From Orchestration to Choreography through Contract Automata *

Davide Basile Pierpaolo Degano Gian-Luigi Ferrari
Dipartimento di Informatica, Università di Pisa, Italy
{basile,degano,giangi}@di.unipi.it
Emilio Tuosto
Computer Science Department, University of Leicester
emilio@le.ac.uk

We study the relations between a contract automata and an interaction model. In the former model, distributed services are abstracted away as automata - oblivious of their partners - that coordinate with each other through an orchestrator. The interaction model relies on channel-based asynchronous communication and choreography to coordinate distributed services.

We define a notion of strong agreement on the contract model, exhibit a natural mapping from the contract model to the interaction model, and give conditions to ensure that strong agreement corresponds to well-formed choreography.

1 Introduction

We investigate the relations between two models of distributed coordination: *contract automata* [3] and *communicating machines* [4].

The former 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 fact, such model envisages components capable of communicating messages on some ports according to an automaton specifying the component's 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" happen. In this way, it is possible to transfer the approach of [2, 1] 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 $\overline{a}$ and $\overline{b}$ 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 Alice, Bob, and Carol correspond to the following regular expressions, used here for conciseness:

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

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 fulfill their contracts. In fact, Alice can

*This work has been partially supported by the MIUR project *Security Horizons* and IST-FP7-FET open-IP project *ASCENS*

I. Lanese, A. Lluch Lafuente, A. Sokolova, H. T. Vieira (Eds.):
7th Interaction and Concurrency Experience (ICE 2014)
EPTCS 166, 2014, pp. 67–85, doi:10.4204/EPTCS.166.8

give her aeroplane to Bob or Carol, while Carol can receive the bike from Alice or Bob, therefore if Alice gives her aeroplane to Carol the contracts of Alice and Carol are fulfilled while Bob's contract is not. In the model of contract automaton the coordinator acts as the mam 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).

Communicating machines - the other model we consider here - were introduced with the aim of studying distributed communication protocols and ensure the correctness of distributed components again formalised as automata. But - unlike contract automata - communicating machines do not require an orchestrator since they interact directly with each other through (FIFO) buffers. In fact, a relation between communicating and distributed choreographies has been recently proved in [8].

We show that these models - invented to address different problems and having different coordination mechanisms - are related. For this purpose, we introduce the notion of *strong agreement*, which requires the fulfillment of all offers *and* requests. Strong agreement differs from previous notions of agreements for contract automata (cf. Section 3) and enables us to introduce strongly safe contract automata, that is those automata accepting only computations that are in strong agreement. Strong agreement and safety are key to establish a correspondence from contract automata to communicating machines.

Indeed if a contract automaton enjoys strong safety (and it is well-behaved on branching constructs) then the corresponding communicating machines are a well-formed choreography.

Structure of the paper. We recall contract automata and communicating finite-state machines in Section 2. The new notion of agreement on contract automata is in Section 3. The translation of contract automata into communicating machines is given in Section 4 where we also prove our main theorem. In Section 5 we discuss possible extensions of our results to other notions of agreement for contract automata and semantics for communicating machines. Finally, concluding remark are in Section 6.

2 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.

2.1 Contract Automata

Before recalling contract automata (introduced in [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).

Transitions of contract automata will be labelled with elements in the set $\mathbb{L} \stackrel{\text{def}}{=} \mathbb{R} \cup \mathbb{O} \cup \{\square\}$ where

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

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

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

A contract automaton (cf. Def. 2) represents the behaviour of a set of participants (possibly made of a single participant) capable of performing some *actions*; more precisely, as formalised in Def. 1, the actions of a contract automaton allow them to " advertise" offers, "make" requests, or "handshake" on simultaneous offer/request matches.

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

- *a* request (action) on *b iff $\vec{a}$ is of the form $\square^* b \square^*$ with $b \in \mathbb{R}$*
- *an* offer (action) on *b iff $\vec{a}$ is of the form $\square^* b \square^*$ with $b \in \mathbb{O}$*
- *a* match (action) on *b iff $\vec{a}$ is of the form $\square^* b \square^* \overline{b} \square^*$ with $b \in \mathbb{R} \cup \mathbb{O}$.*

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 $\vec{a}_1 \dot{\bowtie} \vec{a}_2$ iff

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

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

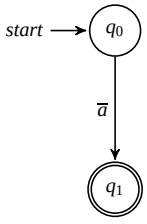
Fact 1 *$\bowtie$ is an equivalence relation on $\mathbb{L}^*$.*

Definition 2 (Contract Automata) *Let $\mathscr{Q}$ (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 $\mathscr{A} = \langle \mathscr{Q}^n, \vec{q}_0, \mathbb{L}^n, T, F \rangle$, where*

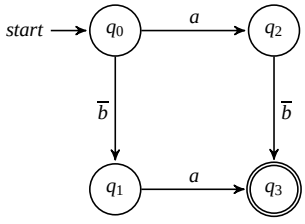
- *$\vec{q}_0 \in \mathscr{Q}^n$ is the* initial *state*
- *$F \subseteq \mathscr{Q}^n$ is the set of* accepting *states*
- *$T \subseteq \mathscr{Q}^n \times \mathbb{L}^n \times \mathscr{Q}^n$ is the set of* transitions *such that $(\vec{q}, \vec{a}, \vec{q}') \in T$ iff*
  - *if $\vec{a}_{(i)} = \square$ then $\vec{q}_{(i)} = \vec{q}'_{(i)}$ (i.e. , the i-th participant stays idle) and*
  - *$\vec{a}$ is either a request, or an offer, or else a match action*

A principal *is a contract automaton $\mathscr{A}$ of rank 1 such that, for any two transitions (q_1, a_1, q'_1), (q_2, a_2, q'_2) in $\mathscr{A}$, it is not the case that $a_1 \bowtie a_2$.*

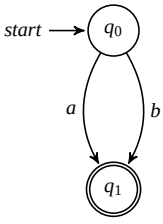
Example 1 *The principals of Alice, Bob, and Carol in Section 1 are given below*



Automaton of Alice



Automaton of Bob



Automaton of Carol

Given a contract automaton $\mathscr{A} = \langle \mathscr{Q}^n, \vec{q}_0, \mathbb{L}^n, T, F \rangle$ of rank *n*, usual definitions and constructions of finite-state automata apply. In particular,

- the configurations of $\mathscr{A}$ are pairs in $\mathscr{Q}^n \times (\mathbb{L}^n)^*$ of strings of *n*-tuples of labels and states of $\mathscr{A}$;

- $\mathscr{A}$ *moves from* $(\vec{q},w)$ *to* $(\vec{q}',w')$, written $(\vec{q},w)\xrightarrow{\vec{a}}(\vec{q}',w')$, iff $w=\vec{a}w'$ and $(\vec{q},\vec{a},\vec{q}')\in T$; we write $(\vec{q},w)\rightarrow(\vec{q}',w')$ when $\vec{a}$ is immaterial and $\vec{q}\xrightarrow{\vec{a}}\vec{q}'$ when w is immaterial;
- the *language of* $\mathscr{A}$ is $\mathscr{L}(\mathscr{A})=\{w \mid (\vec{q}_0,w)\rightarrow^*(\vec{q},\varepsilon),\vec{q}\in F\}$ where $\rightarrow^*$ is the reflexive and transitive closure of $\rightarrow$. As usual, $s_1\xrightarrow{\ell_1\cdots\ell_m}s_{m+1}$ shortens $s_1\xrightarrow{\ell_1}s_2\cdots s_m\xrightarrow{\ell_m}s_{m+1}$ (for some $s_2,\ldots,s_m$) and $s\not\rightarrow$ iff for no s' it is the case that $s\rightarrow s'$.

