Research Challenges in Orchestration Synthesis

Davide Basile Maurice H. ter Beek
Formal Methods and Tools lab, ISTI–CNR, Pisa, Italy
{davide.basile,maurice.terbeek}@isti.cnr.it

Contract automata allow to formally define the behaviour of service contracts in terms of service offers and requests, some of which are moreover optional and some of which are necessary. A composition of contracts is said to be in agreement if all service requests are matched by corresponding offers. Whenever a composition of contracts is not in agreement, it can be refined to reach an agreement using the orchestration synthesis algorithm. This algorithm is a variant of the synthesis algorithm used in supervisory control theory and it is based on the fact that optional transitions are controllable, whereas necessary transitions are at most semi-controllable and cannot always be controlled. In fact, the resulting orchestration is such that as much of the behaviour in agreement is maintained. In this paper, we discuss recent developments of the orchestration synthesis algorithm for contract automata. Notably, we present a refined notion of semi-controllability and compare it with the original notion by means of examples. We then discuss the current limits of the orchestration synthesis algorithm and identify a number of research challenges together with a research roadmap.

1 Introduction

Orchestrations of services describe how control and data exchanges are coordinated in distributed service-based applications and systems. Their principled design is identified in [16] as one of the primary research challenges for the next 10 years, and the Service Computing Manifesto [16] points out that “Service systems have so far been built without an adequate rigorous foundation that would enable reasoning about them” and, moreover, that “The design of service systems should build upon a formal model of services”.

The problem of synthesising well-behaving orchestrations of services can be viewed as a specific instance of the more general problem of synthesising strategies in games [9, 7]. This can be solved using refined algorithms from supervisory control for discrete event systems [24, 1], which have well-established relationships with reactive systems synthesis [20], parity games [23], automated behaviour composition [21] and automated planning [17].

Contract automata are a specific type of finite state automata that are used to formally define the behaviour of service contracts. These automata express contracts in terms of both offers and requests [10]. When multiple contracts are composed, they are said to be in agreement if all service requests from one contract are matched by another contract’s corresponding offers. A composition of contracts that is not in agreement, can automatically be refined to reach an agreement by means of the orchestration synthesis algorithm, which is a variation of the synthesis algorithm used in supervisory control theory. This orchestration synthesis algorithm for contract automata is described in [8, 9].

The classic algorithm for synthesising a most permissive controller distinguishes transitions whose controllability is invariant [24, 1]. In service contracts, instead, the controllability of certain transitions may vary depending on specific conditions on the orchestration of contracts [9]. The contract automata library `CATLib` [5] implements contract automata and their operations (e.g., composition and synthesis). Orchestrations of contract automata abstract from their underlying realisation; an orchestrator is assumed

C. Aubert, C. Di Giusto, S. Fowler & L. Safina (Eds.):
16th Interaction and Concurrency Experience (ICE 2023)
EPTCS 383, 2023, pp. 73–90, doi:10.4204/EPTCS.383.5

to interact with the services to realise the overall behaviour as prescribed by the orchestration contract. The contract automata runtime environment CARE [6] implements an orchestrator that interprets the synthesised orchestration to coordinate the services, where each service is implementing a contract. Thus, CARE is explicating the low-level interactions that are abstracted in contract automata orchestrations. Notably, one aspect that is abstracted in contract automata and concretised at the implementation level is that of selecting the next transition to execute in the presence of choice. In [6], different implementations are proposed based on whether services may participate externally or internally in a choice.

This paper delves into challenges and research issues for orchestration synthesis of contract automata, given the latest developments in this field. In particular, we start by refining the current definition of semi-controllability to consider the aforementioned possible realisations of choices defined in [6]. We provide several examples to illustrate the differences between the refined definition and the original definition. The various definitions of semi-controllability lead to different sets of contract automata orchestrations, which we present in Figure 3 together with an example for each level of the orchestration hierarchy depicted. This allows us to highlight the unique characteristics of each level and to identify current issues in synthesising orchestrations of contract automata using these examples. Based on the issues presented, we then outline future research challenges in the orchestration synthesis of contract automata and a research roadmap to address them.

