \mathcal{P}, \bar{a} \in \mathbb{O} \\ \hat{f}'\lfloor_{\mathbb{O},\mathtt{p}} & \text{if } \hat{f} = a@\mathtt{i}\mathtt{j}\hat{f}' \text{ for some } \mathtt{i},\mathtt{j} \in \mathcal{P}, a \in \mathbb{R} \\ \hat{f}'\lfloor_{\mathbb{O},\mathtt{p}} & \text{if } \hat{f} = \bar{a}@\mathtt{i}\mathtt{j}\hat{f}' \text{ for some } \mathtt{i},\mathtt{j} \in \mathcal{P}, \mathtt{i} \neq \mathtt{p}, \bar{a} \in \mathbb{O} \\ \varepsilon & \text{if } \hat{f} = \varepsilon \end{cases}$$

We consider the offers of $\mathtt{i}$, if any.

- $[\![\varphi_1]\!]\lfloor_{\mathbb{O},\mathtt{i}} = f_1\lfloor_{\mathbb{O},\mathtt{i}} = \varepsilon$: if all the requests $a@\mathtt{i}\mathtt{j}$ occurring in (matches of) φ_1 also occur in f_1, then we obtain $\vec{q}_{1_{mpc_{(\mathtt{i})}}} = \vec{q}_{2_{mpc_{(\mathtt{i})}}}$, and there is a violation of the branching condition. Hence not all the requests of $\mathtt{i}$ in φ_1 occur in f_1, i.e. $\vec{q}_{mpc} \xrightarrow{\varphi_1^1} \vec{q}_{2_{mpc}} \xrightarrow{\varphi_1^2} \vec{q}_{1_{mpc}}$ with $\varphi_1 = \varphi_1^1\varphi_1^2$ and assume that the missing requests occur in φ_1^2. Then further split φ_1^2 as $\varphi_1^3\vec{a}_2\varphi_1^4$, where $rcv(\vec{a}_2) = \mathtt{i}$, so that $\vec{q}_{2_{mpc}} \xrightarrow{\varphi_1^3} \vec{q}_{4_{mpc}} \xrightarrow{\vec{a}_2\varphi_1^4} \vec{q}_{1_{mpc}}$ and that principal $\mathtt{i}$ stays idle in φ_1^3. Then we have that $\vec{q}_{2_{mpc_{(\mathtt{i})}}} = \vec{q}_{4_{mpc_{(\mathtt{i})}}}$ and by the branching condition we have the transition $(\vec{q}_{4_{mpc}}, \vec{a}, \vec{q}_{4'_{mpc}})$ showing a mixed choice, because the principal $\mathtt{i}$ can perform both an offer and a request: contradiction.
- $[\![\varphi_1]\!]\lfloor_{\mathbb{O},\mathtt{i}} = f_1\lfloor_{\mathbb{O},\mathtt{i}} \neq \varepsilon$: assume that the last offer of $\mathtt{i}$ in $[\![\varphi_1]\!]$ is $\overline{a_5}@\mathtt{i}\mathtt{j}_2$, i.e. $[\![\varphi_1]\!]\lfloor_{\mathbb{O},\mathtt{i}} = [\![\varphi_1^1]\!]\lfloor_{\mathbb{O},\mathtt{i}}\ \overline{a_5}@\mathtt{i}\mathtt{j}_2$, with $[\![\varphi_1]\!] = [\![\varphi_1^1]\!]\overline{a_5}@\mathtt{i}\mathtt{j}_2[\![\varphi']\!]$ and $[\![\varphi']\!]\lfloor_{\mathbb{O},\mathtt{i}} = \varepsilon$. Also, let $\vec{q}_{mpc} \xrightarrow{\varphi_1^1\vec{a}_5} \vec{q}_{5_{mpc}}$ where $\vec{a}_5$ is a match on a_5 with $snd(\vec{a}_5) = \mathtt{i}, rcv(\vec{a}_5) = \mathtt{j}_2$. Now, we show that any requests of $\mathtt{i}$ occurring in $[\![\varphi_1^1]\!]$, also occur in f_1 before the offer $\overline{a_5}@\mathtt{i}\mathtt{j}_2$. Indeed, by Property 5 we have that the communicating machine of the participant $\mathtt{i}$ has no mixed choices (so it cannot choose between offering or requesting) and is therefore forced to read from its buffer the received offers before firing $\overline{a_5}@\mathtt{i}\mathtt{j}_2$, because these were coupled (with corresponding requests of $\mathtt{i}$) in matches occurring in $[\![\varphi_1^1]\!]$.
  We conclude the proof by applying the argument of the previous case with $\vec{q}_{5_{mpc}}$ as the initial state, given that $[\![\varphi']\!]\lfloor_{\mathbb{O},\mathtt{i}} = \varepsilon$. □

