Automata for Analysing Service Contracts ⋆

Davide Basile, Pierpaolo Degano, and Gian-Luigi Ferrari

{basile,degano,giangi}@di.unipi.it
Dipartimento di Informatica, Università di Pisa, Italy

Abstract. A novel approach to the formal description of service contracts is presented in terms of automata. We focus on the basic property of guaranteeing that in the multi-party composition of principals each individual gets his requests satisfied, so that the overall composition reaches its goal. Depending on whether requests are satisfied synchronously or asynchronously, we construct an orchestrator that at static time either yields composed services enjoying the required properties or detects the individuals responsible for possible violations. To do that in the asynchronous case we resort to techniques from Operational Research.

1 Introduction

Service composition is the fundamental part of the service-oriented approach. Often, the computational capabilities offered by a coarse-grained service are implemented taking advantage of multiple, finer-grained and loosely-coupled services. APIs for service composition (orchestration and choreography) are an integral part of service-based systems. Different languages have been proposed for orchestrating Web Services, the most popular among which is BPEL [23]. However, several crucial issues naturally arise, e.g. the usage of resources, the guarantees on the results provided, the correctness of the whole business process in a variety of different operational environments.

Service orchestrations rely on the notion of *service contract* which specifies what a service is going to offer and what in turn it requires. In order to serve its client, the orchestrator defines the duties and responsibilities for each of the different services he calls. This *contract agreement* is obviously based on the contracts of the involved services, and ensures that all the duties are properly kept. The orchestrator can then proceed to organise the overall service composition and to propose the resulting contract to its own clients. This process is called *contract composition*. Contract composition assumes that none of the involved service violates its contract, i.e. that contract agreement is fulfilled, in which case so does the orchestration. Furthermore, a failing service contract can cause the orchestrator to breach the contract with its clients. Thus reaching the agreement among the services is the essential ingredient in designing orchestrations.

The main question addressed by our research is twofold. First, we aim at developing techniques capable of determining a correct contract composition by considering the individual service contracts and their overall satisfaction within an orchestration. Second, we propose a rigorous formal technique, suitable to be automated, to compose

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

contracts. Previous work have tackled similar issues. We only mention a few that use the λ-calculus [4, 5, 8], process calculi [14, 15, 1], non-classical logics [7].

Here, we introduce an automata-based model for contracts called *contract automata* (CA), that are a special kind of finite state automata endowed with two composition operators, involving many parties and reflecting two different orchestration policies. One operation joins a service to an existing orchestration with no reconfiguration; the second requires a global adaptive re-orchestration of all the services involved. We also define properties of CA that guarantee a group of contracts to agree under all circumstances. The first one is called *agreement* and considers the case when all the requests made are synchronously matched in pair by the offers, and thus satisfied; we also synthesise a suitable orchestrator to enforce contract composition to adhere to this behaviour. The second property, *weak agreement*, is more liberal, in that requests can be asynchronously matched, and an offer can be delivered even before a corresponding request.

We also establish conditions ensuring that a group of contracts agree in some specific cases (they *admit* agreement or weak agreement). We develop static techniques to detect which individual in a contract is *liable*, i.e. the responsibles for leading a contract composition into a failing state; as a matter of fact, our notion slightly differs form other versions of liability [6], mainly because we do not admit the possibility of redeem from culpability. While finding the individuals liable for a violation of agreement is not hard (inspecting the automaton plus a little calculation suffice), things become much more intricate when we consider weak agreement, that has a clear interpretation as a context-sensitive language. For the synchronous case, we offer a method for composing contract and detecting liability, that is inspired by so-called controllers of the Control Theory [12]. Instead, we check weak agreement and detect liability in an original manner by resorting to optimization techniques borrowed from Operational Research. In particular, we show that these notions are naturally handled in terms of optimization of network flows. The intuitive idea is that the problem of service composition is rendered as a flow itinerary in the networks, that is automatically constructed from the contract automata resulting from service composition.

Because of space limitation, all the proofs of our results are omitted and can be found in [9].

2 The Model

This section formally introduces the notion of contract automata. We start with some notation and preliminary definitions. Let $\Sigma = \mathbb{R} \cup \mathbb{O} \cup \{\square\}$ be the alphabet of *actions* partitioned in *requests* $\mathbb{R} = \{a, b, c, \ldots\}$ and *offers* $\mathbb{O} = \{\overline{a}, \overline{b}, \overline{c}, \ldots\}$ where $\mathbb{R} \cap \mathbb{O} = \emptyset$, and $\square \notin \mathbb{R} \cup \mathbb{O}$ is a distinguished element representing the *idle* action. We define the involution $co(\bullet) : \Sigma \mapsto \Sigma$ such that $co(\mathbb{R}) = \mathbb{O}$, $co(\mathbb{O}) = \mathbb{R}$, $co(\square) = \square$.

Let $\vec{v} = (v_1, ..., v_n)$ be a vector of *rank* $n = r_v \geq 1$, then $\vec{v}_i$ denotes the i-th element. We write $\vec{v}^1\vec{v}^2 \ldots \vec{v}^m$ for the concatenation of m vectors $\vec{v}^i$. The alphabet of a CA consists of vectors, each element of which records the activity of a single participant in the contract. In a vector there is either a single offer or a single request, or there is a pair of complementary request-offer that match; all the other elements of the vector contain

the symbol $\square$, meaning that the corresponding individuals stay idle. In the following let $\square^m$ denote a vector of rank m, all components of which are $\square$.

Definition 1 (Actions). *Given a vector $\vec{a} \in \Sigma^n$, if*

- $\vec{a} = \square^{n_1}\alpha\square^{n_2}, n_1, n_2 \geq 0$*, then $\vec{a}$ is a* request *if $\alpha \in \mathbb{R}$, and is an* offer *if $\alpha \in \mathbb{O}$*
- $\vec{a} = \square^{n_1}\alpha\square^{n_2}co(\alpha)\square^{n_3}, n_1, n_2, n_3 \geq 0$*, then $\vec{a}$ is a* match*, where $\alpha \in \mathbb{R} \cup \mathbb{O}$.*

We define $\vec{a} \bowtie \vec{b}$ iff $\vec{a}$ is an offer and $\vec{b}$ is a request or vice versa with $\vec{a}_i = co(\vec{b}_j) \neq \square$ and $r_a = r_b$, for two indexes $i, j \leq r_a$ (note that $\bowtie$ is an equivalence relation).

Definition 2 (Observable).
Let $w = \vec{a}_1 \ldots \vec{a}_n$ be a sequence of vectors, and let ε be the empty one, then its observable is given by the partial function $Obs(w) \in \mathbb{R} \cup \mathbb{O} \cup \{\tau\}$ where:

$$Obs(\varepsilon) = \varepsilon \qquad Obs(\vec{a}w') = \begin{cases} \vec{a}_i\, Obs(w') \text{ if } \forall j \neq i.\vec{a}_j = \square, (\vec{a} \text{ is a offer/request}) \\ \tau\, Obs(w') \quad (\vec{a} \text{ is a match}) \end{cases}$$

From now onwards we will only consider strings w such that $Obs(w)$ is defined.

Definition 3 (Contract Automata). *Assume as given a finite set of states $\mathfrak{Q} = \{q_1, q_2, \ldots\}$. Then a* contract automaton *(CA) $\mathcal{A}$ of rank n is a tuple $\langle Q, \vec{q_0}, A^r, A^o, T, F\rangle$, where*

- *$Q = Q_1 \times \ldots \times Q_n \subseteq \mathfrak{Q}^n$*
- *$\vec{q_0} \in Q$ is the initial state*
- *$A^r \subseteq \mathbb{R}, A^o \subseteq \mathbb{O}$ are finite sets of actions*
- *$F \subseteq Q$ is the set of final states*
- *$T \subseteq Q \times A \times Q$ is the set of transitions, where $A \subseteq (A^r \cup A^o \cup \{\square\})^n$ and if $(\vec{q}, \vec{a}, \vec{q'}) \in T$ then: $\begin{cases} \vec{a} \text{ is either a request or an offer or a match;} \\ \text{if } \vec{a}_i = \square \text{ then it must be } \vec{q}_i = \vec{q'}_i \end{cases}$*

A principal *is a contract automaton of rank 1 such that $A^r \cap co(A^o) = \emptyset$.*
A step $(w, \vec{q}) \to (w', \vec{q'})$ occurs iff $w = \vec{a}w', w' \in A^*$ and $(\vec{q}, \vec{a}, \vec{q'}) \in T$.
The language of $\mathcal{A}$ is $\mathscr{L}(\mathcal{A}) = \{w \mid (w, \vec{q_0}) \to^* (\varepsilon, \vec{q}), \vec{q} \in F\}$ where $\to^*$ is the reflexive, transitive closure of $\to$.