We now borrow from [3] the product operation of contract automata. 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) *Let* $\mathscr{A}_i=\langle\mathscr{Q}^{n_i},\vec{q}_{0i},\mathbb{L}^{n_i},T_i,F_i\rangle$ *be contract automata of rank* n_i*, for* $i\in\{1,\ldots,h\}$*. The* product *of* $\mathscr{A}_1,\ldots,\mathscr{A}_h$*, denoted as* $\bigotimes_{i\in\{1,\ldots,h\}}\mathscr{A}_i$*, is the contract automaton* $\langle\mathscr{Q}^n,\vec{q}_0,\mathbb{L}^n,T,F\rangle$ *of rank* $n=n_1+\ldots+n_h$ *where:*

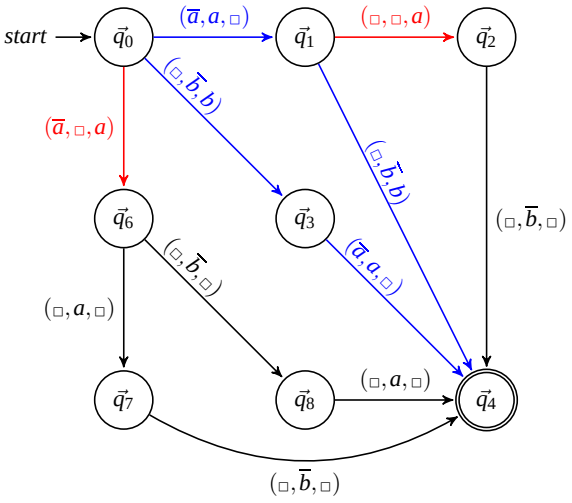
- $\vec{q}_0=\vec{q}_{01}\ldots\vec{q}_{0h}$
- $F=\{\vec{q}_1\ldots\vec{q}_h \mid \forall i\in 1\ldots h:\ \vec{q}_i\in F_i\}$
- T *is the least subset of* $\mathscr{Q}^n\times\mathbb{L}^n\times\mathscr{Q}^n$ *such that* $(\vec{q},\vec{c},\vec{q}')\in T$ *iff, letting* $\vec{q}=\vec{q}_1\ldots\vec{q}_h\in\mathscr{Q}^n$*,*

  either *there are* $1\leq i<j\leq h$ *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*

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

  or $\vec{c}_{(i)}=\vec{a}_i$, $\vec{c}_{(l)}=\square^{n_l}$ *for each* $l\neq i\in\{1,\ldots,h\}$*, and* $\vec{q}'=\vec{q}_1\ldots\vec{q}_{i-1}\vec{q}_i'\vec{q}_{i+1}\ldots\vec{q}_h$ *when* $(\vec{q}_i,\vec{a}_i,\vec{q}_i')\in T_i$ *and for all* $j\neq i$ *and* $(\vec{q}_j,\vec{a}_j,\vec{q}_j')\in T_j$ *it does not hold that* $\vec{a}_i\bowtie\vec{a}_j$*.*

Example 2 *The contract automaton below is the product of the contract automata in Example 1.*



Notice that from the states $\vec{q}_0$ *and* $\vec{q}_6$ *(where participants can handshake) only match actions depart; offer and request actions are not included in the product.*

Remark 1 *Notice that the product in Def. 3 is not associative; an alternative (but more complex) definition of associative product can be given by "breaking" existing matches when composing automata [3].*

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

2.2 Communicating Machines

Communicating machines [4] are a simple automata-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 [7] to our needs; in particular, the only relevant difference with the original model is that we have to add the set of final states. Let $\mathscr{P}$ be a finite set of *participants* (ranged over by $\mathtt{p}$, $\mathtt{q}$, $\mathtt{r}$, $\mathtt{s}$, etc.) and $C \stackrel{\text{def}}{=} \{\mathtt{pq} \mid \mathtt{p},\mathtt{q} \in \mathscr{P} \text{ and } \mathtt{p} \neq \mathtt{q}\}$ be the set of *channels*.

Remark 2 *The set $\mathscr{P}$ can be thought of as the set of integers $\{1,\ldots,n\}$ (and likewise for contract automata). However, we adopt a different notation to make the translation from contract automata to communicating machines clearer.*

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 *sending* of a from machine $\mathtt{s}$ to $\mathtt{r}$) and, 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) *A* communicating finite-state machine *is an automaton $M = (Q, q_0, \mathbb{R} \cup \mathbb{O}, \delta, F)$ where Q is a finite set of* states*, $q_0 \in Q$ is the* initial *state, $\delta \subseteq Q \times Act \times Q$ is a set of* transitions*, and $F \subseteq Q$ is the set of final states. We say that M is* deterministic *iff for all states $q \in Q$ and all actions $\ell \in Act$, if $(q,\ell,q'),(q,\ell,q'') \in \delta$ then $q' = q''$. Also, we write $\mathscr{L}(M) \subseteq Act^*$ for the language on Act accepted by the automaton corresponding to machine M.*

We will consider only deterministic CFSMs. The notion of deterministic CFSMs adopted here differs from the standard one 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 $q' = q''$ (see e.g., [7]). The reason for the definition is to reflect the semantics of contract automata.

The communication model of CFSMs (cf. Definitions 5 and 6) is based on (unbounded) FIFO buffers - the channels in C - used by participants to exchange messages. To spare another syntactic category and cumbersome definitions, we draw the messages appearing in the buffers of CFSMs from the set of requests $\mathbb{R}$. Recall that the set of participants $\mathscr{P}$ is finite.

Definition 5 (Communicating systems) *Given a CFSM $M_\mathtt{p} = (Q_\mathtt{p}, q_{0\mathtt{p}}, \mathbb{R} \cup \mathbb{O}, \delta_\mathtt{p}, F_\mathtt{p})$ for each $\mathtt{p} \in \mathscr{P}$, the tuple $S = (M_\mathtt{p})_{\mathtt{p}\in\mathscr{P}}$ is a* communicating system *(*CS*). A* configuration *of S is a pair $s = (\vec{q};\vec{w})$ where $\vec{q} = (q_\mathtt{p})_{\mathtt{p}\in\mathscr{P}}$ with $q_\mathtt{p} \in Q_\mathtt{p}$ and where $\vec{w} = (w_\mathtt{pq})_{\mathtt{pq}\in C}$ with $w_\mathtt{pq} \in \mathbb{R}^*$; component $\vec{q}$ is the* control state *and $q_\mathtt{p} \in Q_\mathtt{p}$ is the* local state *of machine $M_\mathtt{p}$. The* initial configuration *of S is $s_0 = (\vec{q}_0;\vec{\varepsilon})$ with $\vec{q}_0 = (q_{0\mathtt{p}})_{\mathtt{p}\in\mathscr{P}}$.*

Hereafter, we fix a machine $M_\mathtt{p} = (Q_\mathtt{p}, q_{0\mathtt{p}}, \mathbb{R} \cup \mathbb{O}, \delta_\mathtt{p}, F_\mathtt{p})$ for each participant $\mathtt{p} \in \mathscr{P}$ and let $S = (M_\mathtt{p})_{\mathtt{p}\in\mathscr{P}}$ be the corresponding system.

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

1. *either $\ell = \overline{a}@\mathtt{sr}$ and $(\vec{q}_{(\mathtt{s})},\ell,\vec{q}'_{(\mathtt{s})}) \in \delta_\mathtt{s}$, $\vec{q}'_{(\mathtt{p})} = \vec{q}_{(\mathtt{p})}$ for all $\mathtt{p} \neq \mathtt{s}$, and $\vec{w}'_{(\mathtt{sr})} = \vec{w}_{(\mathtt{sr})} \cdot a$ and, for all $\mathtt{pq} \neq \mathtt{sr}$, $\vec{w}'_{(\mathtt{pq})} = \vec{w}_{(\mathtt{pq})}$*

2. *or* $\ell = a@\mathtt{sr}$ *and* $(\vec{q}_{(\mathtt{r})}, \ell, \vec{q}'_{(\mathtt{r})}) \in \delta_{\mathtt{r}}$, $\vec{q}'_{(\mathtt{p})} = \vec{q}_{(\mathtt{p})}$ *for all* $\mathtt{p} \neq \mathtt{r}$, *and* $\vec{w}'_{(\mathtt{sr})} = a \cdot \vec{w}_{(\mathtt{sr})}$ *and* $\vec{w}'_{(\mathtt{pq})} = \vec{w}_{(\mathtt{pq})}$ *for all* $\mathtt{pq} \neq \mathtt{sr}$.

We write $s \to s'$ *for* $\exists \ell : \ s \xrightarrow{\ell} s'$ *and denote with* $\to^*$ *the reflexive and transitive closure of* $\to$. *The set of* reachable configurations of S *is* $RS(S) = \{s \mid s_0 \to^* s\}$. *A sequence of transitions is* k-bounded *if no channel of any intermediate configuration on the sequence contains more than* k *messages.*

Condition (1) in Def. 6 puts the content a on a channel `sr`, while (2) gets the content a from `sr`.

2.3 Notational Synopsis

To avoid their continuous repetition, through the paper we assume fixed a contract automaton $\mathscr{A} = \langle \mathscr{Q}^n, \vec{q}_0, \mathbb{L}^n, T, F \rangle$ of rank n.

For readability we summarise the notations introduced so far in the following table.

X^*	set of finite words on a set X; ε is the empty word
$w_{(i)}$	the i-th symbol of w
$\lvert w \rvert$	the length of w
x^n (resp. x^*)	x concatenated n-times (resp. arbitrarily many) with itself
$\vec{x}$ or $(x_i)_{1 \leq i \leq n}$	indexed tuples
$\mathbb{L}$	labels (ranged over by a, b, c, etc.)
$\mathbb{R}$	request labels
$\mathbb{O}$	offer labels
$\square \notin \mathbb{R} \cup \mathbb{O}$	idle label
$\mathscr{A}$	contract automata of rank n
$\mathscr{P}$	set of participants (ranged over by `p`, `q`, `i`, `j`,`A`, `B`,`C`, etc.)
C	set of channels (ranged over by `pq`)
$M_{\mathtt{p}}$	communicating machine of participant `p`
S	a system of communicating machines

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

3 Enforcing Agreement

This section introduces a new notion of agreement on contract automata - called *strong agreement* - that elaborates the notions of *agreement* and *weak agreement* introduced in [3]. The three notions differ on the conditions for the fulfillment of an interaction between different principals. Briefly, an *agreement* exists if all the requests, but not necessarily all the offers, are satisfied synchronously. Intuitively, this means that the orchestrator "simultaneously" guarantees two participants that their complementary actions are matched. Instead, a *weak agreement* exists when request actions can be performed "on credit". In other words, a computation yelds weak agreement when the fulfillment of a request action can happen after the action has been taken. Intuitively, this corresponds to an asynchronous communication admitting actions taken on credit provided that obligations will be honored later on.