Note in passing that for simulating a step in non-empty trace φ of a most permissive controller, the corresponding communicating system can pass through several possible configurations, in order to execute a sequence of actions f such that $f\lfloor_{\mathbb{O}} = [\![\varphi]\!]\lfloor_{\mathbb{O}}$.

As a matter of fact, a participant can fire many requests, not registered in $[\![\varphi]\!]\lfloor_{\mathbb{O}}$, after its last offer registered therein. Indeed the trace φ is formed by match actions, hence all the requests are fired, while in the trace f there could be some participant which has fired all the offers but not yet all its requests.

Example 12. Consider the automata depicted in Fig. 8. The controller $KS_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}}$ has the branching condition and no mixed choices. Three possible traces of $S(KS_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}})$ are:

$$f = \bar{a}@\mathtt{AB}\ \bar{b}@\mathtt{CB}\ a@\mathtt{AB}\ b@\mathtt{CB}\ \bar{c}@\mathtt{BC}\ \bar{d}@\mathtt{BA} \quad f_1 = f\ c@\mathtt{BC} \quad f_2 = f_1\ d@\mathtt{BA}$$

Consider the trace of $KS_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}}$:

$$\varphi = (\bar{a}, a, \square)(\square, b, \bar{b})(\square, \bar{c}, c)(d, \bar{d}, \square)$$

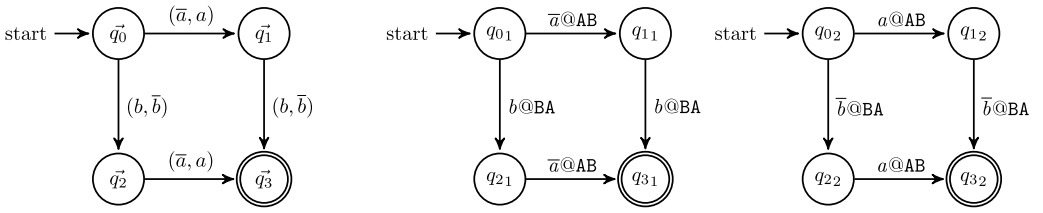

Fig. 9. From left to right: the contract automaton (with mixed choices) $\mathtt{A} \otimes \mathtt{B}$ (where $\mathtt{A} = \overline{a}.b + b.\overline{a}$ and $\mathtt{B} = \overline{b}.a + a.\overline{b}$), and its corresponding communicating machines $[\![\mathcal{K}_{\mathtt{A}\otimes\mathtt{B}}]\!]_{\mathtt{A}}$, $[\![\mathcal{K}_{\mathtt{A}\otimes\mathtt{B}}]\!]_{\mathtt{B}}$.

We have $f\downarrow_{\mathbb{O}} = f_1\downarrow_{\mathbb{O}} = f_2\downarrow_{\mathbb{O}} = [\![\varphi]\!]\downarrow_{\mathbb{O}}$. Note that in order to fire the offer $\overline{c}$@BC the participant C needs to fire all the previous requests. Moreover in the trace f_2 each participant has fired all its requests, indeed we have: $\vec{q}_{mpc} \xrightarrow{\varphi} \vec{q'}_{mpc}$, $s \xrightarrow{f_2} (\vec{q}_r; \vec{u})$ and $\vec{q'}_{mpc} \sim \vec{q}_r$, where $\vec{q}_{mpc}$ and s are the starting configurations of $KS_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}}$ and $S(KS_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}})$.

The main result of this sub-section follows. If the strong controller of a contract automaton $\mathcal{A}$ has the branching condition and no mixed choices, then its corresponding system is convergent with unrestricted semantics.

Theorem 5. *Let $S(KS_{\mathcal{A}})$ be the communicating system corresponding to the most permissive controller $KS_{\mathcal{A}}$. Then*

$KS_{\mathcal{A}}$ has the branching condition and no mixed choices implies $S(KS_{\mathcal{A}})$ is convergent.