Related Work At last year's ICE 2022 workshop, the compositionality of communicating finite state machines (CFSM) with asynchronous semantics was discussed in [3]. Also contract automata are composable, enabling the modelling of systems of systems. Moreover, under certain specific conditions that were presented at the 2014 edition of ICE [11, 12], an orchestration of contract automata can be translated into a choreography of synchronous or asynchronous CFSM. The relation between multiparty session types and CFSM is discussed in [27]. Therefore, contract automata can be related to multiparty session types by exploiting their common relation with CFSM [11, 12, 27].

The contract automata approach is closer to [22], in which behavioural types are expressed as finite state automata of Mungo, called typestates [25]. Similarly to CARE, the runtime environment for contract automata [6], in Mungo finite state automata are used as behaviour assigned to Java classes (one automaton per class), with transition labels corresponding to methods of the classes. A tool to translate typestates into automata was presented at ICE 2020 [26]. CATApp, a graphical front-end tool for designing contract automata, is available in [19]. A tool similar to Mungo is JaTyC (Java Typestate Checker) [2].

The refined definition of semi-controllability presented in this paper closely aligns with the notion of weak receptiveness in team automata [14, 15]. However, the challenges addressed in this paper are primarily related to the problem of synthesising an orchestration of services and as such are not directly relevant to team automata.

Differently from the semi-controllability for orchestrations, a distinct notion of semi-controllability has been studied in [9, 4] for choreographies of services. Finally, while a runtime environment for the orchestration of services has been proposed in [6], this has yet to be realised for the case of choreographies, which could result in improvements in the notion of semi-controllability for choreographies.

Outline We start by providing some background on contract automata and orchestration synthesis in Section 2. We introduce a refined notion of semi-controllability in Section 3. In Section 4, we present several research challenges for orchestration synthesis of contract automata. We conclude in Section 5.

2 Background

We will begin by formally introducing contract automata and their synthesis operation. Contract automata are a type of finite state automata that use a partitioned alphabet of actions. A Contract Automaton (CA) can model either a single service or a composition of multiple services that perform actions. The number of services in a CA is known as its rank. If the rank of a CA is 1, then the contract is referred to as a principal (i.e., a single service).

The labels of a CA are vectors of atomic elements known as actions. Actions are categorised as either requests (prefixed by ?), offers (prefixed by !), or idle actions (represented by a distinguished symbol $-$). Requests and offers belong to the sets R and O, respectively, and they are pairwise disjoint. The states of a CA are vectors of atomic elements known as basic states. Labels are restricted to requests, offers or matches. In a request (resp. offer) label there is a single request (resp. offer) action and all other actions are idle. In a match label there is a single pair of request and offer actions that match, and all other actions are idle. The length of the vectors of states and labels is equal to the rank of the CA. For example, the label $[!a,?a,-,-]$ is a match where the request action $?a$ is matched by the offer action $!a$, and all other actions are idle. Note the difference between a request label (e.g., $[?a,-]$) and a request action (e.g., $?a$). A transition may also be called a request, offer or match according to its label. Figure 4 depicts three principal contracts, whilst Figure 5 depicts a contract of rank 3.

The goal of each service is to reach an accepting (*final*) state such that all its request (and possibly offer) actions are matched. Transitions are equipped with *modalities*, i.e., *necessary* ($\Box$) and *optional* ($\circ$) transitions, respectively [1]. Optional transitions are controllable, whereas necessary transitions can be uncontrollable (called *urgent* necessary transitions) or semi-controllable (called *lazy* necessary transitions). The resulting formalism is called *Modal Service Contract Automata* (MSCA). In the following definition, given a vector $\vec{a}$, its ith element is denoted by $\vec{a}_{(i)}$.

Definition 1 (MSCA)**.** *Given a finite set of states $\mathcal{Q} = \{q_1, q_2, \ldots\}$, an MSCA $\mathcal{A}$ of rank n is a tuple $\langle Q, \vec{q}_0, A^r, A^o, T, F\rangle$, with set of states $Q = Q_1 \times \ldots \times Q_n \subseteq \mathcal{Q}^n$, initial state $\vec{q}_0 \in Q$, set of requests $A^r \subseteq \mathsf{R}$, set of offers $A^o \subseteq \mathsf{O}$, set of final states $F \subseteq Q$, set of transitions $T \subseteq Q \times A \times Q$, where $A \subseteq (A^r \cup A^o \cup \{\bullet\})^n$, partitioned into* optional *transitions T° and* necessary *transitions $T^{\Box}$, with $T^{\Box}$ further partitioned into* urgent *necessary transitions $T^{\Box_u}$ and* lazy *necessary transitions $T^{\Box_l}$, and such that given $t = (\vec{q}, \vec{a}, \vec{q}\,') \in T$: i) $\vec{a}$ is either a request, an offer or a match; ii) if $\vec{a}$ is an offer, then $t \in T^{\circ}$; and iii) $\forall i \in 1 \ldots n$, $\vec{a}_{(i)} = \bullet$ implies $\vec{q}_{(i)} = \vec{q}\,'_{(i)}$.*