Here, we focus on *strong agreement*, which strengthens the previous notion of agreement by requiring the fulfillment of all offers *and* requests in a synchronous way. In Section 4 we will show how this condition corresponds to interactions between communicating machines.

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

A contract automaton $\mathscr{A}$ *is* strongly safe *if* $\mathscr{L}(\mathscr{A}) \subseteq \Im$, *otherwise it is* strongly unsafe. *We say that* $\mathscr{A}$ admits strong agreement *when* $\mathscr{L}(\mathscr{A}) \cap \Im \neq \emptyset$.

Note that ε does not belong to $\Im$; the reason is that ε would not be an interesting agreement because it does not require any interaction (neither with the controller nor between principals). For this we require that the initial states of contract automata are not accepting states.

We show how to generate a strongly safe composition of contracts with an approach borrowed by the supervisory control theory for discrete event systems [6]. In this theory, discrete event systems are basically automata where accepting states represent the successful termination of a few tasks while *forbidden states* are those that should not be traversed in "good" computations. The purpose is then to synthesize a controller that enforces this property. 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 [6].

Since the behaviors that we want to enforce in $\mathscr{A}$ are exactly those traces labeled by words in $\Im \cap \mathscr{L}(\mathscr{A})$, we 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* $\mathscr{A}$ *is a contract automaton* $KS_{\mathscr{A}}$ *such that* $\mathscr{L}(KS_{\mathscr{A}}) \subseteq \Im \cap \mathscr{L}(\mathscr{A})$. *The* most permissive (strong) controller *(MPC) of* $\mathscr{A}$ *is the controller* $KS_{\mathscr{A}}$ *such that* $\mathscr{L}(KS'_{\mathscr{A}}) \subseteq \mathscr{L}(KS_{\mathscr{A}})$ *for all* $KS'_{\mathscr{A}}$ *controllers of* $\mathscr{A}$.

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

Example 3 *The MPC of the contract automaton in Example 2 consists of the states* $\vec{q}_0$, $\vec{q}_1$, $\vec{q}_3$, *and* $\vec{q}_4$ *with transitions* $(\vec{q}_0,(\overline{a},a,\square),\vec{q}_1)$, $(\vec{q}_3,(\overline{a},a,\square),\vec{q}_4)$, $(\vec{q}_0,(\square,\overline{b},b),\vec{q}_3)$, *and* $(\vec{q}_1,(\square,\overline{b},b),\vec{q}_4)$.

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

Proof. By contradiction, assume $\mathscr{L}(KS_{\mathscr{A}}) \subset \Im \cap \mathscr{L}(\mathscr{A})$. Since $\Im \cap \mathscr{L}(\mathscr{A})$ is the intersection of two regular languages and all actions are controllable, there exists a contract automaton $KS'_{\mathscr{A}}$ accepting it (cf. [6]). By definition, $KS'_{\mathscr{A}}$ is a controller of $\mathscr{A}$ strictly containing $\mathscr{L}(KS_{\mathscr{A}})$, contradicting the hypothesis that $\mathscr{L}(KS_{\mathscr{A}})$ is the most permissive controller. ◇

A state $\vec{q}$ of a contract automaton $\mathscr{A}$ is called *redundant* if, and only if, from $\vec{q}$ no accepting state of $\mathscr{A}$ can be reached.

Lemma 2 (MPC) *A contract automaton is the most permissive controller of* $\mathscr{A}$ *if, and only if, it is language-equivalent to*

$$KS_{\mathscr{A}} \stackrel{def}{=} \langle \mathscr{Q}^n, \vec{q}_0, \mathbb{L}^n, T' \setminus \{(\vec{q},a,\vec{q}') \mid \vec{q} \text{ or } \vec{q}' \text{ is redundant in } \mathscr{K}\}, F\rangle$$

where $\mathscr{K} = \langle \mathscr{Q}^n, \vec{q}_0, \mathbb{L}^n, \{t \in T \mid t \text{ is a match transition}\}, F\rangle$ *is the sub-automaton of* $\mathscr{A}$ *consisting of the match transitions of* $\mathscr{A}$ *only.*

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

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

Example 4 *The MPC of Example 3 is obtained from the CA in Example 2 by applying the construction of Lemma 2.*

The *controlled system* of a contract automaton $\mathscr{A}$ identifies the match transitions of $\mathscr{A}$ and those transitions that lead "outside" of the controller; for this we use a distinguished state $\bot \notin \mathscr{Q}^n$ (for any n) in the following definition.

Definition 9 (Controlled system) *Let $KS_{\mathscr{A}} = \langle \mathscr{Q}^n, \vec{q}_0, \mathbb{L}^n, T' \subseteq T, F \rangle$ be the MPC of $\mathscr{A}$ as computed in Lemma 2. The* controlled system of $\mathscr{A}$ under $KS_{\mathscr{A}}$ *is defined as the automaton $KS_{\mathscr{A}}/\mathscr{A} = \langle \mathscr{Q}^n \cup \{\bot\}, \vec{q}_0, \mathbb{L}^n, T'', F \rangle$ such that*

$$T'' \quad = \quad T' \quad \cup \quad \{(\vec{q}, \vec{a}, \bot) \;\big|\; \vec{q} \text{ reachable from } \vec{q}_0 \text{ in } KS_{\mathscr{A}} \text{ and } \exists \vec{q}' \in \mathscr{Q}^n : \; (\vec{q}, \vec{a}, \vec{q}') \in T \setminus T'\}$$

Example 5 *The controlled system of the CA in Example 2 is obtained by adding the transitions $(\vec{q}_1, (\square, \square, a), \vec{q}_2), (\vec{q}_0, (\overline{a}, \square, a), \vec{q}_6)$ to the MPC of Example 3.*

It is worth remarking that the transitions reaching $\bot$ in the controlled system of $\mathscr{A}$ identify the start of the computations in $\mathscr{A}$ 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 10 (Strong Liability) *Given a controlled system $KS_{\mathscr{A}}/\mathscr{A}$, the set of* liable *participants on a trace $w \in \mathscr{L}(\mathscr{A})$ is given by:*

$$\mathit{Liable}(KS_{\mathscr{A}}/\mathscr{A}, w) = \{1 \leq \mathtt{i} \leq n \mid (\vec{q}_0, w) \to^* (\vec{q}, \vec{a}w') \to (\bot, w) \text{ in } KS_{\mathscr{A}}/\mathscr{A}, \vec{a}_{(\mathtt{i})} \neq \square\}$$

The potentially liable principals *in $KS_{\mathscr{A}}/\mathscr{A}$ are $\mathit{Liable}(KS_{\mathscr{A}}/\mathscr{A}) \stackrel{def}{=} \bigcup_{w \in \mathscr{L}(\mathscr{A})} \mathit{Liable}(KS_{\mathscr{A}}/\mathscr{A}, w)$.*

We let $\mathit{TLiable}(KS_{\mathscr{A}}/\mathscr{A})$ to denote the set of transitions of $\mathscr{A}$ that make principals liable.

Note that the transition labelled by $\vec{a}$ in Definition 10 is the first which diverges from the expected path (since, by Definition 9, state $\bot$ does not have outgoing transitions). Indeed a liable index identifies a principal that fires an action taking the computation away from agreements.

Example 6 *The liable indexes of the contract automaton in Example 2 are* 1 *and* 3*, corresponding to Alice and Carol respectively; the transitions that make them liable are respectively $\big(\vec{q}_0,\ (\overline{a}, \square, a),\ \vec{q}_6\big)$ and $\big(\vec{q}_1,\ (\square, \square, a),\ \vec{q}_2\big)$. The former liable transition is a match that leads to a non-match transition.*

Note that labels allow us to track participants firing actions so to find (the indexes of) the liable principals. Our aim is to restrict the behaviour of principals so that they follow only the traces of the automaton which lead to strong agreement, while avoiding the others.

4 From Contract Automata to Communicating Machines

The translation of a principal into a communicating machine is conceptually straightforward. Indeed, the translation just yields a machine isomorphic to a principal in the composed contract automaton; the only difference are the labels. To account for the "openness" nature of contract automata - where principals can fire transitions not matched by other principals - in Definition 11 below we use the '$-$' symbol representing a special ("anonymous") participant distinguished by the participants corresponding to the principals and playing the role of the environment. For this reason, we will assume from now on that actions in Act are built on $C \cup \{-\}$.