Proof. Let s be the initial configuration of $S(KS_{\mathcal{A}})$ and $\vec{q}_{mpc}$ be the initial state of $KS_{\mathcal{A}}$. By contradiction assume that $KS_{\mathcal{A}}$ has the branching condition and no mixed choices but $S(KS_{\mathcal{A}})$ is not convergent, i.e. there exists a non-empty run f such that $s \xrightarrow{f} (\vec{q}_r; \vec{u})$ and no final configurations are reachable from $(\vec{q}_r; \vec{u})$. By Theorem 4 there exists a non-empty trace of the controller φ such that $\vec{q}_{mpc} \xrightarrow{\varphi} \vec{q'}_{mpc}$ and $f\downarrow_{\mathbb{O}} = [\![\varphi]\!]\downarrow_{\mathbb{O}}$. Since all the offers are matched in φ, by Definitions 6 and 12 there exists a trace where all the request actions occur, and let it be our f. Since both $KS_{\mathcal{A}}$ and $S(KS_{\mathcal{A}})$ are deterministic, and since f drives each communicating machine to execute all and only the actions occurring in the matches of φ, we have that $\vec{q}_r \sim \vec{q'}_{mpc}$.

Finally since $KS_{\mathcal{A}}$ is a controller, there must be a trace φ' and a final state $\vec{q}_f$ such that $\vec{q'}_{mpc} \xrightarrow{\varphi'} \vec{q}_f$. By applying Theorem 1.1 we obtain $(\vec{q}_r; \vec{u}) \xrightarrow{[\![\varphi']\!]} s_f$ where s_f is a final configuration, because the transition relation of the 1-buffer semantics is included in that of the semantics of Definition 6: contradiction. □

Note that the above theorem does not fully generalise Corollary 3, as shown by the following example.

Example 13. Consider the contract automaton of Fig. 9 that is strongly safe (so it is also its most permissive controller) and has the branching condition. Its corresponding system consists of the communicating machines in Fig. 9 and it is convergent. However, a mixed choice is possible from $\vec{q}_0 = (q_{01}, q_{02})$.

7. An example

We show our proposal at work on the *two buyers protocol* (2BP for short) presented in [31]. There are two buyers $\mathtt{B}_1$ and $\mathtt{B}_2$ that collaborate in purchasing an item from a seller S. Buyer $\mathtt{B}_1$ starts the protocol by asking S the price of the desired item ($\overline{price}$); the seller S replies with the quote for the requested item by sending the quotation message $\overline{quote}$ to both buyers. Once received its quote, buyer $\mathtt{B}_1$ sends to $\mathtt{B}_2$ its contribution for purchasing the item ($\overline{contrib}$).

Buyer $\mathtt{B}_2$ waits for the quote from S and the contribution from $\mathtt{B}_1$. Then, it decides whether to terminate by issuing the $\overline{nop}$ message to S, or to proceed by sending an acknowledgement to S.

Upon receiving the acknowledgement, the seller sends the item to $\mathtt{B}_2$ ($\overline{delivery}$), while if it receives *nop* it terminates with no further action.

The contract automata $\mathtt{B}_1$, $\mathtt{B}_2$ and S admit strong agreement, and they are shown in Fig. 10, together with their most permissive controller.

We now analyse the translation of the orchestration into a choreography of communicating finite state machines. The most permissive controller does not enjoy the branching condition because, for example:

- in $\vec{q}_2$ and $\vec{q}_4$, buyer $\mathtt{B}_1$ is in state q_{B1_2},
- $\vec{q}_4 \xrightarrow{(\overline{contrib}, contrib, \square)}$, but
- there is no transition from $\vec{q}_2$ such that $\vec{q}_2 \xrightarrow{(\overline{contrib}, contrib, \square)}$,