Example 1. Figure 1 shows three CAs. The automa $\mathcal{A}_1$ may be understood as producing a certain number of resources through one or more offers $\overline{res}$ and it terminates with the request *sig*. The contract $\mathcal{A}_2$ starts by sending the signal $\overline{sig}$ and then collects resources produced by $\mathcal{A}_1$. The contract $\mathcal{A}_3$ represents the CA where $\mathcal{A}_1$ and $\mathcal{A}_2$ interact. We have that $\mathcal{A}_1, \mathcal{A}_2$ are of rank 1 while $\mathcal{A}_3$ is of rank 2.

Contract automata can be composed, by making the cartesian product of their states and of the labels of the joined transitions, with the additional possibility of labels recording matching request-offer, as it happens for $\mathcal{A}_3$ in Figure 1.

We introduce two different operators for composing contract automata. Both products interleave all the transitions of their operands. We only force a synchronization to happen when two contract automata are ready on their respective request/offer action. These operators represent two different policies of orchestration. The first operator, $\otimes$,

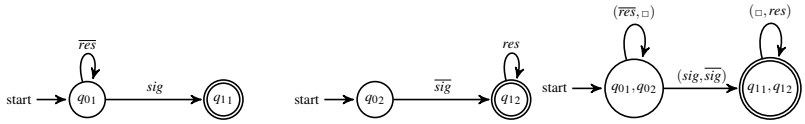

Fig. 1: Three contract automata: from left $\mathcal{A}_1, \mathcal{A}_2$, and $\mathcal{A}_3$

considers the case when a service S joins a group of services already clustered as a single orchestrated service S'. In this case S can only accept the still available offers (requests, respectively) of S' and vice versa. In other words, S cannot interact with the individual components of the orchestration S', but only with S' as a whole. This is not the case with the second operation of composition, $\boxtimes$, that puts instead all the components of S at the same level of those of S'. Any matching request-offer of either contract can be split, and the offers and requests, that become available again, can be re-combined with complementary actions. This second operation of composition turns out to satisfactorily model business processes in dynamically changing environments.

Definition 4 (Product). *Let $\mathcal{A}_i = \langle Q_i, \vec{q_0}^i, A_i^r, A_i^o, T_i, F_i \rangle, i \in 1 \dots n$ be CA of rank r_i. The product $\bigotimes_{i \in 1 \dots n} \mathcal{A}_i$ is the CA $\langle Q, \vec{q}_0, A^r, A^o, T, F \rangle$ of rank $m = \sum_{i \in 1 \dots n} r_i$, where:*

- $Q = Q_1 \times \dots \times Q_n$, *where* $\vec{q}_0 = \vec{q}_0^1 \dots \vec{q}_0^n$
- $A^r = \bigcup_{i \in 1 \cdots n} A_i^r$, $A^o = \bigcup_{i \in 1 \cdots n} A_i^o$
- $F = \{\vec{q}^1 \dots \vec{q}^n \mid \vec{q}^1 \dots \vec{q}^n \in Q, \vec{q}_i \in F_i, i \in 1 \dots n\}$
- *T is the least subset of $Q \times A \times Q$ s.t. $(\vec{q}, \vec{c}, \vec{q}') \in T$ iff, letting $\vec{q} = \vec{q}_1 \dots \vec{q}_n \in Q$,*
  - *either there are $1 \leq i < j \leq n$ s.t. $(\vec{q}_i, \vec{a}_i, \vec{q}_i') \in T_i$, $(\vec{q}_j, \vec{a}_j, \vec{q}_j') \in T_j$, $\vec{a}_i \bowtie \vec{a}_j$ and*
    $$\begin{cases} \vec{c} = \Box^u \vec{a}_i \Box^v \vec{a}_j \Box^z \text{ with } u = r_1 + \dots + r_{i-1},\ v = r_{i+1} + \dots + r_{j-1}, |\vec{c}| = m \\ \text{and} \\ \vec{q}' = \vec{q}_1 \dots \vec{q}_{i-1}\ \vec{q}_i'\ \vec{q}_{i+1} \dots\ \vec{q}_{j-1}\ \vec{q}_j'\ \vec{q}_{j+1} \dots \vec{q}_n \end{cases}$$
  - *or $\vec{c} = \Box^u \vec{a}_i \Box^v$ with $u = r_1 + \dots + r_{i-1}$, $v = r_{i+1} + \dots + r_n$, and $\vec{q}' = \vec{q}_1 \dots \vec{q}_{i-1}\ \vec{q}_i'\ \vec{q}_{i+1} \dots \vec{q}_n$ 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 $\vec{a}_i \bowtie \vec{a}_j$.*

There is an obvious way of retrieving the principals involved in a composition.

Definition 5 (Projection). *Let $\mathcal{A} = \langle Q, \vec{q}_0, A^r, A^o, T, F \rangle$ be a contract automaton of rank n, then $\prod^i(\mathcal{A}) = \langle \prod^i(Q), \vec{q}_{i_0}, \prod^i(A^r), \prod^i(A^o), \prod^i(T), \prod^i(F) \rangle$ where:*

$$\prod^i(Q) = \{\vec{q}_i \mid \vec{q} \in Q\} \quad \prod^i(F) = \{\vec{q}_i \mid \vec{q} \in F\} \quad \prod^i(A^r) = \{a \mid a \in A^r, (q, a, q') \in \prod^i(T)\}$$

$$\prod^i(T) = \{(\vec{q}_i, \vec{a}_i, \vec{q'}_i) \mid (\vec{q}, \vec{a}, \vec{q'}) \in T \wedge \vec{a}_i \neq \Box\} \quad \prod^i(A^o) = \{\overline{a} \mid \overline{a} \in A^o, (q, \overline{a}, q') \in \prod^i(T)\}$$

Consider the CAs of Figure 1, we have $\mathcal{A}_1 = \prod^1(\mathcal{A}_3)$ and $\mathcal{A}_2 = \prod^2(\mathcal{A}_3)$.

Property 1 (Product Decomposition). Let $\mathcal{A}_1, \dots, \mathcal{A}_n$ be a set of principal contract automata, then $\prod^i(\bigotimes_{j \in 1 \dots n} \mathcal{A}_j) = \mathcal{A}_i$.

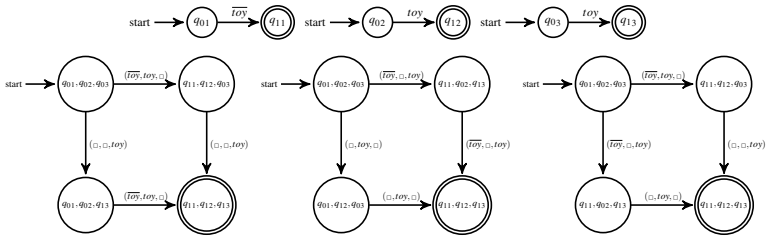

Fig. 2: From left to right and top-down: the contract automata of Alice, Bob and Eve, the contract automata $(Alice \otimes Bob) \otimes Eve$, $(Alice \otimes Eve) \otimes Bob$ and $Alice \boxtimes Bob \boxtimes Eve$.

Definition 6 (A-Product). *Let $\mathcal{A}_1, \mathcal{A}_2$ be two contract automata of rank n and m, and let $I = \{\prod^i(\mathcal{A}_1) \mid 0 < i \leq n\} \cup \{\prod^j(\mathcal{A}_2) \mid 0 < j \leq m\}$. The a-product of $\mathcal{A}_1$ and $\mathcal{A}_2$ is $\mathcal{A}_1 \boxtimes \mathcal{A}_2 = \bigotimes_{\mathcal{A}_i \in I} \mathcal{A}_i$*

From now on we assume that every contract automaton $\mathcal{A}$ of rank $r_{\mathcal{A}} > 1$ is composed by principal contract automata using the operations of product and a-product.