Definition 11 (Translation) *The translation $[\![\vec{a}]\!]_{\mathtt{p}} \in Act$ of an action $\vec{a}$ on $\mathbb{L}^n$ respect to a participant* $\mathtt{p}$ *(with $1 \leq \mathtt{p} \leq n$) is defined as:*

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

The translation of $\mathscr{A}$ to a CFSM *is given by the map*

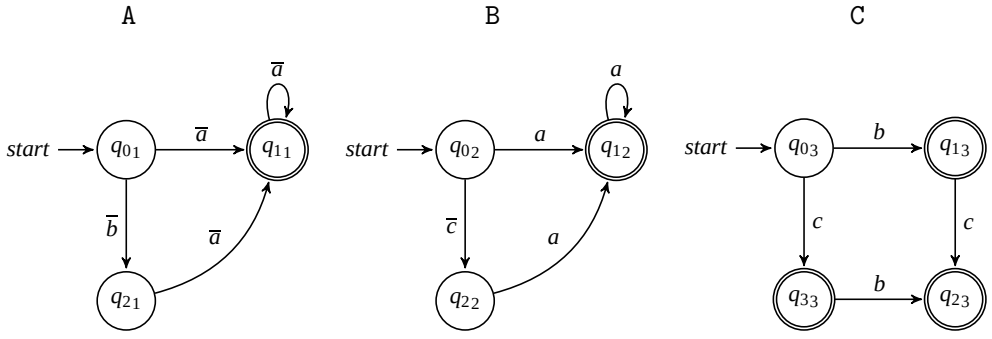
$$[\![\mathscr{A}]\!]_{\mathtt{p}} \stackrel{def}{=} \langle \mathscr{Q}, \vec{q}_{0(p)}, Act, \{(\vec{q}_{(p)}, [\![\vec{a}]\!]_{\mathtt{p}}, \vec{q}'_{(p)}) \mid (\vec{q}, \vec{a}, \vec{q}') \in T \text{ and } [\![\vec{a}]\!]_{\mathtt{p}} \neq \varepsilon\}, F \rangle$$

We denote with $S(\mathscr{A}) = ([\![\mathscr{A}]\!]_{\mathtt{p}})_{\mathtt{p} \in \{1,\ldots,n\}}$ the communicating system obtained by translating the contract automaton $\mathscr{A}$.

Given $\varphi \in \left(\mathbb{L}^n\right)^*$, we define

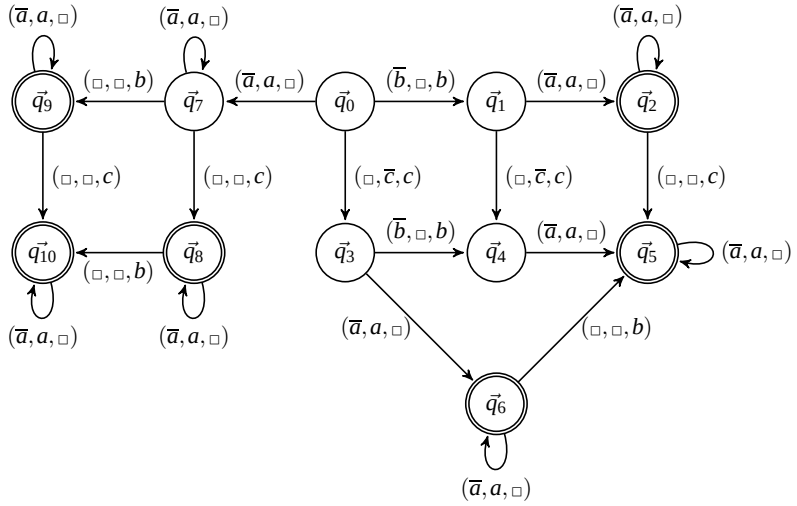
$$[\![\varphi]\!] \stackrel{\text{def}}{=} \begin{cases} \overline{a}@\mathtt{ij}\ a@\mathtt{ij}[\![\varphi']\!] & \text{if } \varphi = \vec{a}\varphi' \text{ and } \vec{a} \text{ is a match action on } a \text{ with } \vec{a}_{(i)} \in \mathbb{O} \text{ and } \vec{a}_{(j)} \in \mathbb{R} \\ \overline{a}@\mathtt{i}-[\![\varphi']\!]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

Example 7 *Consider the following principal CAs:*



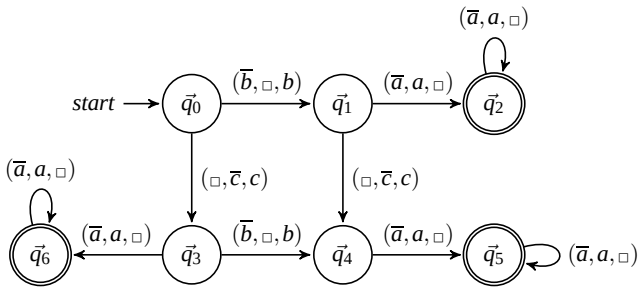
The product is the contract automaton below with initial state $\vec{q}_0 = \langle q_{01}, q_{02}, q_{03} \rangle$

$\mathtt{A} \otimes \mathtt{B} \otimes \mathtt{C}$



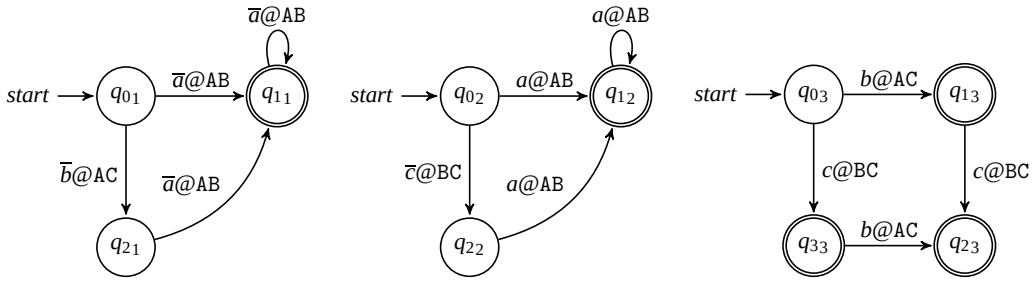
By applying Lemma 2 on the product automaton we obtain the MPC:

$KS_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}}$



The translation of Definition 11 yelds the CMs:

$[\![KS_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}}]\!]_{\mathtt{A}}$ $[\![KS_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}}]\!]_{\mathtt{B}}$ $[\![KS_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}}]\!]_{\mathtt{C}}$



We introduce the *1-buffer* semantics of communicating machines, recall that $\vec{w}$ is the vector of buffers. Intuitively, this semantics forbids a machine to send a message to one of its partners if there is a non empty channel in the system.

Definition 12 (1-buffer, deadlock,convergent) *A configuration $(\vec{q};\vec{w})$ of a CS $S = (M_{\mathtt{p}})_{\mathtt{p}\in\mathscr{P}}$ is* stable *if and only if $\vec{w} = \vec{\varepsilon}$, while is* final *if it is stable and $\vec{q} \in (F_{\mathtt{p}})_{\mathtt{p}\in\mathscr{P}}$. The* 1-buffer semantics of *S is given by the relation*

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

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

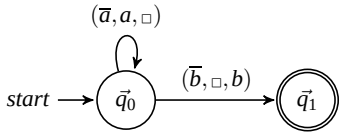
$$RS_{\leq 1}(S) \stackrel{def}{=} \{(\vec{q};\vec{w}) \in RS(S) \mid (\vec{q};\vec{w}) \text{ is stable or } \exists \mathtt{pq} \in C : \exists a \in \mathbb{R} : \vec{w}_{(\mathtt{pq})} = a \wedge \forall \mathtt{p'q'} \neq \mathtt{pq}.\vec{w}_{(\mathtt{p'q'})} = \varepsilon\}$$

We say that the system S is convergent *if and only if for every* reachable *configuration $(\vec{q};\vec{w})$ it is possible to reach a final configuration in the 1-buffer semantics.*

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

Note that if a system is *convergent* than it is *deadlock-free*. The 1-buffer semantics above is instrumental to the relation we establish between strong agreement of contract automata and convergence of CFMSs.

Remark 3 *Note that by considering only finite traces, we rule out all the unfair traces. For example, consider the following* strongly safe *CA:*



If the first and second participants could execute the transition $(\vec{q}_0, (\overline{a}, a, \square), \vec{q}_0)$ infinitely often then the third participant would be prevented from reading the message b. This behaviour is ruled out by considering only finite traces. Indeed all the possible traces generated by the automaton above are described by the regular expression $(\overline{a}, a, \square)^*(\overline{b}, \square, b)$, where the third participant will eventually reach its goal.

We define $snd(\vec{a}) \stackrel{\text{def}}{=} \mathtt{i}$ when $\vec{a}$ is a match action or an offer action such that $\vec{a}_{(i)} \in \mathbb{O}$ and, similarly, $rcv(\vec{a}) \stackrel{\text{def}}{=} \mathtt{j}$ when $\vec{a}$ is a request or a match and j is such that $\vec{a}_{(j)} \in \mathbb{R}$.