Composition of services is rendered through the composition of their MSCA models by means of the *composition operator* $\otimes$, which is a variant of a synchronous product. This operator basically interleaves or matches the transitions of the component MSCA, but, whenever two component MSCA are enabled to execute their respective request/offer action, then the match is forced to happen. Moreover, a match involving a necessary transition of an operand is itself necessary. The rank of the composed MSCA is the sum of the ranks of its operands. The vectors of states and actions of the composed MSCA are built from the vectors of states and actions of the component MSCA, respectively. In this paper, we will only consider principal contracts and compositions of principals, which will be automatically refined into orchestrations (as shown in Figure 2). However, it is important to note that contracts can be created by composing contracts with a rank of one or higher.

In a composition of MSCA, typically various properties are analysed. We are especially interested in *agreement*. The property of agreement requires to match all requests, whereas offers can go unmatched.

[1] Originally, in [8], the optional modality was called permitted and denoted with $\Diamond$. Since in contract automata the two modalities are a partition, the terminology has been updated to avoid confusion with modal transition systems, where $\Box \subseteq \Diamond$.

CA support the synthesis of the most permissive controller (mpc) known from the theory of supervisory control of discrete event systems [24, 18], where a finite state automaton model of a *supervisory controller* is synthesised from given (component) finite state automata that are composed. The synthesised automaton, if successfully generated (i.e., non-empty), is such that it is *non-blocking*, *controllable*, and *maximally permissive*. An automaton is said to be *non-blocking* if, from each state, at least one of the *final states* (distinguished stable states that represent completed 'tasks' [24]) can be reached without passing through so-called *forbidden states*, meaning that there is always a possibility to return to an accepted stable state (e.g., a final state).

The synthesised automaton is said to be *controllable* when only controllable transitions are disabled. Indeed, the supervisory controller is not permitted to directly block uncontrollable transitions from occurring; the controller is only allowed to disable them by preventing controllable actions from occurring. Finally, the fact that the resulting supervisory controller is said to be *maximally permissive* (or least restrictive) means that as much behaviour of the uncontrolled system as possible is present in the controlled system without violating neither the requirements, nor controllability nor the non-blocking condition.

Orchestration Synthesis As stated previously, optional transitions are controllable, whereas necessary transitions can be either uncontrollable (called *urgent*) or semi-controllable (called *lazy*). In the mpc synthesis (implemented in `CATLib` [9, 5]), all necessary transitions are *urgent*, i.e., they are always uncontrollable. This stems from the fact that traditionally uncontrollable transitions relate to an unpredictable environment.

When synthesising an orchestration of services, all necessary transitions are instead *lazy*, i.e., they are *semi-controllable* [8, 9]. A semi-controllable transition t is a transition that is either uncontrollable or controllable according to given conditions. In [9], different conditions are given according to whether the synthesis of an orchestration or a choreography is computed. In this paper, we only consider orchestrations. Below, we denote with $\mathit{Dangling}(\mathcal{A})$ the set of states that are not reachable from the initial state or cannot reach any final state. More in detail, a semi-controllable transition t is controllable if in a given portion $\mathcal{A}'$ of $\mathcal{A}$ there exists a semi-controllable match transition t', with source and target states not dangling, such that in both t and t' the *same* service, in the *same* local state, does the *same* request. Otherwise, t is uncontrollable.