Note that if $\mathcal{A}, \mathcal{A}'$ are principal contract automata, then $\mathcal{A} \otimes \mathcal{A}' = \mathcal{A} \boxtimes \mathcal{A}'$. E.g. in Figure 1, we have that $\mathcal{A}_3 = \mathcal{A}_1 \otimes \mathcal{A}_2 = \mathcal{A}_1 \boxtimes \mathcal{A}_2$.

Both compositions are commutative, up to suitable rearrangement of the vectors of actions, and $\boxtimes$ is associative, while $\otimes$ is not.

Proposition 1. *The following properties hold:*
- $\exists \mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3.(\mathcal{A}_1 \otimes \mathcal{A}_2) \otimes \mathcal{A}_3 \neq \mathcal{A}_1 \otimes (\mathcal{A}_2 \otimes \mathcal{A}_3)$
- $\forall \mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3.(\mathcal{A}_1 \boxtimes \mathcal{A}_2) \boxtimes \mathcal{A}_3 = \mathcal{A}_1 \boxtimes (\mathcal{A}_2 \boxtimes \mathcal{A}_3)$

Example 2. In Figure 2 Alice offers a toy while Bob and Eve perform the same request for the toy of Alice. In the product $(Alice \otimes Bob) \otimes Eve$ the toy is assigned to Bob who first enters in the composition with Alice, no matter if Eve performs the same move. Equally, in the product $(Alice \otimes Eve) \otimes Bob$ the toy is assigned to Eve. In the last row we have the a-product $\mathcal{A}_1 \boxtimes (\mathcal{A}_2 \boxtimes \mathcal{A}_3) = (\mathcal{A}_1 \boxtimes \mathcal{A}_2) \boxtimes \mathcal{A}_3$ which represents a dynamic orchestration: no matter if Eve or Bob enters first the composition with Alice, the toy will be assigned to the first participant who makes the move.

3 Enforcing Agreement

An *agreement* between different contracts exists only when the final states of the product are reachable from the initial state by strings only made of match and offer actions. Our agreement resembles the notion of *compliance* of [14, 15]. Since we use vectors of actions as labels, it is easy to track every action performed by each participant, and to find who is liable in a bad interaction. Our goal is to enforce the behaviour of participants so that they only follow the traces of the automaton which lead to agreement.

We now introduce the notion of *agreement* as a property of the language recognised by a CA; an auxiliary definition helps.

Definition 7 (Agreement). *A word accepted by a CA is in* agreement *if it belongs to the set*

$$\mathfrak{A} = \{w \in (\Sigma^n)^* \mid Obs(w) \in (\mathbb{O} \cup \{\tau\})^*, n > 1\}$$

Example 3. The automa $\mathcal{A}_3$ in Figure 1 has a word in agreement: $Obs((\overline{res}, \Box)(sig, \overline{sig})) = \overline{res}\,\tau \in \mathfrak{A}$, and one not: $Obs((sig, \overline{sig})(\Box, res)) = \tau\, res \notin \mathfrak{A}$.

Note that the set $\mathfrak{A}$ can be seen as a safety property, where the set of bad prefixes of $\mathfrak{A}$ contains those strings ending with a trailing request, i.e. $\{w\vec{a} \mid w \in \mathfrak{A}, Obs(\vec{a}) \in \mathbb{R}\}$.

Definition 8 (Safety). *A contract automaton $\mathcal{A}$ is* safe *if $\mathscr{L}(\mathcal{A}) \subseteq \mathfrak{A}$, otherwise it is* unsafe. *Additionally, if $\mathscr{L}(\mathcal{A}) \cap \mathfrak{A} \neq \emptyset$ then $\mathcal{A}$* admits agreement.

Example 4. The contract automaton $\mathcal{A}_3$ of Figure 1 is unsafe, but it admits agreement since $\mathscr{L}(\mathcal{A}_3) \cap \mathfrak{A} = (\overline{res}, \Box)^*(sig, \overline{sig})$. The contract automaton $Alice \otimes Bob$ of Figure 2 is safe since $\mathscr{L}(Alice \otimes Bob) = (\overline{toy}, toy) \subset \mathfrak{A}$.

We now introduce a technique for driving a safe composition of contracts, in the style of the Supervisory Control for Discrete Event Systems [12]. These are automata, where accepting states represent successful termination of a task, while *forbidden states* are those that should not be traversed in "good" computations. Generally, the purpose is then to synthesize a controller that enforces this property. Here, we devise a way of synthesising an orchestration that leads to safely composed contracts. Also we assume that all the actions are controllable and observable, so the theory guarantees that a most permissive controller exists that never blocks a good computation. Furthermore we assume that a request leads to a forbidden state. Finally, the behaviours that we want to enforce upon $\mathcal{A}$ are exactly those words belonging to $\mathfrak{A} \cap \mathscr{L}(\mathcal{A})$. In order to effectively build such a controller, we introduce below the notion of hanged state, i.e. a state from which no final state can be reached.

Definition 9 (Controller). *Let $\mathcal{A}$ and $\mathcal{K}$ be contract automata, we call $\mathcal{K}$* controller *of $\mathcal{A}$ iff $\mathscr{L}(\mathcal{K}) \subseteq \mathfrak{A} \cap \mathscr{L}(\mathcal{A})$.*
$\mathcal{K}$ is a most permissive controller (mpc) *iff $\forall \mathcal{K}'$ controller of $\mathcal{A}$ it is $\mathscr{L}(\mathcal{K}') \subseteq \mathscr{L}(\mathcal{K})$.*

Proposition 2. *Let $\mathcal{K}$ be a mpc of the CA $\mathcal{A}$, then $\mathscr{L}(\mathcal{K}) = \mathfrak{A} \cap \mathscr{L}(\mathcal{A})$.*

Definition 10 (Hanged state). *Let $\mathcal{A} = \langle Q, \vec{q}_0, A^r, A^o, T, F\rangle$ be a CA, then $\vec{q} \in Q$ is* hanged*, and belongs to the set $Hanged(\mathcal{A})$, if given $\vec{q}_f \in F, \nexists w.(w, \vec{q}) \to^* (\varepsilon, \vec{q}_f)$.*

Definition 11 (Building the mpc).
Let $\mathcal{A} = \langle Q, \vec{q}_0, A^r, A^o, T, F\rangle$ be a CA, $\mathcal{K}_1 = \langle Q, \vec{q}_0, A^r, A^o, T \setminus (\{t \in T \mid t \text{ is a request transition }\}, F\rangle$ and define

$$\mathcal{K}_{\mathcal{A}} = \langle Q \setminus Hanged(\mathcal{K}_1), \vec{q}_0, A^r, A^o, T_{\mathcal{K}_1} \setminus \{(\vec{q}, a, \vec{q'}) \mid \{\vec{q}, \vec{q'}\} \cap Hanged(\mathcal{K}_1) \neq \emptyset\}, F\rangle$$

Proposition 3 (MPC). *The controller $\mathcal{K}_{\mathcal{A}}$ of Definition 11 is the mpc of the CA $\mathcal{A}$.*

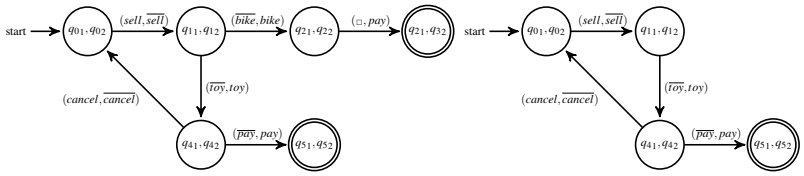

Fig. 3: In the left part the CA $\mathcal{A}$; in the right one its most permissive controller $K_{\mathcal{A}}$

Example 5. In Figure 3 we have a simple selling scenario involving two parties Alice and Bob, who is willing to pay the toy, only. We first compute the auxiliary set $\mathcal{K}_1$ that does not contain the transition $((q_{21},q_{22}),(\square,pay),(q_{21},q_{32}))$ because it represents a request of the second participant which is not fulfilled. As a consequence, some states become hanged: $Hanged(\mathcal{K}_1) = \{(q_{21},q_{22}),(q_{21},q_{32})\}$. By removing them, we eventually obtain the mpc controller of $\mathcal{A}$. Indeed this transition represents a request of the second participant which is not fulfilled. Once the request is removed, some hanged states are generated, in particular we have $Hanged(\mathcal{K}_1) = \{(q_{21},q_{22}),(q_{21},q_{32})\}$. By removing the hanged states and corresponding transition the mpc $K_{\mathcal{A}}$ of is obtained.