Property 3 *Let $S(KS_{\mathscr{A}})$ be a CS obtained by Definition 11, and s_0 be its initial configuration. Then for all f such that $s_0 \stackrel{f}{\twoheadrightarrow}$ there is a strong agreement φ such that $f = [\![\varphi]\!]$ or $f = [\![\varphi]\!]\overline{a}@\mathtt{ij}$ for some $a, \mathtt{i}, \mathtt{j}$.*

Proof. The proof follows trivially by observing that $KS_{\mathscr{A}}$ contains only match transitions and that the 1-buffer semantics does not allow other behaviours for the CS. ◇

Before providing the main results, we introduce a notion of well-formedness of contract automata. We require that an output action of a participant in a particular state is independent from the states of the other participants in the system.

Definition 13 (Branching Condition) *A contract automaton $\mathscr{A}$ has the* branching condition *iff for each $\vec{q}_1, \vec{q}_2$ reachable in $\mathscr{A}$ the following holds*

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

Example 8 *Consider the CAs of Example 7. The product automaton has the branching condition while this is not true for the MPC. Indeed, we have* $\vec{q}_{0(1)} = \vec{q}_{3(1)} = q_{01}$ *and* $\vec{q}_3 \xrightarrow{(\overline{a},a,\square)} \rightarrow$ *while there is no* $\vec{q}_i$ *such that* $(\vec{q}_1,(\overline{a},a,\square),\vec{q}_i) \in T_{KS}$.

The next theorem characterizes the relations between a CA and the corresponding CS. It states that the CS is capable of performing all the moves of the controller of the CA, while the CA is capable of performing those traces of the CS that are in *strong agreement*. Moreover, if a CA has the branching condition, the runs of the CS leading to a deadlock configuration correspond in CA to runs traversing liable transitions, and likewise when the CS reaches a non-deadlock configuration which does not reach a final configuration.

Theorem 1 *Let* $S(KS_{\mathscr{A}})$ *be the communicating system obtained by the MPC* $KS_{\mathscr{A}}$ *with* s_0, s_1, *and* s_2 *be the initial configurations of* $\mathscr{A}$, $KS_{\mathscr{A}}$, *and* $S(KS_{\mathscr{A}})$, *respectively. The following hold:*

1. *if* $s_1 \xrightarrow{\varphi}$ *then* $s_2 \overset{[\![\varphi]\!]}{\twoheadrightarrow}$
2. *if* $s_2 \overset{[\![\varphi]\!]}{\twoheadrightarrow}$ *and* φ *is a strong agreement then* $s_0 \xrightarrow{\varphi}$
3. *if* $s_2 \overset{f}{\twoheadrightarrow}$ *reaches a deadlock configuration where* $f = [\![\varphi]\!]$ *or* $f = [\![\varphi]\!]\overline{a}@\mathtt{ij}$ *and the branching condition holds in* $\mathscr{A}$ *then* $s_0 \xrightarrow{\hat{\varphi}}$ *has traversed a transition in* $TLiable(KS_{\mathscr{A}}/\mathscr{A})$ *where* $\hat{\varphi}$ *can be respectively* $\hat{\varphi} = \varphi$ *or* $\hat{\varphi} = \varphi\vec{a}$ *where* $\vec{a}$ *is a match on* a *with* $snd(\vec{a}) = \mathtt{i}, rcv(\vec{a}) = \mathtt{j}$.
4. *if* $s_2 \overset{[\![\varphi]\!]}{\twoheadrightarrow} s_2'$, s_2' *is not a deadlock configuration and no final configurations are reachable from* s_2' *then* $s_0 \xrightarrow{\varphi}$ *has traversed a transition in* $TLiable(KS_{\mathscr{A}}/\mathscr{A})$.

Proof. Through the proof assume that s_0', s_1' and s_2' are such that $s_0 \xrightarrow{\varphi} s_0'$, $s_1 \xrightarrow{\varphi} s_1'$ and $s_2 \overset{[\![\varphi]\!]}{\twoheadrightarrow} s_2'$.

1. By induction on φ. Assume that $s_1 \xrightarrow{\vec{a}} s_1'$ with $\vec{a}$ match action on a where principal i makes the offer and principal j makes the request on a. Let $\vec{q}_{0(i)}$ and $\vec{q}_{0(j)}$ be the initial states of participants `i` and `j` in $S(KS_{\mathscr{A}})$. By Definition 11, we have for some $\vec{q}_1$ and $\vec{q}_2$ that

$$(\vec{q}_{0(i)}, \overline{a}@\mathtt{ij}, \vec{q}_{1(i)}) \qquad \text{and} \qquad (\vec{q}_{0(j)}, a@\mathtt{ij}, \vec{q}_{2(i)})$$

   are transitions of participants `i` and `j`, respectively. We have $s_2 \overset{\overline{a}@\mathtt{ij}}{\twoheadrightarrow} \overset{a@\mathtt{ij}}{\twoheadrightarrow} s_2'$ since after the first transition participant `j` remains in its initial state.

   When $|\varphi| > 1$, we have for a configuration s_1'' that $s_1 \xrightarrow{\varphi} s_1'' \xrightarrow{\vec{a}_1} s_1'$. Hence $s_2 \overset{[\![\varphi]\!]}{\twoheadrightarrow} s_2''$ (by the induction hypothesis) and, since $\vec{a}_1$ is a match, with the same reasoning we can conclude $s_2'' \overset{[\![\vec{a}_1]\!]}{\twoheadrightarrow} s_2'$.

2. The proof is again by induction. Assume $s_2 \overset{[\![\vec{a}]\!]}{\twoheadrightarrow} s_2'$, where $\vec{a}$ is a match on a which involves (the principals i and j corresponding to) participants `i` and `j`. As before let i perform the offer and j the request. By Definition 3 (of product), we have that there is a transition $(\vec{q}_0, \vec{a}, \vec{q})$ in $\mathscr{A}$ from its initial state.

   When $|\varphi| > 1$, we have $s_2 \overset{[\![\varphi]\!]}{\twoheadrightarrow} s_2'' \overset{[\![\vec{a}]\!]}{\twoheadrightarrow} s_2'$ and there is $w' \in (\mathbb{L}^n)^*$ such that (by the induction hypothesis) $s_0 \xrightarrow{\varphi} s_0' = (\vec{q}', w')$ is a run in $\mathscr{A}$. Reasoning as in the base case, we conclude that $\mathscr{A}$ has a transition of the form $(\vec{q}', \vec{a}, \vec{q}'')$.
3. Let $s_2' = (\vec{q}; \vec{w})$ be the deadlock configuration reached from s_2 with f.

We distinguish two cases:

- if $\vec{w} \neq \vec{\varepsilon}$ (namely some buffer in $\vec{w}$ is not empty) then $\hat{\varphi}$ is not a strong agreement (in fact, if it were a strong agreement then the 1-buffer semantics of $S(KS_{\mathscr{A}})$ would yield $\vec{w} = \vec{\varepsilon}$). Then by Property 3 we have $f = [\![\varphi]\!]\overline{a}@\mathtt{ij}$ and $\hat{\varphi} = \varphi\vec{a}$. Moreover, Theorem 1.1 guarantees that a run $s_0 \xrightarrow{\varphi} s_0' = (\vec{q}', w')$ exists in $\mathscr{A}$ and, by Definition 11 and the branching condition, there is a transition $(\vec{q}', \vec{a}, \vec{q}'')$ in $\mathscr{A}$ where $\vec{a}$ is a match action on a. By contradiction, assume that there is no liable transition in the run $s_0 \xrightarrow{\varphi\vec{a}} (\vec{q}'', w'')$ in $\mathscr{A}$. Then, by construction (cf. Definition 8), the MPC of $\mathscr{A}$ has the same run, namely $s_1 \xrightarrow{\varphi\vec{a}} (\vec{q}'', w'')$. Finally, by Theorem 1.1, $s_2' \overset{a@\mathtt{ij}}{\twoheadrightarrow}$, contradicting the hypothesis that s_2' is a deadlock configuration.
- if $\vec{w} = \vec{\varepsilon}$ (namely, all buffers are empty) then, by definition of deadlock configuration of CFSMs (cf. Definition 12), the state $\vec{q}$ of configuration $s_2' = (\vec{q}; \vec{w})$ is not final, there is no participant ready to fire an output, and there is a participant waiting for an input on one of its buffers. The latter condition is guaranteed by the construction of CFSMs from controllers. Since $\vec{w} = \vec{\varepsilon}$, we have $f = [\![\varphi]\!]$ and $\hat{\varphi} = \varphi$ where $\varphi \in \mathfrak{Z}$ and there is a run $\mathtt{s}_0 \xrightarrow{\varphi} s_0' = (\vec{q}_1, w')$ in $\mathscr{A}$ (by Theorem 1.2). Note that $\vec{q}_1$ is redundant in $KS_{\mathscr{A}}$ (otherwise by Theorem 1.1, s_2' would not be a deadlock configuration) and, by construction (Lemma 2), $\vec{q}_1$ is removed from $KS_{\mathscr{A}}$. This implies that a liable transition has been traversed in $s_0 \xrightarrow{\varphi} s_0'$.