Definition 2 (Controllability). *Let $\mathcal{A}$ be an MSCA and let $t = (\vec{q}_1, \vec{a}_1, \vec{q}_1{}') \in T_{\mathcal{A}}$. Then:*

- *if $t \in T_{\mathcal{A}}^{\bigcirc}$, then t is* controllable *(in $\mathcal{A}$);*
- *if $t \in T_{\mathcal{A}}^{\square_u}$, then t is* uncontrollable *(in $\mathcal{A}$);*
- *if $t \in T_{\mathcal{A}}^{\square_l}$, then t is* semi-controllable *(in $\mathcal{A}$).*

Moreover, given $\mathcal{A}' \subseteq \mathcal{A}$, if t is semi-controllable and $\exists t' = (\vec{q}_2, \vec{a}_2, \vec{q}_2{}') \in T_{\mathcal{A}'}^{\square}$ in $\mathcal{A}'$ such that $\vec{a}_2$ is a match, $\vec{q}_2, \vec{q}_2{}' \notin \mathit{Dangling}(\mathcal{A}')$, $\vec{q}_{1(i)} = \vec{q}_{2(i)}$, and $\vec{a}_{1(i)} = \vec{a}_{2(i)} = ?a$ for some $i \in 0 \ldots \mathit{rank}(\mathcal{A})$, then t is controllable *in $\mathcal{A}'$ (via t'). Otherwise, t is* uncontrollable *in $\mathcal{A}'$.*

The interpretation of optional/controllable and urgent/uncontrollable transitions is standard [24, 18]. In the upcoming section, we will delve into different understandings and interpretations of the concept of semi-controllability. We remark that the orchestration synthesis defined below does not support urgent transitions. The orchestration synthesis, as defined below, involves an iterative refinement of the initial automaton $\mathcal{A}$ (i.e., the composition of contracts). In each iteration, transitions are selectively pruned, and a set R of forbidden states is updated accordingly. A transition t is pruned under one of two conditions: if it is a request (thus violating the agreement property enforced by the orchestration), or if the target

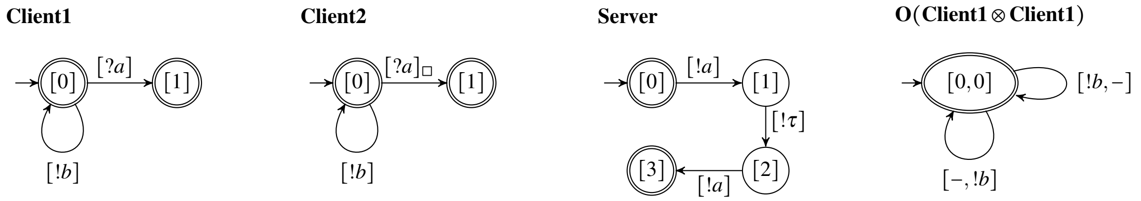

Figure 1: Contracts of **Client1**, **Client2** and **Server**, and orchestration **O**(**Client1** ⊗ **Client1**)

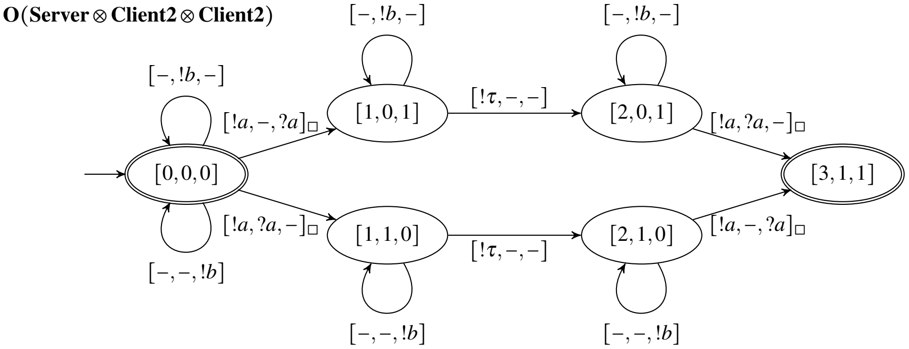

Figure 2: Orchestration **O**(**Server** ⊗ **Client2** ⊗ **Client2**)

state of t belongs to the set R computed up to that point. During the first iteration, all request transitions, including both lazy and optional ones, are pruned.