Property 2. Let $\mathcal{A}$ be a contract automaton and let $\mathcal{K}_{\mathcal{A}}$ be its mpc. If $\mathscr{L}(\mathcal{K}_{\mathcal{A}}) = \mathscr{L}(\mathcal{A})$ then $\mathcal{A}$ is safe, otherwise if $\mathscr{L}(\mathcal{K}_{\mathcal{A}}) \subset \mathscr{L}(\mathcal{A})$ then $\mathcal{A}$ is unsafe.
Additionally, if $\mathscr{L}(\mathcal{K}_{\mathcal{A}}) \neq \emptyset$, then $\mathcal{A}$ admits agreement.

We introduce now a novel notion of *liability*, that characterises those participants potentially responsible of the divergence from the expected behaviour. The liable participants are those who perform the first transition in a run, that is not possible in the most permissive controller. As noticed above, after this step is done, a successful state cannot be reached any longer, and so the participants who performed it will be blamed. (Note in passing that hanged states play a crucial role here: just removing the request transitions from $\mathcal{A}$ would result in a CA language equivalent to the mpc, but detecting liable participants would be much more intricate).

Definition 12 (Liability). *Let $\mathcal{A}$ be a CA and $\mathcal{K}_{\mathcal{A}}$ be the mpc of Definition 11; let $\rightarrow_{\mathcal{A}}$ and $\rightarrow_{\mathcal{K}_{\mathcal{A}}}$ be steps performed by $\mathcal{A}$ and $\mathcal{K}_{\mathcal{A}}$, resp.; let $(v\vec{a}w,\vec{q}_0) \rightarrow^*_{\mathcal{K}_{\mathcal{A}}} (\vec{a}w,\vec{q})$ be a run such that $(\vec{a}w,\vec{q}) \rightarrow_{\mathcal{A}} (w,\vec{q'})$ and $(\vec{a}w,\vec{q}) \not\rightarrow_{K_{\mathcal{A}}} (w,\vec{q'})$. The participants $\Pi^i(\mathcal{A})$ such that $\vec{a}_i \neq \square, i \in 1 \dots r_{\mathcal{A}}$ are* liable for *$v\vec{a}$ and belong to $Liable(\mathcal{A},v\vec{a})$.*

The set of liable *participants in $\mathcal{A}$ is $Liable(\mathcal{A}) = \{i \mid \exists w.i \in Liable(\mathcal{A},w)\}$.*

Example 6. In Figure 3 we have $Liable(\mathcal{A}) = \{1,2\}$, hence both Alice and Bob are possibly liable, because the match transition with label $(\overline{bike},bike)$ can be performed, that leads to a violation of the agreement.

Proposition 4. *A CA $\mathcal{A}$ is safe iff $Liable(\mathcal{A}) = \emptyset$.*

Note that the set $Liable(\mathcal{A})$ is easily computable as follows.

Lemma 1. *Let $\mathcal{A}$ be a CA and $\mathcal{K}_{\mathcal{A}}$ be its mpc as in Definition 11, then*

$$Liable(\mathcal{A}) = \{i \mid (\vec{q},\vec{a},\vec{q'}) \in T_{\mathcal{A}}, \vec{a}_i \neq \square, \vec{q} \in Q_{K_{\mathcal{A}}}, \vec{q'} \notin Q_{\mathcal{K}_{\mathcal{A}}}\}$$

Some properties of $\otimes$ and $\boxtimes$ follow, that enable us to predict under which conditions a composition is safe without actually computing it.

We first introduce the notions of collaborative and competitive contracts. Intuitively, two contracts are *collaborative* if some requests of one meet the offers of the other, and are *competitive* if both can satisfy the same request.

Definition 13 (Competitive, Collaborative).
The pair of CA $\mathcal{A}_1 = \langle Q_1, q_{01}, A_1^r, A_1^o, T_1, F_1 \rangle$ *and* $\mathcal{A}_2 = \langle Q_2, q_{02}, A_2^r, A_2^o, T_2, F_2 \rangle$ *are*

- competitive *if* $A_1^o \cap A_2^o \cap co(A_1^r \cup A_2^r) \neq \emptyset$
- collaborative *if* $(A_1^o \cap co(A_2^r)) \cup (co(A_1^r) \cap A_2^o) \neq \emptyset$.

The offers shared by $\mathcal{A}_1$ and $\mathcal{A}_2$ are $A_1^o \cap A_2^o$, while all their requests are $A_1^r \cup A_2^r$. If there is an offer provided by both participants that is matched by a request of one of the two CAs, that is $A_1^o \cap A_2^o \cap co(A_1^r \cup A_2^r) \neq \emptyset$, then the CAs are *competitive*. Note that *competitive* and *collaborative* are not mutually exclusive, as formally proved below. Moreover if two contract automata are *non-competitive* then all their match actions are preserved in the composition, indeed we have $\mathcal{A}_1 \boxtimes \mathcal{A}_2 = \mathcal{A}_1 \otimes \mathcal{A}_2$.

Example 7. The CAs $Alice, Bob \otimes Eve$ in Figure 2 are collaborative and not competitive, indeed there is a match on the toy action but no participant interferes in this match. In Example 8 the pair $\mathcal{A}_1, \mathcal{A}_2$ is competitive since $\mathcal{A}_2$ interferes with $\mathcal{A}_1$ on the toy offer.

The next theorem says that the composition of safe non-competitive contracts prevents all participants from harmful interactions, unlike the case of safe competitive contracts. In other words, when $\mathcal{A}_1$ and $\mathcal{A}_2$ are safe, no participants will be found liable in $\mathcal{A}_1 \otimes \mathcal{A}_2$ (i.e. $Liable(\mathcal{A}_1 \otimes \mathcal{A}_2) = \emptyset$), and the same happens for $\mathcal{A}_1 \boxtimes \mathcal{A}_2$ if the two are also non-competitive (i.e. $Liable(\mathcal{A}_1 \boxtimes \mathcal{A}_2) = \emptyset$).

Theorem 1. *If two CA* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *are*

1. *competitive then they are collaborative,*
2. *safe then* $\mathcal{A}_1 \otimes \mathcal{A}_2$ *is safe,* $\mathcal{A}_1 \boxtimes \mathcal{A}_2$ *admits agreement,*
3. *non-collaborative, and one or both unsafe, then* $\mathcal{A}_1 \otimes \mathcal{A}_2, \mathcal{A}_1 \boxtimes \mathcal{A}_2$ *are unsafe,*
4. *safe and non-competitive, then* $\mathcal{A}_1 \boxtimes \mathcal{A}_2$ *is safe.*

Example 8. We feel free to present contract automata through a sort of extended regular expressions. So, let $\mathcal{A}_1 = \overline{toy} + bike \otimes toy + \overline{bike}$ and $\mathcal{A}_2 = \overline{toy}$. Both contracts are safe, but competitive, and $\mathcal{A}_1 \boxtimes \mathcal{A}_2$ is unsafe because $(\square, toy, \overline{toy})(bike, \square, \square) \in \mathscr{L}(\mathcal{A})$.

4 Weak Agreement

The literature considers many different types of agreement, including one where actions are taken on credit if in the future the obligations will be honoured. This last notion is introduced for accepting as good those compositions of contracts where all the requests are matched by offers, but an agreement is not reached due to lack of synchronous match actions. This is a common scenario in contract composition, where each participant requires its requests to be satisfied before providing the corresponding offers. This kind of agreement has been faced using a variety of formal techniques as Process Algebras, Petri Nets, non classical Logics, Event Structures [7, 3, 2, 6]. A simple example follows.

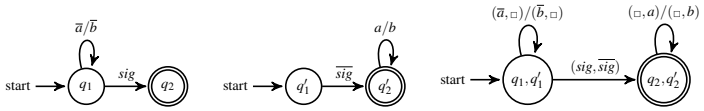

Fig. 4: From left to right: $\mathcal{A}_4, \mathcal{A}_5$, and $\mathcal{A}_4 \otimes \mathcal{A}_5$

Example 9. Suppose Alice and Bob want to share a bike and a toy, but neither trusts the other. Before providing their offer they first ask for the complementary request. As regular expressions: $Alice = toy.\overline{bike}$ and $Bob = bike.\overline{toy}$. Their composition is: $Alice \otimes Bob = (toy,\square)(\overline{bike},bike)(\square,\overline{toy}) + (\square,bike)(toy,\overline{toy})(\overline{bike},\square)$. In both possible runs the contracts fail on exchanging the bike or the toy, hence $\mathscr{L}(Alice \otimes Bob) \cap \mathfrak{A} = \emptyset$ and the composition does not admit agreement.