4. Wlog we can assume that φ is a strong agreement (otherwise we have $s_2 \overset{[\![\varphi]\!]\overline{a}@\mathtt{ij}}{\twoheadrightarrow} s_2'$ for some $\mathtt{i}, \mathtt{j}, \overline{a}$ and since s_2' is not a deadlock it is possible to perform the step $s_2' \overset{a@\mathtt{ij}}{\twoheadrightarrow} s_2''$ and we have that $\varphi\vec{a}$ is a strong agreement where $\vec{a}$ is a match action with $\vec{a}_{(i)} = \overline{a}, \vec{a}_{(j)} = a$). Moreover, from s_2'' is not possible to reach a final state, and we apply the following reasoning to $s_2'', \varphi\vec{a}$ instead of s_2', φ.

   Assume by contradiction that $s_0 \xrightarrow{\varphi} s_0'$ has traversed no liable transitions. Then by Definition 10 there exists φ' such that $s_0' \xrightarrow{\varphi'} s_0''$ and s_0'' is a final configuration. By Lemma 2 we must have $s_1 \xrightarrow{\varphi\varphi'} s_1'$ where s_1' is a final configuration. Hence by applying Theorem 1.1 we have $s_2' \overset{[\![\varphi']\!]}{\twoheadrightarrow} s_2'''$ where s_2''' is a final configuration, obtaining a contradiction. ◇

Note that the converse of Theorem 1.3 does not hold. Indeed if a CA $\mathscr{A}$ passes through a liable transition, it can be that $S(KS_{\mathscr{A}})$ never reaches a deadlock configuration.

Example 9 *Consider the CAs and CMs of Example 7. A possible trace belonging to the system $S(KS_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}})$ is generated by the transitions: $(q_{01}, \overline{a}@\mathtt{AB}, q_{11}), (q_{01}, a@\mathtt{AB}, q_{21})$. By using Theorem 1.2 this trace corresponds to the liable transition $(\vec{q}_0, (\overline{a}, a, \square), \vec{q}_7)$ of the product automaton.*

However after this two steps the system $S(KS_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}})$ will never reach a deadlock configuration. Indeed, it is always possible to perform the transitions:$(q_{11}, \overline{a}@\mathtt{AB}, q_{11}), (q_{21}, a@\mathtt{AB}, q_{21})$. Note that $S(KS_{\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}})$ is deadlock-free but not convergent.

We are now ready to state our main result: the controller of a CA has the branching condition if and only if the corresponding CS is convergent.

Theorem 2 *Let $\mathscr{A}$ be a contract automaton, $KS_{\mathscr{A}}$ be its MPC, and $S(KS_{\mathscr{A}})$ be the communicating system obtained by $KS_{\mathscr{A}}$. The following statements are equivalent :*

1 $S(KS_{\mathscr{A}})$ is convergent

2 $KS_{\mathscr{A}}$ has the branching condition

Proof. Let s_0, s_1, and s_2 be the initial configurations of $\mathscr{A}$, $KS_{\mathscr{A}}$, and $S(KS_{\mathscr{A}})$, respectively.

($\leftarrow$) Assume by contradiction that $S(KS_{\mathscr{A}})$ is not convergent and $KS_{\mathscr{A}}$ has the branching condition holds, namely there exists $s_2' = (\vec{q},\vec{w})$ such that $s_2 \overset{\varphi}{\twoheadrightarrow} s_2'$ and no final configurations are reachable from $(\vec{q},\vec{w})$. We distinguish two cases:

- s_2' is not a deadlock configuration. Then by applying Theorem 1.4 we have $\varphi = \varphi_1 \vec{a} \varphi''$ for some $\varphi_1, \vec{a}, \varphi''$ such that $\vec{q_0} \xrightarrow{\varphi_1} \vec{q_1}$ is a run of $\mathscr{A}$ and $(\vec{q_1}, \vec{a}, \vec{q_1'}) \in TLiable(KS_{\mathscr{A}}/\mathscr{A})$. Note that $\vec{a}$ is a match action, otherwise a participant in $S(KS_{\mathscr{A}})$ has fired an action to the environment in φ. Since $S(KS_{\mathscr{A}})$ is derived from $KS_{\mathscr{A}}$ and all the transitions of $KS_{\mathscr{A}}$ are match, this is not possible. Assume $\vec{a}$ is an action on a with $snd(\vec{a}) = \mathtt{i}$, $rcv(\vec{a}) = \mathtt{j}$ for some $\mathtt{i},\mathtt{j} \in \mathscr{P}$.

  By hypothesis we know that $s_2 \overset{[\![\varphi_1]\!]}{\twoheadrightarrow} s_2' \overset{\overline{a}@\mathtt{ij}}{\twoheadrightarrow}$, hence in the configuration s_2' the participant $\mathtt{i}$ is able to fire the action $\overline{a}@\mathtt{ij}$. By Definition 11, 3 there must be a state $\vec{q_2}$ in $KS_{\mathscr{A}}$ such that $(\vec{q_2}, \vec{a}, \vec{q_3})$ is a transition in $KS_{\mathscr{A}}$ (recall that $S(KS_{\mathscr{A}})$ is derived from $KS_{\mathscr{A}}$) and $\vec{q_2}_{(i)} = \vec{q_1}_{(i)}$ (otherwise we would not have $s_2' \overset{\overline{a}@\mathtt{ij}}{\twoheadrightarrow}$), and since $(\vec{q_1}, \vec{a}, \vec{q_1'}) \in TLiable(KS_{\mathscr{A}}/\mathscr{A})$ we conclude that the branching condition does not hold in $KS_{\mathscr{A}}$, obtaining a contradiction.

- all the possible configurations s_2' are deadlock. Then it must be that $s_2 \overset{[\![\varphi]\!]}{\twoheadrightarrow} s_2'' \overset{\overline{a}@\mathtt{ij}}{\twoheadrightarrow} s_2'$ for some $\mathtt{i},\mathtt{j},\overline{a}$, and from s_2'' it is possible to reach a final configuration, that is $s_2'' \overset{[\![\varphi']\!]}{\twoheadrightarrow} s_2'''$ where s_2''' is final. Note that it is not possible to have $s_2'' \overset{a@\mathtt{ij}}{\twoheadrightarrow} s_2'$ otherwise we would have that from s_2'' , which is not a deadlock, is not possible to reach a final configuration, or that s_2' is not a deadlock.

  By Theorem 1. 2 we have $s_0 \xrightarrow{\varphi\varphi'} s_0'$ where s_0' is final, hence by Lemma 2 it must be $s_1 \xrightarrow{\varphi} s_1' = (\vec{q_1}, w)$. As the previous case, by Definition 11, 3 there must be a state $\vec{q_2}$ in $KS_{\mathscr{A}}$ such that $(\vec{q_2}, \vec{a}, \vec{q_3})$ is a transition in $KS_{\mathscr{A}}$ where $\vec{a}$ is a match an action on a with $snd(\vec{a}) = \mathtt{i}$, $rcv(\vec{a}) = \mathtt{j}$ and $\vec{q_2}_{(i)} = \vec{q_1}_{(i)}$. Moreover since s_2' is a deadlock, it must be that there is no transition $(\vec{q_1}, \vec{a}, \vec{q_4})$ in $KS_{\mathscr{A}}$, otherwise by Theorem 1 . 1 we have $s_2' \overset{a@\mathtt{ij}}{\twoheadrightarrow}$, obtaining a contradiction. Hence we have that the branching condition does not hold in $KS_{\mathscr{A}}$, since there is no transition $(\vec{q_1}, \vec{a}, \vec{q_4})$ in $KS_{\mathscr{A}}$.

($\rightarrow$) By contradiction assume that the branching condition does not hold in $KS_{\mathscr{A}}$. Hence we have two states $\vec{q_1}, \vec{q_2}$ in $KS_{\mathscr{A}}$ such that $\vec{q_0} \xrightarrow{\varphi} \vec{q_1} \xrightarrow{\vec{a}}, \vec{q_0} \xrightarrow{\varphi'} \vec{q_2} \overset{\vec{a}}{\not\rightarrow}$ where $\vec{a}$ is a match on a with $snd(\vec{a}) = \mathtt{i}$, $rcv(\vec{a}) = \mathtt{j}$ for some $\mathtt{i},\mathtt{j} \in \mathscr{P}$ and $\vec{q_1}_{(i)} = \vec{q_2}_{(i)}$.