In Definition 2, the automaton $\mathcal{A}'$ represents an intermediate refinement of $\mathcal{A}$ (the starting composition) which occurs during an iteration of the synthesis process. Intuitively, the semi-controllable transition t of $\mathcal{A}$ is controllable in $\mathcal{A}'$ because there is another transition t' in $\mathcal{A}'$ matching the same request from the same service in the same state. Otherwise, if there is no such transition t' in $\mathcal{A}'$, then t is uncontrollable. Put differently, the controllability of t in $\mathcal{A}'$ relies on the presence of a corresponding transition t' within $\mathcal{A}'$ itself. If such a matching transition t' does not exist in $\mathcal{A}'$, then t is deemed uncontrollable.

Note that in Definition 2, it is not required for t and t' to be distinct. This implies that during the synthesis process, a semi-controllable match transition t can switch from being controllable to uncontrollable only after it has been pruned in a previous iteration. To clarify further, a semi-controllable match transition t can switch its controllability status from controllable to uncontrollable only when t is absent in the sub-automaton $\mathcal{A}'$ during the current iteration. If t is present in $\mathcal{A}'$ (i.e., it has not been pruned thus far), then, according to Definition 2, t is considered semi-controllable and controllable within $\mathcal{A}'$ via t itself. It is important to note that these considerations are applicable only if t is a match. Additionally, it is never the case that a semi-controllable transition t switches from uncontrollable to controllable since transitions are only removed during the synthesis process and are never added back.

The set R of forbidden states is updated at each iteration by adding source states of uncontrollable transitions and dangling states of the refined automaton in the current iteration. Specifically, when the synthesis process eliminates all transitions t' that satisfy the conditions for rendering the semi-controllable transition t controllable via t', then t becomes uncontrollable within the sub-automaton in the current iteration. It is worth noting that even if t was previously pruned in an earlier iteration, its source state $\vec{q}_1$ might still be reachable in the sub-automaton of the current iteration. Consequently, $\vec{q}_1$ is added to the set R. In the subsequent iteration, all transitions with target state $\vec{q}_1$ will be pruned. This pruning of transitions whose target is $\vec{q}_1$ can potentially render another previously pruned semi-controllable transition as uncontrollable, thereby adding its source state to the updated set R. This refinement process continues until no further transitions are pruned, and no additional states are added to R. The resulting refined automaton obtained at the end of the synthesis process represents the orchestration automaton.

The algorithm for synthesising an orchestration enforcing agreement of MSCA is defined below.

Definition 3 (MSCA orchestration synthesis)**.** *Let $\mathcal{A}$ be an MSCA and let $\mathcal{K}_0 = \mathcal{A}$ and $R_0 = Dangling(\mathcal{K}_0)$. We let the* orchestration synthesis function $f_o : MSCA \times 2^{Q} \rightarrow MSCA \times 2^{Q}$ *be defined as follows:*

$$f_o(\mathcal{K}_{i-1}, R_{i-1}) = (\mathcal{K}_i, R_i), \text{ with}$$

$$\begin{aligned} T_{\mathcal{K}_i} &= T_{\mathcal{K}_{i-1}} \setminus \{ (\vec{q} \rightarrow \vec{q}\,') = t \in T_{\mathcal{K}_{i-1}} \mid (\vec{q}\,' \in R_{i-1} \vee t \text{ is a request}) \} \\ R_i &= R_{i-1} \cup \{ \vec{q} \mid (\vec{q} \rightarrow) \in T_{\mathcal{A}}^{\Box_l} \text{ is uncontrollable in } \mathcal{K}_i \} \cup Dangling(\mathcal{K}_i) \end{aligned}$$

The orchestration automaton is obtained from the fixpoint of the function f_o. In the rest of the paper, if not stated otherwise, all necessary transitions in the examples are lazy (cf. Definition 1); for brevity and less cluttering in the figures, we denote them by $\Box$ rather than $\Box_l$.

Example 1. We provide an illustrative example to underline the differences between optional transitions, urgent necessary transitions and lazy necessary transitions. Figure 1 shows two client contracts and a server contract. Firstly, we discuss the difference between optional and necessary transitions. When all actions of the client contract are optional (**Client1**), there exists an orchestration of the composition of two **Client1** contracts, also depicted in Figure 1 (**O**(**Client1** ⊗ **Client1**)). Indeed the (transition labelled with the) request $?a$ is optional and can be removed to obtain the orchestration. If instead the request $?a$ was necessary (**Client2**), then there would be no orchestration for the composition of two **Client2** contracts, because the necessary request is never matched by a corresponding offer.