The circularity in the requests/offers is solved by weakening the notion of agreement, allowing a request to be performed on credit if in the future a complementary offer will occur. We will say that a contract automaton admits *weak agreement* if there exists a run in which every request can be paired with the corresponding offer, allowing asynchronous matches between requests and offers. Of course, agreement is a proper subset of weak agreement. As for agreement, we have an auxiliary definition.

Definition 14 (Weak Agreement). *A word accepted by a CA of rank $n > 1$ is in* weak agreement *if it belongs to $\mathfrak{W} = \{w \in (\Sigma^n)^* \mid |w| = m, \exists$ a function $f : [1..m] \to [1..m]$ total and injective on the requests of w, and such that $f(i) = j$ only if $\vec{a}^i \bowtie \vec{a}^j\}$.*

Example 10. Consider $\mathcal{A}_3$ in Figure 1, whose run $(\overline{res},\square)(sig,\overline{sig})(\square,res)$ is in $\mathfrak{W}$ but not in $\mathfrak{A}$, while $(\overline{res},\square)(sig,\overline{sig})(\square,res)(\square,res) \notin \mathfrak{W}$.

Definition 15 (Weak Safety). *A contract automaton $\mathcal{A}$ is* weakly safe *if $\mathscr{L}(\mathcal{A}) \subseteq \mathfrak{W}$, otherwise is weakly unsafe. If $\mathscr{L}(\mathcal{A}) \cap \mathfrak{W} \neq \emptyset$ then $\mathcal{A}$* admits weak agreement.

Example 11. In Example 9 we have $\mathscr{L}(Alice \otimes Bob) \subset \mathfrak{W}$, hence the composition of *Alice* and *Bob* is weakly safe.

The following is an analogous of Theorem 1.

Theorem 2. *Let $\mathcal{A}_1, \mathcal{A}_2$ be two CA, then if $\mathcal{A}_1, \mathcal{A}_2$ are*

1. *weakly safe then $\mathcal{A}_1 \otimes \mathcal{A}_2$ is weakly safe, $\mathcal{A}_1 \boxtimes \mathcal{A}_2$ admits weak agreement*
2. *non-collaborative and one or both unsafe, then $\mathcal{A}_1 \otimes \mathcal{A}_2, \mathcal{A}_1 \boxtimes \mathcal{A}_2$ are weakly unsafe*
3. *safe and non-competitive, then $\mathcal{A}_1 \boxtimes \mathcal{A}_2$ is weakly safe.*

The example below shows that weak agreement is not a context-free notion, in language theoretical sense; rather it is context-sensitive. Therefore, we cannot define a controller for weak agreement in terms of CA.

Example 12. Let $\mathcal{A}_4$, $\mathcal{A}_5$ and $\mathcal{A}_4 \otimes \mathcal{A}_5$ be the automata in Figure 4, then we have that $L = \mathfrak{W} \cap \mathscr{L}(\mathcal{A}_4 \otimes \mathcal{A}_5) \neq \emptyset$ is not context-free. Consider the following regular language $L' = \{(\overline{a},\square)^*(\overline{b},\square)^*(sig,\overline{sig})(\square,a)^*(\square,b)^*\}$. We have that $L \cap L'$ is not context-free (by pumping lemma), and since L' is regular L is not context-free.

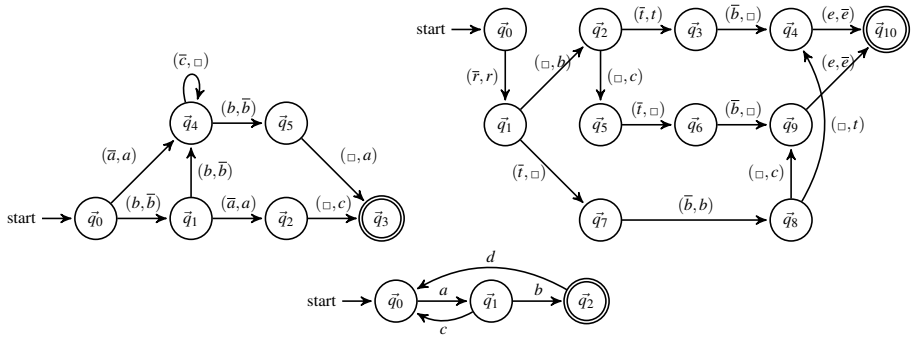

Fig. 5: Top left: a product of two CA; top right a booking service; bottom: a principal CA.

Theorem 3. *$\mathfrak{W}$ is a context-sensitive language, but not context-free.*

The language $\mathfrak{W}$ is *mildly* context-sensitive, and checking membership can be done through *two way deterministic pushdown automata* (2DPDA) in $O(n^2 log(n))$ time and $O(n)$ space [18]. A 2DPDA is a push-down automaton with a read-only input tape readable backward and forward. To check whether $w \in \mathfrak{W}$ we scroll the input tape twice for each action: the first time all the requests of the selected action are pushed and then, the second time we scroll the tape, when an offer of the same action is encountered we pop a request from the stack. At the end if the stack is empty the string w is in $\mathfrak{W}$.

In general, it is undecidable deciding whether a regular language L is included in a context-sensitive one, as well as checking emptiness of the intersection of a regular language with a context-sensitive one. In our case, however, these problems are decidable and so we can check whether a contract automaton $\mathcal{A}$ is weakly safe, or whether it admits weak agreement.

The technique we propose amounts to find optimal solutions of network flow problems [19]. We first fix some useful notation. Assume as given a CA $\mathcal{A}$, with a single final state $\vec{q}_f \neq \vec{q}_0$ (this condition can easily be met, by slightly manipulating a CA with many final states); assume the requests and the transitions of $\mathcal{A}$ be enumerated, i.e. $A^r = \{a^i \mid i \in I_l = \{1,2,\ldots,l\}\}$ and $T = \{t_1,\ldots,t_n\}$; let $FS(\vec{q}) = \{(\vec{q},\vec{a},\vec{q'}) \mid (\vec{q},\vec{a},\vec{q'}) \in T\}$ be the *forward star* of a node $\vec{q}$, and let $BS(\vec{q}) = \{(\vec{q'},\vec{a},\vec{q}) \mid (\vec{q'},\vec{a},\vec{q}) \in T\}$ be its *backward star*. For each transition t_i we introduce the flow variables $x_{t_i} \in \mathbb{N}$, and $z^{\vec{q}}_{t_i} \in \mathbb{R}$ where $\vec{q} \in Q, \vec{q} \neq \vec{q}_0$. Now we define the following set $F_{\vec{s},\vec{d}}$ of *flow constraints*, an element of which $\vec{x} = (x_{t_1},\ldots,x_{t_n}) \in F_{\vec{s},\vec{d}}$ defines runs from the source state $\vec{s}$ to the target state $\vec{d}$; as an abbreviation, we will write F_x for $F_{\vec{q}_0,\vec{q}_f}$. The intuition is that each variable x_{t_i} represents how many times the transition t_i is traversed in the runs defined by $\vec{x}$. A simple example follows; hereafter we identify a transition by its source and target states.