By Theorem 1.1 we have $s_2 \xrightarrow{[\![\varphi]\!]} s_2' \xrightarrow{\overline{a}@\mathtt{ij}}$ and $s_2 \xrightarrow{[\![\varphi']\!]} s_2''$. By Definition 11 and 3 we know that the participant $\mathtt{i}$ is in the same state in the configuration s_2', s_2'', hence we have $s_2'' \xrightarrow{\overline{a}@\mathtt{ij}} s_2'''$ and from s_2''' is is not possible to reach a final configuration. Otherwise if $s_2''' \xrightarrow{a@\mathtt{ij}[\![\varphi_2]\!]} s_f$ where s_f is final, then by Theorem 1.2 we would have $\vec{q_2} \xrightarrow{\vec{a}\varphi_2} \vec{q_f}$ where $\vec{q_f}$ is a final state of the CA, hence $\vec{q_2} \xrightarrow{\vec{a}}$ is not liable and belongs to $KS_{\mathscr{A}}$, obtaining a contradiction.

$\diamond$

A consequence of Theorem 2 is that a *strongly safe* CA has the branching condition if and only if the corresponding CS is convergent.

Corollary 4 *Let $\mathscr{A}$ be a contract automaton, then $\mathscr{A}$ is strongly safe and has branching condition if and only if $S(\mathscr{A})$ is convergent.*

Proof. The statement follows trivially by notice that if $\mathscr{A}$ is strongly safe then $\mathscr{A} = KS_{\mathscr{A}}$, hence $KS_{\mathscr{A}}$ has the branching condition and we can apply Theorem 2.

$\diamond$

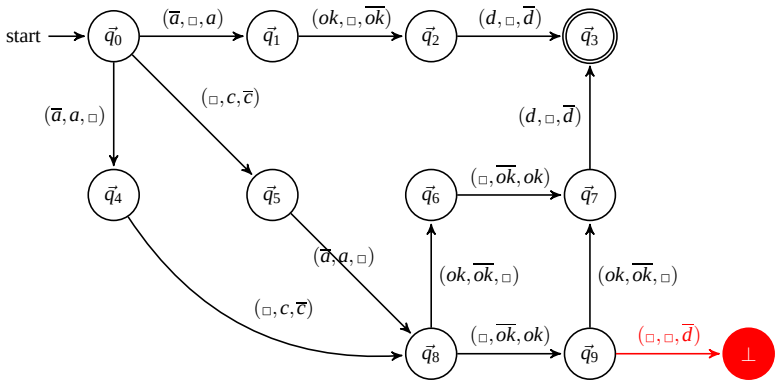
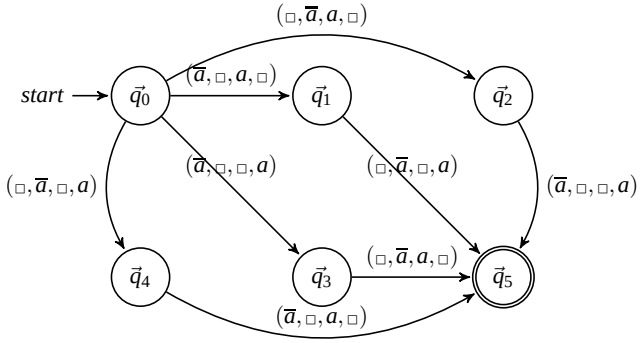


Figure 1: $KS_{\mathscr{A}}$

Example 10 *Consider the following* strongly safe *CA* $\mathscr{A} = \mathtt{A} \otimes \mathtt{B} \otimes \mathtt{C} \otimes \mathtt{D}$*:*



In this example we have four participants: the first two $(\mathtt{A},\mathtt{B})$ *perform the same offer* $\overline{a}$*, while the others* $(\mathtt{C},\mathtt{D})$ *perform the request* a*. The CA* $\mathscr{A}$ *has no branching condition: for example the internal state of the participant* $\mathtt{B}$ *is the same in both states* $\vec{q}_1,\vec{q}_3$*. From state* $\vec{q}_1$ *we have the match transition* $(\square,\overline{a},\square,a)$ *which is not available in state* $\vec{q}_3$*, and from state* $\vec{q}_3$ *we have the match transition* $(\square,\overline{a},a,\square)$ *which is not available from state* $\vec{q}_1$*.*

The translation yields the CMs:

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

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

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

Example 11 *Figure 1 depicts the automaton* $KS_{\mathscr{A}}$ *where* $\mathscr{A} = \mathtt{A} \otimes \mathtt{B} \otimes \mathtt{C}$*:*

$$\mathtt{A} = \overline{a}.ok.d \qquad \mathtt{B} = (a.c + c.a).\overline{ok}.\overline{ok} \qquad \mathtt{C} = a.\overline{ok}.\overline{d} + \overline{c}.ok.\overline{d}$$

Participant $\mathtt{A}$ *sends an offer* $\overline{a}$ *and then waits on acknowledgement* ok *and then a message* d*. Participant* $\mathtt{B}$ *acts as an intermediary: it receives the requests* a *and* c *and then replies with* $\overline{ok}$*. Finally, participant* $\mathtt{C}$ *can either behave similarly to* $\mathtt{A}$ *or directly acknowledge the message received on* a *(and then send* d*). The translation in Definition 11 yields the following communicating machines, written as regular expressions:*

$$[\![KS_{\mathscr{A}}]\!]_{\mathtt{A}} = \overline{a}@\mathtt{AC}.ok@\mathtt{CA}.d@\mathtt{CA} + \overline{a}@\mathtt{AB}.ok@\mathtt{BA}.d@\mathtt{CA}$$
$$[\![KS_{\mathscr{A}}]\!]_{\mathtt{B}} = (a@\mathtt{AB}.c@\mathtt{CB} + c@\mathtt{CB}.a@\mathtt{AB}).(\overline{ok}@\mathtt{BA}.\overline{ok}@\mathtt{BC} + \overline{ok}@\mathtt{BC}.\overline{ok}@\mathtt{BA})$$
$$[\![KS_{\mathscr{A}}]\!]_{\mathtt{C}} = \overline{c}@\mathtt{CB}.ok@\mathtt{BC}.\overline{d}@\mathtt{CA} + a@\mathtt{AC}.\overline{ok}@\mathtt{CA}.\overline{d}@\mathtt{CA}$$

Note that $KS_{\mathscr{A}}$ has no branching condition, indeed $\vec{q}_{9(3)} = \vec{q}_{7(3)}$ but there is no $(\vec{q}_9,(d,\square,\overline{d}),\vec{q}')$ in $KS_{\mathscr{A}}$ for some $\vec{q}'$. Moreover there is a liable *transition with label $(\vec{q}_9,(\square,\square,\overline{d}),\bot)$ in Figure 1 which represents the possible deadlock in the system.*

A deadlock configuration in $S(KS_{\mathscr{A}})$ is given by the trace $[\![\varphi]\!]\overline{d}@\mathtt{CA}$ where:

$$\varphi = (\overline{a},a,\square)(\square,c,\overline{c})(,\overline{ok},ok)$$

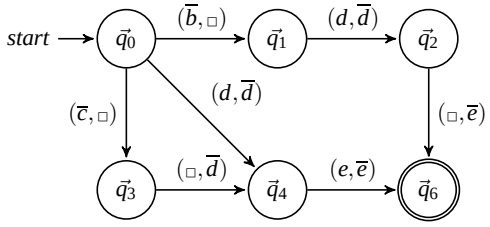
Indeed $s_0 \overset{[\![\varphi]\!]\overline{d}@\mathtt{CA}}{\twoheadrightarrow} s_0'$ where s_0 is the initial configuration of $S(KS_{\mathscr{A}})$. In the configuration $s_0' = (\vec{q},\vec{w})$ the buffer $\vec{w}$ is not empty, because $\vec{w}_{(CA)} = d$. Moreover the machine A is prevented to read the message on the buffer since its configuration in $\vec{q}$ is $ok@\mathtt{BA}.d@\mathtt{CA}$.

5 On extending the approach

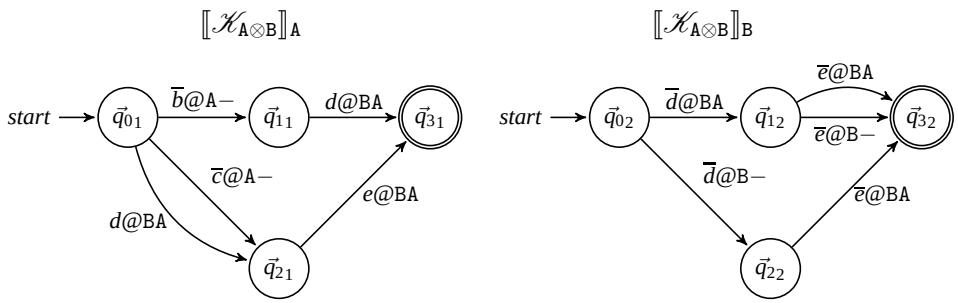
We discuss possible extensions of our approach to other existing types of agreement on CAs, and on different semantics of CMs, where there are no constraints on the number of messages in a buffer. We start by comparing the other existing types of agreement with the 1-buffer semantics for CMs.