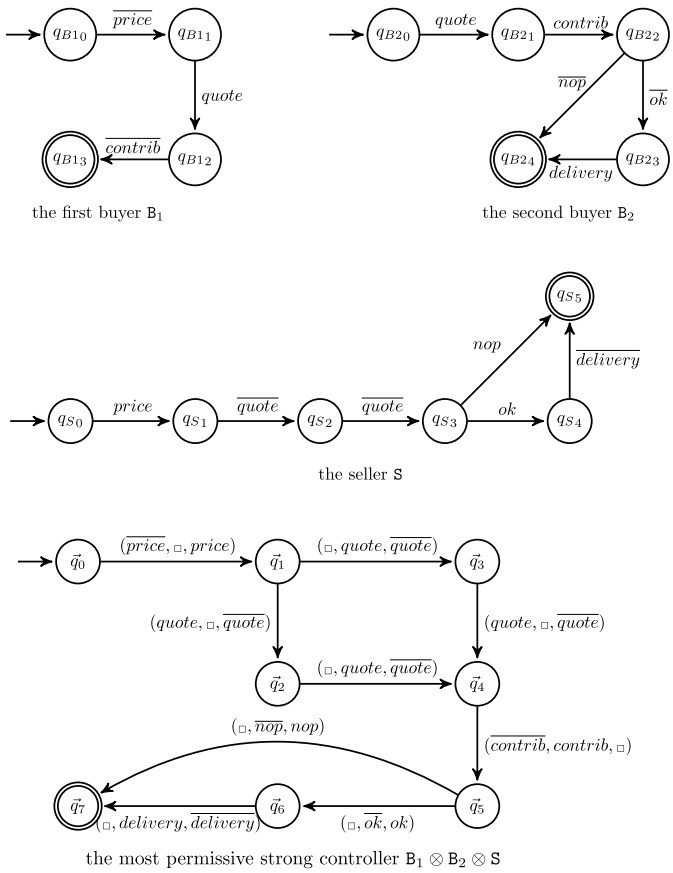

Fig. 10. The contract automata for 2BP and their most permissive controller.

as required by Definition 15. This happens because $\mathtt{B}_1$ could send the contribution to $\mathtt{B}_2$ before this has received *quote* from $\mathtt{S}$, so blocking the system. An easy recover would be forcing the seller to first send the quotation to buyer $\mathtt{B}_2$, so reducing the nondeterminism. A convergent choreography of communicating machines, both for the synchronous and the asynchronous case is then derivable as specified above.

The most permissive controller has no states with mixed choices. However, this case may arise if $\mathtt{B}_2$ could additionally withdraw before accepting the contribution from $\mathtt{B}_1$, because the quotation is too high. It is out of the scope of this paper discussing the ways to recover from this situation, and we only note that our proposal clearly detects why and where it shows up, making the corresponding choreography not convergent.

8. Concluding remarks

We have established a formal correspondence between contract automata, an orchestration model, and communicating systems, i.e. sets of communicating machines, that is a model of choreography. More precisely, we introduced the notion of strong agreement on contract automata, stating that all the traces accepted by the automata are made by requests matched by corresponding offers; on the second model we considered convergence, i.e. successful termination of computations. We proved that strong agreement corresponds to convergence (cf. Theorem 2) when communicating systems are endowed with the 1-buffer semantics, according to which the execution of the machines is basically synchronous. We note in passing that this has some advantages since communicating systems with the 1-buffer semantics are computationally more tractable than in the general case [30]. We then generalised the above result by adopting a more relaxed notion of agreement that admits computations where offer actions can go unmatched (cf. Corollary 4). We also proved a slightly weaker result for communicating systems with the unrestricted semantics (cf. Theorem 5). Currently, we are developing a verification toolkit based on the results presented here and in [3]; a prototype is available at https://github.com/davidebasile/workspace.

The constructions to obtain contract automata from principals (e.g., products of contract automata, controllers, controlled systems) could be complemented with the recent results on the synthesis of choreographies for communicating finite state machines proposed by [11]. We conjecture that the conditions guaranteeing well-behaviour identified here imply the *generalised multiparty compatibility* condition identified in [11]. For a given communicating system satisfying this property, a choreography always exists and can be synthesised effectively.

In [32] progress is attained under assumptions allowing a process P to expose unmatched actions as long as the closed system made of P and a *catalyser* process (derived from P) results to be lock-free. We can give an interesting analogy with this work: our environments resemble their *catalysers* and the extended branching condition (cf. Definition 13) is similar to their relaxed notion of safety. As far as we can see, our notion of convergence is stronger than the above notion of safety, in that we also guarantee that a final state is reached, possibly infinitely often. Additionally, only dyadic session types are dealt with in [32], while here we can cope with multiparty scenarios.