Example 13. Figure 5 (right) shows a simple service of booking. The contract of the client requires to book a room (r), including breakfast (b) and a transport service, by car (c) or taxi (t); finally it sends a signal of termination ($\overline{e}$); briefly $C = r.b.(c+t).\overline{e}$. The hotel offers a room, breakfast and taxi: $H = \overline{r}.\overline{t}.\overline{b}.e$. The composition $C \otimes H$ has four complete traces: $w_1 = (\overline{r},r)(\square,b)(\overline{t},t)(\overline{b},\square)(e,\overline{e})$, $w_2 = (\overline{r},r)(\square,b)(\square,c)(\overline{t},\square)(\overline{b},\square)(e,\overline{e})$,

$w_3 = (\overline{r},r)(\overline{t},\square)(\overline{b},b)(\square,t)(e,\overline{e})$, $w_4 = (\overline{r},r)(\overline{t},\square)\ (\overline{b},b)(\square,c)(e,\overline{e})$. We now detail the flows associated with each trace giving the set of variables with value 1, all the others having value 0, because there are no loops. For w_1: $\{x_{\vec{q}_0,\vec{q}_1}, x_{\vec{q}_1,\vec{q}_2}, x_{\vec{q}_2,\vec{q}_3}, x_{\vec{q}_3,\vec{q}_4}, x_{\vec{q}_4,\vec{q}_{10}},\}$; for w_2: $\{x_{\vec{q}_0,\vec{q}_1}, x_{\vec{q}_1,\vec{q}_2}, x_{\vec{q}_2,\vec{q}_5},\ x_{\vec{q}_5,\vec{q}_6}, x_{\vec{q}_6,\vec{q}_9}, x_{\vec{q}_9,\vec{q}_{10}}\}$; for w_3: $\{x_{\vec{q}_0,\vec{q}_1}, x_{\vec{q}_1,\vec{q}_7},\ x_{\vec{q}_7,\vec{q}_8}, x_{\vec{q}_8,\vec{q}_4},\ x_{\vec{q}_4,\vec{q}_{10}}\}$; and for w_4: $\{x_{\vec{q}_0,\vec{q}_1}, x_{\vec{q}_1,\vec{q}_7}, x_{\vec{q}_7,\vec{q}_8}, x_{\vec{q}_8,\vec{q}_9}, x_{\vec{q}_9,\vec{q}_{10}}\}$.

As a matter of fact, a flow $\vec{x}$ may represent many runs that have the same balance of requests/offers for each action occurring therein. As an example, consider the CA at the bottom of Figure 5 and the flow $x_{\vec{q}_0,\vec{q}_1} = 3, x_{\vec{q}_1,\vec{q}_2} = 2, x_{\vec{q}_2,\vec{q}_0} = x_{\vec{q}_1,\vec{q}_0} = 1$ that represents both $w_1 = acabdab$ and $w_2 = abdacab$.

The definition of $F_{\vec{s},\vec{d}}$ follows. The auxiliary variables $z^{\vec{q}}_{t_i}$ occurring there represent $|Q|-1$ auxiliary flows and make sure that $\vec{x}$ represent valid runs, i.e. there are no disconnected cycles with positive flow; a more detailed discussion on them is in Example 14 below. Note also that the values of $z^{\vec{q}}_{t_i}$ are *not* integers, and so we are defining Mixed Integer Linear Programming problems that have efficient solutions [19].

$$F_{\vec{s},\vec{d}} = \{(x_{t_1},\ldots,x_{t_n}) \mid \forall \vec{q} : (\sum_{t_i \in BS(\vec{q})} x_{t_i} - \sum_{t_i \in FS(\vec{q})} x_{t_i}) = \begin{cases} -1 \text{ if } \vec{q} = \vec{s} \\ 0 \quad \text{if } \vec{q} \neq \vec{s},\vec{d} \\ 1 \quad \text{if } \vec{q} = \vec{d} \end{cases}$$

$$\forall \vec{q} \neq \vec{s}, t_i. \quad 0 \leq z^{\vec{q}}_{t_i} \leq x_{t_i},$$

$$\forall \vec{q} \neq \vec{s}, \forall \vec{q'} : (\sum_{t_i \in BS(\vec{q'})} z^{\vec{q}}_{t_i} - \sum_{t_i \in FS(\vec{q'})} z^{\vec{q}}_{t_i}) = \begin{cases} -p^{\vec{q}} \text{ if } \vec{q'} = \vec{s} \\ 0 \quad \text{if } \vec{q'} \neq \vec{s},\vec{q} \\ p^{\vec{q}} \quad \text{if } \vec{q'} = \vec{q} \end{cases} \quad \text{where}$$

$$p^{\vec{q}} = \begin{cases} 1 \text{ if } \sum_{t_i \in FS(\vec{q})} x_{t_i} > 0 \\ 0 \text{ } otherwise \end{cases} \quad \}$$

We eventually define a set of variables $a^i_{t_j}$ for each action and each transition, that take the value -1 for requests, 1 for offers, and 0 otherwise; they help counting the difference between offers and requests of an action in a flow.

$$\forall t_j = (\vec{q},\vec{a},\vec{q'}) \in T, \forall i \in I_l : \quad a^i_{t_j} = \begin{cases} 1 \quad \text{if } Obs(\vec{a}) = \overline{a^i} \\ -1 \text{ if } Obs(\vec{a}) = a^i \\ 0 \quad otherwise \end{cases}$$

Example 14. Figure 5 (left) depicts the contract $A \otimes B$, where $A = \overline{a}.\overline{c}^*.b + b.(b.\overline{c}^*.b + \overline{a})$ and $B = a.\overline{b}.a + \overline{b}.(\overline{b}.\overline{b}.a + a.c)$. To check whether there exists a run recognizing a trace w with less or equal requests than the offers (for each action) we solve $\sum_{t_j} a^i_{t_j} x_{t_j} \geq 0$, for $\vec{x} \in F_x$.

We now discuss the role of the auxiliary variables $z^{\vec{q}}_{t_i}$. As said, they are used to ensure that the solutions considered represent valid runs. Consider the following assignment to $\vec{x}$: $x_{\vec{q}_0,\vec{q}_1} = x_{\vec{q}_1,\vec{q}_2} = x_{\vec{q}_2,\vec{q}_3} = 1, x_{\vec{q}_4,\vec{q}_4} \geq 1$, and null everywhere else. It does not represent valid runs, because the transition $(\vec{q}_4,(\overline{c},\square),\vec{q}_4)$ cannot be fired in a run that only takes transitions with non-null value in $\vec{x}$. Note that the constraints on $\vec{x}$ are

satisfied (e.g. we have $\sum_{t_j \in FS(\vec{q}_4)} x_{t_j} = \sum_{t_j \in BS(\vec{q}_4)} x_{t_j}$). Indeed, the constraints on the auxiliary $z^{\vec{q}}_{t_i}$ are introduced for checking if a node is reachable from the initial state on a run defined by the flow $\vec{x}$; note that their value being not integer is immaterial for checking that. The assignment above is not valid since for $z^{\vec{q}_4}$ we have $0 \leq z^{\vec{q}_4}_{(\vec{q}_0,\vec{q}_4)} \leq x_{(\vec{q}_0,\vec{q}_4)} = 0, 0 \leq z^{\vec{q}_4}_{(\vec{q}_1,\vec{q}_4)} \leq x_{(\vec{q}_1,\vec{q}_4)} = 0, 0 \leq z^{\vec{q}_4}_{(\vec{q}_4,\vec{q}_5)} \leq x_{(\vec{q}_4,\vec{q}_5)} = 0$, hence $\sum_{t_j \in BS(\vec{q}_4)} z^{\vec{q}_4}_{t_j} = z^{\vec{q}_4}_{(\vec{q}_4,\vec{q}_4)}, \sum_{t_j \in FS(\vec{q}_4)} z^{\vec{q}_4}_{t_j} = z^{\vec{q}_4}_{(\vec{q}_4,\vec{q}_4)}$ and we have $\sum_{t_j \in BS(\vec{q}_4)} z^{\vec{q}_4}_{t_j} - \sum_{t_j \in FS(\vec{q}_4)} z^{\vec{q}_4}_{t_j} = 0 \neq 1$.

Finally, note in passing that there are no valid flows $\vec{x} \in F_x$ for this problem.

Our main results follow.

Theorem 4. *Let $\vec{v}$ be a binary vector. Then a CA $\mathcal{A}$ is* weakly safe *iff $\min \gamma \geq 0$ where:*

$$\sum_{i \in I_l} v_i \sum_{t_j \in T} a^i_{t_j} x_{t_j} \leq \gamma \quad \sum_{i \in I_l} v_i = 1 \quad \forall i \in I_l . v_i \in \{0,1\} \quad (x_{t_1} \ldots x_{t_n}) \in F_x \quad \gamma \in \mathbb{R}$$

The minimum value of γ selects the trace and the action a for which the difference between the number of offers and requests is the minimal achievable from $\mathcal{A}$. If this difference is non negative, there will always be enough offers matching the requests, and so $\mathcal{A}$ will never generate a trace not in $\mathfrak{W}$: $\mathcal{A}$ is *weakly safe*, otherwise it is not.