On agreement The property of *agreement* requires that all the requests are matched. It allows strings made by match and offer actions only. In the following we discuss a correspondence similar to Theorem 2 for the property of *agreement*.

Example 12 *Consider the CAs corresponding to the regular expressions $\mathtt{A} = \overline{b}.d + \overline{c}.e + d.e$ and $\mathtt{B} = \overline{d}.\overline{e}$. The controller $\mathscr{K}_{A\otimes B}$ for the property of* agreement *is given by the CA:*



The translation in Definition 11 yields the CMs:



Under the 1-buffer semantics, the system $S(\mathscr{K}_{A\otimes B})$ always reaches a deadlock configuration since in every execution there are messages in the buffer with no receiver, corresponding to the offer actions in the controller.

If we assume that the unmatched offers are consumed instantaneously by an artificial participant representing the environment, we still have possible deadlock configurations in $S(\mathscr{K}_{A\otimes B})$, for example if the participants execute the sequence of transitions $(\vec{q}_{01}, \overline{b}@\mathtt{A}-, \vec{q}_{11})$, $(\vec{q}_{02}, \overline{d}@\mathtt{B}-, \vec{q}_{22})$, $(\vec{q}_{22}, \overline{e}@\mathtt{BA}, \vec{q}_{32})$.

Note that $\mathscr{K}_{A\otimes B}$ has no branching condition: in $\vec{q}_1, \vec{q}_3$ the participant $\mathtt{B}$ is the state $\vec{q}_{02}$, but from $\vec{q}_3$ there is no match transition on action d.

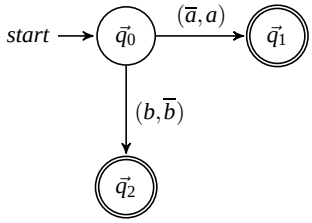
Under the assumptions that the offer actions are consumed by the environment, that it is possible to prove that the controller of the CA $\mathscr{A}$ has a slightly modified version of the branching condition if and only if the corresponding system $S(\mathscr{K}_{\mathscr{A}})$ is convergent. The proof is obtained by noticing that in the 1-buffer semantic a deadlock configuration is reached only if a participant $\mathtt{A}$ send a message a to a participant $\mathtt{B}$ and $\mathtt{B}$ is unable to consume the message. By Definition 11 this can happen only if there are two different states in the CA where $\mathtt{A}$ is in the same internal state and the match transition is available only in one of the two states, i.e. $\mathscr{K}_{\mathscr{A}}$ has no branching condition. We also need to consider those configurations which are not convergent nor deadlocks as done in Theorem 2.

On weak agreement For the property of *weak agreement* things are more intricate, indeed it is necessary to modify the actual translation. This is due to the possibility for a participant to fire a request if in the future the offer will be available, while in the CMs if the buffer is empty it is not possible to perform an input action.

To overcome this problem it is possible to synthesize one or more CMs which act as brokers. They receive as input all the actions of the participants, which are now translated into outputs, and reply with messages in a way to drive the participants through the trace in weak agreement.

On different semantics We now discuss the relations between CAs and other semantics for CMs.

Example 13 *Consider the following CA $\mathtt{A}\otimes\mathtt{B}$:*

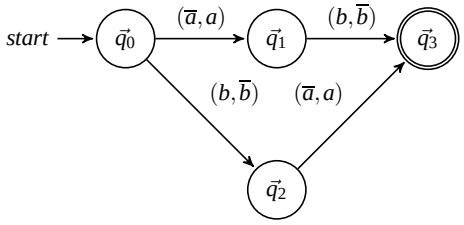


We have $\mathtt{A} = \overline{a} + b$, $\mathtt{B} = \overline{b} + a$. This CA is strongly safe *and has the branching condition. However, by considering the non 1-buffer semantics for CMs, the translated system is not* deadlock free. *Indeed a possible deadlock in $S(\mathscr{K}_{A\otimes B})$ is generated if the first participant performs the action $\overline{a}@\mathtt{AB}$ and then the second performs the action $\overline{b}@\mathtt{BA}$. This is because participant $\mathtt{B}$ can ignore the message received by the participant $\mathtt{A}$ and follow the other branch of the CA. These behaviours is not permitted by the 1-buffer semantics, which forces participants to follow the successful branch.*

The previous example shows that if we allow a less constrained semantics for CMs then Theorem 2 does not hold any more. Indeed, we need to introduce other constraints on the behaviour of the CAs to obtain a correspondence with convergent systems.

Note that in the previous example, from state $\vec{q}_0$ both participants contain a branch where they can execute an input or an output action. This is called a *mixed choice* state. It is possible to prove that if a CA has the branching condition, it is strongly safe and has no mixed choice states then the corresponding system is convergent with non 1-buffer semantics. However the converse does not hold. Indeed there exists systems with mixed choice states that enjoy convergence.

Example 14 *Consider the following CA* $\mathtt{A} \otimes \mathtt{B}$*:*



We have $\mathtt{A} = \overline{a}.b + b.\overline{a}$, $\mathtt{B} = \overline{b}.a + a.\overline{b}$*. This CA is* strongly safe*, has the branching condition, and contains a mixed choice state, i.e.* $\vec{q}_0$*. Nevertheless, the corresponding system is convergent.*

As showed by the previous example, for obtaining a correspondence similar to Theorem 2, we need to consider those *bad* mixed choices, where the participants behave differently in the different branches.

6 Concluding Remarks

We have established a formal correspondence between contract automata, an orchestration model, and communicating machines, a model of choreography. An interesting implication of our results is that contract automata can be seen as an alternative semantics of communicating machines. In fact, the product of communicating machines could be built as a contract automaton once match-actions are properly defined as tuples where output messages appear before the corresponding input ones. However, contract automata are more general in the sense that they would also admit matches where a request appears before its corresponding offer. Exploring those alternative semantics is of interest and it is scope for future work.

The dichotomy orchestration-choreography has been discussed in many papers (see e.g., [9]). The only formal results (we are aware of) that link a choreography to an orchestration framework is in [5]. A precise comparison with [5] is not straightforward as the models use a bisimulation-like relation to exhibit a conformance relation between choreographed and orchestrated computations. Here, we study the conditions to "force" orchestrated computations to well-behave (strong safety), and convergence in a choreography framework in terms of strong safety in the orchestration one.

A practical outcome of our result is that strong safe contract automata can execute without controller (if they are trusted). In fact, one can translate them into communicating machines that run without central control.

For the time being, our result only states that strong agreement corresponds to the 1-buffer semantics of communicating machines. In other words, the execution of the machines is basically synchronous. (We note that this has some advantages since communicating machines with 1-buffer semantics are more computationally tractable [7].) We conjecture that results similar to the one presented in this paper can be achieved for weaker notions of agreement (for example, the ones in [3]) when considering asynchronous behaviours of communicating machines. This is nonetheless left as future work.

References

[1] Massimo Bartoletti, Alceste Scalas, Emilio Tuosto & Roberto Zunino (2013): *Honesty by Typing*. In: *FMOODS/FORTE*, pp. 305–320. Available at http://dx.doi.org/10.1007/978-3-642-38592-6_21.

[2] Massimo Bartoletti, Emilio Tuosto & Roberto Zunino (2012): *On the Realizability of Contracts in Dishonest Systems*. In: *COORDINATION*, pp. 245–260. Available at http://dx.doi.org/10.1007/978-3-642-30829-1_17.

[3] Davide Basile, Pierpaolo Degano & Gian Luigi Ferrari (2014): *Automata for Service Contracts*. In: *TGC*. To appear in TGC 2014.

[4] Daniel Brand & Pitro Zafiropulo (1983): *On Communicating Finite-State Machines*. *JACM* 30(2), pp. 323–342. Available at http://doi.acm.org/10.1145/322374.322380.

[5] Nadia Busi, Roberto Gorrieri, Claudio Guidi, Roberto Lucchi & Gianluigi Zavattaro (2006): *Choreography and Orchestration Conformance for System Design*. In Paolo Ciancarini & Herbert Wiklicky, editors: *Coordination Models and Languages*, *Lecture Notes in Computer Science* 4038, Springer Berlin Heidelberg, pp. 63–81. Available at http://dx.doi.org/10.1007/11767954_5.

[6] Christos G. Cassandras & Stephane Lafortune (2006): *Introduction to Discrete Event Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[7] Gérard Cécé & Alain Finkel (2005): *Verification of programs with half-duplex communication*. *I&C* 202(2), pp. 166–190. Available at http://dx.doi.org/10.1016/j.ic.2005.05.006.

[8] Pierre-Malo Deniélou & Nobuko Yoshida (2012): *Multiparty Session Types Meet Communicating Automata*. In: *ESOP*, pp. 194–213. Available at http://dx.doi.org/10.1007/978-3-642-28869-2_10.

[9] Chris Peltz (2003): *Web services orchestration and choreography*. *Computer* 36(10), pp. 46–52. Available at http://doi.ieeecomputersociety.org/10.1109/MC.2003.1236471.