A possible application of the results in [11] is to use the projections obtained from the communicating system corresponding to the controller. Through them, we can identify the principals that could be liable. Indeed, each principal univocally corresponds to one of those projections. Hence, this would allow to flag the components that may lead to communication mismatches so to refine them and guarantee that the refined principals execute without the intervention of an orchestrator. In general, the suggested approach would give a more efficient and more distributed execution. This is because one can remove the overhead due to the communication with the orchestrator, and avoid the centralisation point of the orchestrator, respectively. Additionally, one could find that only few principals, and more importantly which of them, spoil the conditions for achieving choreographed executions. Hence, by modifying/replacing those principals with the machines obtained by projecting the synthesised choreography it would be possible to retrieve a choreographed executions of the contract automata.

References

[1] C. Peltz, Web services orchestration and choreography, IEEE Comput. 36 (10) (2003) 46–52.
[2] N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, Y. Lafon, C. Barreto, Web services choreography description language version 1.0, World Wide Web Consortium, Candidate Recommendation CR-ws-cdl-10-20051109, November 2005.
[3] D. Basile, P. Degano, G.L. Ferrari, Automata for analysing service contracts, in: M. Maffei, E. Tuosto (Eds.), Trustworthy Global Computing – 9th International Symposium, Revised Selected Papers, TGC 2014, Rome, Italy, in: Lecture Notes in Computer Science, vol. 8902, Springer, 2014, pp. 34–50.
[4] D. Brand, P. Zafiropulo, On communicating finite-state machines, J. ACM 30 (2) (1983) 323–342.
[5] M. Bartoletti, E. Tuosto, R. Zunino, On the realizability of contracts in dishonest systems, in: M. Sirjani (Ed.), Coordination Models and Languages – 14th International Conference, COORDINATION 2012, Proceedings, in: Lecture Notes in Computer Science, vol. 7274, Springer, 2012, pp. 245–260.
[6] M. Bartoletti, A. Scalas, E. Tuosto, R. Zunino, Honesty by typing, in: D. Beyer, M. Boreale (Eds.), Formal Techniques for Distributed Systems – Joint IFIP WG 6.1 International Conference, FMOODS/FORTE 2013, Proceedings, in: Lecture Notes in Computer Science, vol. 7892, Springer, 2013, pp. 305–320.
[7] P. Deniélou, N. Yoshida, Multiparty session types meet communicating automata, in: H. Seidl (Ed.), Programming Languages and Systems – 21st European Symposium on Programming, ESOP 2012, Proceedings, in: Lecture Notes in Computer Science, vol. 7211, Springer, 2012, pp. 194–213.
[8] C.G. Cassandras, S. Lafortune, Introduction to Discrete Event Systems, Springer, Secaucus, NJ, USA, 2006.
[9] D. Basile, P. Degano, G.L. Ferrari, E. Tuosto, From orchestration to choreography through contract automata, in: I. Lanese, A. Lluch-Lafuente, A. Sokolova, H.T. Vieira (Eds.), Proceedings 7th Interaction and Concurrency Experience, ICE 2014, in: EPTCS, vol. 166, 2014, pp. 67–85.
[10] I. Lanese, C. Guidi, F. Montesi, G. Zavattaro, Bridging the gap between interaction- and process-oriented choreographies, in: Software Engineering and Formal Methods, SEFM 2008, 2008, pp. 323–332.
[11] J. Lange, E. Tuosto, N. Yoshida, From communicating machines to graphical choreographies, in: S.K. Rajamani, D. Walker (Eds.), Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, ACM, 2015, pp. 221–232.
[12] J. Lange, A. Scalas, Choreography synthesis as contract agreement, in: Proceedings 6th Interaction and Concurrency Experience, ICE 2013, 2013, pp. 52–67.
[13] R. Dijkman, M. Dumas, Service-oriented design: a multi-viewpoint approach, Int. J. Coop. Inf. Syst. 13 (4) (2004) 337–368.
[14] G. Alonso, F. Casati, H. Kuno, V. Machiraju, Web Services – Concepts, Architectures and Applications. Data-Centric Systems and Applications, Springer, 2004.
[15] M. Papazoglou, Web Services and SOA: Principles and Technology, Pearson-Prentice Hall, 2012.