Example 15. Consider again Example 13 and let $a^1 = r$, $a^2 = b$, $a^3 = t$, $a^4 = c$, $a^5 = e$. If $v_1 = 1$, for each flow $\vec{x} \in F_x$, we have that $\sum_{t_j} a^1_{t_j} x_{t_j} = 0$ (for $i \neq 1$, we have $v_i = 0$). This means that the request of a room is always satisfied. Similarly for breakfast and the termination signal e. If $v_3 = 1$, for the flow representing the traces w_1, w_3 we have $\sum_{t_j} a^3_{t_j} x_{t_j} = 0$, while for the flow representing the traces w_2, w_4 the result is 1. Also in this case the requests are satisfied. Instead, when $v_4 = 1$, for the flow representing the traces w_1, w_4 we have $\sum_{t_j} a^4_{t_j} x_{t_j} = 0$, but for the flow representing w_2, w_3, the result is -1. Hence $\min \gamma = -1$, and the CA $H \otimes C$ is not *weak safe*, indeed we have $w_2, w_3 \notin \mathfrak{W}$.

Theorem 5. *The CA $\mathcal{A}$ admits weak agreement iff :*

$$max\ \gamma \geq 0 \quad and \quad \forall i \in I_l . \sum_{t_j \in T} a^i_{t_j}\ x_{t_j} \geq \gamma \quad (x_{t_1} \ldots x_{t_n}) \in F_x \quad \gamma \in \mathbb{R}$$

The maximum value of γ in Theorem 5 selects the trace w that maximizes the least difference between offers and requests of an action in w. If this value is non negative, then there exists a trace w such that for all the actions in it, the offers are more or as many the requests. In this case, $\mathcal{A}$ admits weak agreement; otherwise it does not.

Example 16. In Example 13, $\max \gamma = -1$ for the flows representing the traces w_2, w_3 and $\max \gamma = 0$ for those of the traces w_1, w_4, that will be part of the solution and are indeed in weak agreement. Consequently, $H \otimes C$ admits *weak agreement*.

We define now the *weakly liable* participants: those who perform the first transition t of a run such that after t it is not possible any more to obtain a trace in $\mathfrak{W}$, i.e. leading to traces $w \in \mathscr{L}(\mathcal{A}) \setminus \mathfrak{W}$ that can not be extended to $ww' \in \mathscr{L}(\mathcal{A}) \cap \mathfrak{W}$.

Definition 16. *Let $\mathcal{A}$ be a CA and let $w = w_1\vec{a}w_2$ such that $w \in \mathscr{L}(\mathcal{A}) \setminus \mathfrak{W}, \forall w'.ww' \notin \mathscr{L}(\mathcal{A}) \cap \mathfrak{W}, \forall w_3.w_1\vec{a}w_3 \notin \mathscr{L}(\mathcal{A}) \cap \mathfrak{W}$ and $\exists w_4.w_1w_4 \in \mathscr{L}(\mathcal{A}) \cap \mathfrak{W}$. The participants $\Pi^i(\mathcal{A})$ such that $\vec{a}_i \neq \square$ are* weakly liable *and are denoted with $i \in WLiable(\mathcal{A}, w_1\vec{a})$.*

Let $WLiable(\mathcal{A}) = \{i \mid \exists w \text{ such that } i \in WLiable(\mathcal{A}, w)\}$ be the set of all potentially weakly liable *participants in $\mathcal{A}$.*

For computing the set $WLiable(\mathcal{A})$ we optimize a network flow problem for a transition $\bar{t}$ to check if there exists a trace w in which $\bar{t}$ reveals some weakly liable participants. By solving this problem for all transitions we obtain the set $WLiable(\mathcal{A})$.

Theorem 6. *The participant $\Pi^i(\mathcal{A})$ of a CA $\mathcal{A}$ is* weakly liable *if and only if there exists a transition $\bar{t} = (\vec{q}_s, \vec{a}, \vec{q}_t)$, $\vec{a}_i \neq \square$, and $\gamma_{\bar{t}} < 0$, where*

$$\gamma_{\bar{t}} = \mathit{min}\,\{f(\vec{x}) \mid \vec{x} \in F_{\vec{q}_0,\vec{q}_s},\ \vec{y} \in F_{\vec{q}_s,\vec{q}_f},\ \forall i \in I_l.\ \sum_{t_j \in T} a^i_{t_j}(x_{t_j} + y_{t_j}) \geq 0\}$$

$$f(\vec{x}) = \mathit{max}\,\{\gamma \mid \vec{u} \in F_{\vec{q}_t,\vec{q}_f},\ \forall i \in I_l.\ \sum_{t_j \in T} a^i_{t_j}(x_{t_j} + u_{t_j}) + a^i_{\bar{t}} \geq \gamma, \gamma \in \mathbb{R}\}$$

Intuitively, the flow defined above is split into three parts: the flow $\vec{x}$ goes from $\vec{q}_0$ to $\vec{q}_s$, the flow $\vec{y}$ goes from $\vec{q}_s$ to $\vec{q}_f$, and the flow $\vec{u}$ goes from $\vec{q}_t$ to $\vec{q}_f$.

The function f takes in input the flow $\vec{x}$ and selects a flow $\vec{u}$ such that by concatenating $\vec{x}$ and $\vec{u}$ through $\bar{t}$ we obtain a trace w, where the least difference between offers and requests is maximized for an action in w. Using the same argument of Theorem 5, if the value computed is negative, then there not exists a flow $\vec{u}$ that composed with $\vec{x}$ selects traces in weak agreement.

Finally $\gamma_{\bar{t}}$ yields the minimal result of $f(\vec{x})$, provided that there exists a flow $\vec{y}$, that combined with $\vec{x}$ represents only traces in weak agreement. If $\gamma_{\bar{t}} < 0$ then the transition $\bar{t}$ identifies some *weakly liable* participants. Indeed the flow $\vec{x}$ represents the traces w such that (1) $\exists w_1$, represented by $\vec{y}$, with $ww_1 \in \mathscr{L}(\mathcal{A}) \cap \mathfrak{W}$ and (2) $\forall w_2$, represented by $\vec{u}$, with $w\vec{a}w_2 \in \mathscr{L}(\mathcal{A}) \setminus \mathfrak{W}$. Note that if a flow $\vec{x}$ reveals some weakly liable participants, the minimization carried on by $\gamma_{\bar{t}}$ guarantees that the relevant transition $\bar{t}$ is found.

Example 17. In Figure 5 (right), the transitions $(\vec{q}_2, (\square, c), \vec{q}_5), (\vec{q}_8, (\square, c), \vec{q}_9)$ reveal the participant 2 *weakly liable*. Indeed from $\vec{q}_2$ the trace $(\bar{r}, r)(\square, b)$ can be extended to one in weak agreement, while $(\bar{r}, r)(\square, b)(\square, c)$ cannot. Also the trace $(\bar{r}, r)(\bar{t}, \square)(\bar{b}, b)$ can be extended to one in weak agreement while $(\bar{r}, r)(\bar{t}, \square)(\bar{b}, b)(\square, c)$ cannot.

For the transition $(\vec{q}_2, (\square, c), \vec{q}_5)$ we have the trace $(\bar{r}, r)(\square, b)$ for the flow $\vec{x}$ and $(\bar{t}, t)(\bar{b}, \square)(e, \bar{e})$ for the flow $\vec{y}$, and we have $\forall i \in I_l.\sum_{t \in T} a^i_t(x_t + y_t) \geq 0$. Note that if we select as flow $\vec{y}$ the trace $(\square, c)(\bar{t}, \square)(\bar{b}, \square)(e, \bar{e})$ then the constraints $\forall i \in I_l.\sum_{t_j \in T} a^i_{t_j}(x_{t_j} + y_{t_j}) \geq 0$ are not satisfied for the action $a^4 = c$. For the flow $\vec{u}$ the only possible trace is $(\bar{t}, \square)(\bar{b}, \square)(e, \bar{e})$, and $\mathsf{max}\ \gamma = -1 = \gamma_{(\vec{q}_2, (\square, c), \vec{q}_5)}$ since $\sum_{t_j \in T} a^4_{t_j}(x_{t_j} + u_{t_j}) + (-1) = -1$.