[16] J. Misra, Computation orchestration, in: M. Broy, J. Grünbauer, D. Harel, T. Hoare (Eds.), Engineering Theories of Software Intensive Systems, in: NATO Science Series, vol. 195, Springer, 2005, pp. 285–330.
[17] Z. Qiu, X. Zhao, C. Cai, H. Yang, Towards the theoretical foundation of choreography, in: C.L. Williamson, M.E. Zurko, P.F. Patel-Schneider, P.J. Shenoy (Eds.), Proceedings of the 16th International Conference on World Wide Web, WWW 2007, ACM, 2007, pp. 973–982.
[18] H. Yang, X. Zhao, C. Cai, Z. Qiu, Exploring the connection of choreography and orchestration with exception handling and finalization/compensation, in: J. Derrick, J. Vain (Eds.), Formal Techniques for Networked and Distributed Systems – FORTE 2007, 27th IFIP WG 6.1 International Conference, Proceedings, in: LNCS, vol. 4574, Springer, 2007, pp. 81–96.
[19] S.R. Talbot, Orchestration and choreography: standards, tools and technologies for distributed workflows, http://www.nettab.org/2005/docs/NETTAB2005_Ross-TalbotOral.pdf.
[20] N. Busi, R. Gorrieri, C. Guidi, R. Lucchi, G. Zavattaro, Choreography and orchestration conformance for system design, in: P. Ciancarini, H. Wiklicky (Eds.), Coordination Models and Languages, 8th International Conference, COORDINATION 2006, Proceedings, in: Lecture Notes in Computer Science, vol. 4038, Springer, 2006, pp. 63–81.
[21] I. Lanese, F. Montesi, G. Zavattaro, Amending choreographies, in: A. Ravara, J. Silva (Eds.), Proceedings 9th International Workshop on Automated Specification and Verification of Web Systems, WWV 2013, in: EPTCS, vol. 123, 2013, pp. 34–48.
[22] S. Basu, T. Bultan, M. Ouederni, Deciding choreography realizability, in: J. Field, M. Hicks (Eds.), Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012, ACM, 2012, pp. 191–202.
[23] S. Hallé, T. Bultan, Realizability analysis for message-based interactions using shared-state projections, in: G. Roman, K.J. Sullivan (Eds.), Proceedings of the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2010, ACM, 2010, pp. 27–36.
[24] C. Laneve, L. Padovani, An algebraic theory for web service contracts, Form. Asp. Comput. (2015) 1–28.
[25] S. Basu, T. Bultan, Choreography conformance via synchronizability, in: S. Srinivasan, K. Ramamritham, A. Kumar, M.P. Ravindra, E. Bertino, R. Kumar (Eds.), Proceedings of the 20th International Conference on World Wide Web, WWW 2011, ACM, 2011, pp. 795–804.
[26] M. Bravetti, G. Zavattaro, Contract-based discovery and composition of web services, in: M. Bernardo, L. Padovani, G. Zavattaro (Eds.), Formal Methods for Web Services, 9th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, 2009, in: Lecture Notes in Computer Science, vol. 5569, Springer, 2009, pp. 261–295.
[27] M. Bravetti, G. Zavattaro, Contract compliance and choreography conformance in the presence of message queues, in: R. Bruni, K. Wolf (Eds.), Web Services and Formal Methods, 5th International Workshop, WS-FM 2008, Revised Selected Papers, in: Lecture Notes in Computer Science, vol. 5387, Springer, 2008, pp. 37–54.

[28] M. Autili, D.D. Ruscio, A.D. Salle, P. Inverardi, M. Tivoli, A model-based synthesis process for choreography realizability enforcement, in: V. Cortellessa, D. Varró (Eds.), Fundamental Approaches to Software Engineering – 16th International Conference, FASE 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, 2013. Proceedings, in: Lecture Notes in Computer Science, vol. 7793, Springer, 2013, pp. 37–52.
[29] L. de Alfaro, T.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, ACM, 2001, pp. 109–120.
[30] G. Cécé, A. Finkel, Verification of programs with half-duplex communication, Inf. Comput. 202 (2) (2005) 166–190.
[31] K. Honda, N. Yoshida, M. Carbone, Multiparty asynchronous session types, in: G.C. Necula, P. Wadler (Eds.), POPL, ACM, 2008, pp. 273–284.
[32] M. Carbone, O. Dardha, F. Montesi, Progress as compositional lock-freedom, in: E. Kühn, R. Pugliese (Eds.), COORDINATION 2014, in: Lecture Notes in Computer Science, vol. 8459, Springer, 2014, pp. 49–64.