For the transition $(\vec{q}_8, (\square, c), \vec{q}_9)$ the flow $\vec{x}$ selects the trace $(\bar{r}, r)(\bar{t}, \square)(\bar{b}, b)$, the flow $\vec{y}$ selects the trace $(\square, t)(e, \bar{e})$, since the other possible trace, that is $(\square, c)(e, \bar{e})$, does not respect the constraints for the action a^4. Finally, for the flow $\vec{u}$ we have the trace $(e, \bar{e})$, and as the previous case $\mathsf{max}\ \gamma = -1 = \gamma_{(\vec{q}_8, (\square, c), \vec{q}_9)}$.

5 Conclusions and related work

We have introduced an original approach to contract composition for services. In our proposal contract composition is handled formally by taking advantage of the novel class of Contract Automata and of their compositional operators. The algebra of Contract Automata allows one to compose services according to two different notions of orchestrations: one when a participant joins an existing orchestration without a global reconfiguration, and the other when a global adaptive re-orchestration is required. We have defined notions that illustrate when a composition of contracts enjoys interesting properties, namely agreement and safety, both in the case when requests are satisfied by offers synchronously and asynchronously. Furthermore, a notion of liability has been put forward. A liable participant is the service leading the contract composition into a failing state. Key results of the paper prove the correctness of our approach by taking advantage of optimization techniques borrowed from Operational Research. Using them, we efficiently find the optimal solutions of the flow in the network automatically derived from contract automata. An interesting topic to investigate concerns whether and how our results and techniques can be used to define choreographies of services and their properties, in particular liability.

A main advantage of our framework is that it supports development of automatic verification tools for checking and verifying properties of contract composition. The formal treatment of contract composition in terms of optimal solutions of network flows paves the way of exploiting efficient optimization algorithms. We plan to develop such verification tools.

Related work The problem of formalizing contracts for service oriented computing, specifying and verifying the properties of a good composition received a lot of attention. In [14, 1] behavioural contracts are expressed via suitable process algebras, where the interactions between services are modelled via I/O actions. Two different choice operators, namely internal and external, describe how two services interact. The internal choice requires the other party to be able to synchronize with all the possible branches. This approach is extended to a multi party version by exploiting the π-calculus in [15]. The above papers focus on formalising the notion of progress of interactions. In our model, the internal choice is represented as a branching of requests. Also, we consider stronger properties than theirs: with agreement/weak agreement we require that all the requests of the contracts are satisfied, while for the property of progress it suffices that a subset of contracts meets their requests.

An extension of Intuitionistic Logic called Propositional Contract Logic (PCL) [7] has been proposed for modelling contracts with circular offers/requests. A new operator called *contractual implication* is introduced for dealing with actions taken on credit, if in the future the obligations will be honoured. Our notion of *weak agreement* is quite similar, and we also have a decision procedure to check if a contract automaton admits weak agreement.

Processes and contracts are two separate entities in [6], unlike ours. A process can fulfil its duty by obeying its contract or it behaves dishonestly and becomes *culpable* — and become honest again by performing later on the prescribed actions. We also do

not assume participants to be honest, and our notion of *liability* is slightly different. It is inspired by Control Theory [12] and expressed in language-theoretic terms.

Session types are studied, among others, in [13, 20] where *global types* represent a formal specification of a choreography of services in terms of their interactions. The projection of a safe global type to its components yields a safe *local type*, which is a term of a process algebra similar to [14]. From given safe local types, a choreography is synthesized, as a safe global type in [22]. In [16] local types are proved to correspond to communicating machines (CM) [11], that are finite state automata similar to ours. The main difference between the two is that CM interact through FIFO buffers, hence a participant can receive an input only if it was previously enqueued, while CA can offer/request on credit. However, under mild conditions, CA and CM can be proved equivalent [10], so establishing a first bridge between orchestration and choreography.

Acknowledgements We are indebted with Giancarlo Bigi and Emilio Tuosto for many enlightening discussions.

References

1. Acciai, L., Boreale, M., Zavattaro, G.: Behavioural contracts with request-response operations. Sci. Comput. Program. 78(2), 248–267 (2013)
2. Bartoletti, M., Cimoli, T., Pinna, G.M.: Lending Petri nets and contracts. In: Arbab, F., Sirjani, M. (eds.) FSEN. LNCS, vol. 8161, pp. 66–82. Springer (2013)
3. Bartoletti, M., Cimoli, T., Zunino, R.: A theory of agreements and protection. In: Basin, D.A., Mitchell, J.C. (eds.) POST. LNCS, vol. 7796, pp. 186–205. Springer (2013)
4. Bartoletti, M., Degano, P., Ferrari, G.L.: Planning and verifying service composition. Journal of Computer Security 17(5), 799–837 (2009)
5. Bartoletti, M., Degano, P., Ferrari, G.L., Zunino, R.: Call-by-contract for service discovery, orchestration and recovery. In: Wirsing, M., Hölzl, M.M. (eds.) Results of the SENSORIA Project, LNCS, vol. 6582, pp. 232–261. Springer (2011)
6. Bartoletti, M., Tuosto, E., Zunino, R.: Contract-oriented computing in co2. Sci. Ann. Comp. Sci. 22(1), 5–60 (2012)
7. Bartoletti, M., Zunino, R.: A logic for contracts. In: Cherubini, A., Coppo, M., Persiano, G. (eds.) ICTCS. pp. 34–37 (2009)
8. Basile, D., Degano, P., Ferrari, G.L.: Secure and unfailing services. In: Malyshkin, V. (ed.) PaCT. LNCS, vol. 7979, pp. 167–181. Springer (2013)
9. Basile, D., Degano, P., Ferrari, G.L., Tuosto, E.: Contract automata for services. Extended version: `http://www.di.unipi.it/~basile/TGC14.pdf`
10. Basile, D., Degano, P., Ferrari, G.L., Tuosto, E.: From orchestration to choreography through Contract Automata. In: ICE (2014), to appear
11. Brand, D., Zafiropulo, P.: On communicating finite-state machines. J. ACM 30(2), 323–342 (1983)
12. Cassandras, C.G., Lafortune, S.: Introduction to Discrete Event Systems. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2006)
13. Castagna, G., Dezani-Ciancaglini, M., Padovani, L.: On global types and multi-party session. Logical Methods in Computer Science 8(1) (2012)
14. Castagna, G., Gesbert, N., Padovani, L.: A theory of contracts for web services. ACM Trans. Program. Lang. Syst. 31(5) (2009)

15. Castagna, G., Padovani, L.: Contracts for mobile processes. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR. LNCS, vol. 5710, pp. 211–228. Springer (2009)
16. Deniélou, P.M., Yoshida, N.: Multiparty compatibility in communicating automata: Characterisation and synthesis of global session types. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M.Z., Peleg, D. (eds.) ICALP (2). LNCS, vol. 7966, pp. 174–186. Springer (2013)
17. Gischer, J.L.: Shuffle languages, Petri nets, and context-sensitive grammars. Commun. ACM 24(9), 597–605 (1981)
18. Gray, J., Harrison, M.A., Ibarra, O.H.: Two-way pushdown automata. Information and Control 11(1/2), 30–70 (1967)
19. Hemmecke, R., Koppe, M., Lee, J., Weismantel, R.: Nonlinear integer programming. In: Junger, M., Liebling, T.M., Naddef, D., Nemhauser, G.L., Pulleyblank, W.R., Reinelt, G., Rinaldi, G., Wolsey, L.A. (eds.) 50 Years of Integer Programming 1958-2008, pp. 561–618. Springer Berlin Heidelberg (2010)
20. Honda, K., Yoshida, N., Carbone, M.: Multiparty asynchronous session types. In: Necula, G.C., Wadler, P. (eds.) POPL. pp. 273–284. ACM (2008)
21. Kuroda, S.Y.: Classes of languages and linear-bounded automata. Information and Control 7(2), 207–223 (1964)
22. Lange, J., Tuosto, E.: Synthesising choreographies from local session types. In: Koutny, M., Ulidowski, I. (eds.) CONCUR. LNCS, vol. 7454, pp. 225–239. Springer (2012)
23. OASIS-Technical-Committee: OASIS WSBPEL TC, Web services business process execution language version 2.0 (2007), technical Report, OASIS, available at http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html