To illustrate the distinction between urgent and lazy necessary transitions, we consider also the server contract shown in Figure 1. If we were to employ the traditional mpc synthesis, the clients' necessary requests ($?a$) would be treated as urgent. In such a scenario, the orchestration of the composition between two clients and the server (generated using the mpc synthesis algorithm) would be empty, indicating that no feasible orchestration exists.

However, if the clients' necessary requests ($?a$) are considered lazy instead, an orchestration of the composition between the server and the two clients can be achieved (computed using the orchestration synthesis). This orchestration is depicted in Figure 2. In this case, the clients take turns fulfilling their lazy necessary requests. This alternating behaviour is not possible when the necessary requests are urgent.

The orchestration in Figure 2 is obtained after three iterations of the algorithm specified in Definition 3. Initially, $\mathcal{K}_0 = \mathcal{A} = \textbf{Server} \otimes \textbf{Client2} \otimes \textbf{Client2}$ and $R_0 = Dangling(\mathcal{A}) = \varnothing$.

With respect to the orchestration in Figure 2, the automaton $\mathcal{A}$ contains four additional transitions that are $t_1 = [1,0,1] \xrightarrow{[-,?a,-]_\Box} [1,1,1]$, $t_2 = [1,1,0] \xrightarrow{[-,-,?a]_\Box} [1,1,1]$, $t_3 = [1,1,1] \xrightarrow{[!\tau,-,-]} [2,1,1]$ and $t_4 = [2,1,1] \xrightarrow{[!a,-,-]} [3,1,1]$. In the first iteration, t_1 and t_2 are removed from $\mathcal{K}_1$ because they are request transitions. We have $T_{\mathcal{K}_1} = T_{\mathcal{K}_0} \setminus \{t_1, t_2\}$. Since there are no forbidden states, these are the only two transitions that are removed during the first iteration.

Concerning the set of forbidden states R_1, we have that $t_1 \in T_{\mathcal{A}}^{\Box_l}$ is controllable in $\mathcal{K}_1$ via transition $[0,0,0] \xrightarrow{[a!,a?,-]_\Box} [1,1,0]$. Similarly, $t_2 \in T_{\mathcal{A}}^{\Box_l}$ is controllable in $\mathcal{K}_1$ via $[0,0,0] \xrightarrow{[a!,-,a?]_\Box} [1,0,1]$. Hence, the source states of t_1 and t_2 will not be added to R_1. Concerning the set $Dangling(\mathcal{K}_1)$, state $[1,1,1]$ was the target of only t_1 and t_2. Moreover, state $[2,1,1]$ was the target of only t_3. Therefore, states $[1,1,1]$ and $[2,1,1]$ are unreachable in $\mathcal{K}_1$. We have that $R_1 = Dangling(\mathcal{K}_1) = \{[1,1,1],[2,1,1]\}$. In the subsequent iteration $i = 2$, since transition t_3 has target in R_1, we have $T_{\mathcal{K}_2} = T_{\mathcal{K}_1} \setminus \{t_3\}$, whilst $R_2 = R_1$.

Finally, we reach the fixpoint at iteration $i = 3$, where $T_{\mathcal{K}_3} = T_{\mathcal{K}_2}$ and $R_3 = R_2$. The finalising operations for obtaining the orchestration $\mathbf{O}$ in Figure 2 from the fixpoint $\mathcal{K}_3$ consist in removing the states in R_3, i.e., $Q_{\mathbf{O}} = Q_{\mathcal{K}_3} \setminus R_3$, and removing the remaining unreachable transitions in $\mathcal{K}_3$. In this case, transition $t_4 \in T_{\mathcal{K}_3}$ is removed from the orchestration, i.e., $T_{\mathbf{O}} = T_{\mathcal{K}_3} \setminus \{t_4\}$.

In the subsequent section, we will delve deeper into additional details and interpretations regarding the semi-controllable transitions of contract automata.

3 Refined Semi-Controllability

We start by introducing a refined notion of semi-controllability to be used in the orchestration synthesis, formalised below. After that, we discuss how this refined notion may assist to discard some counter-intuitive orchestrations.

Definition 4 (Refined Semi-Controllability)**.** *Let $\mathcal{A}$ be an MSCA and let $t = (\vec{q}_t, \vec{a}_t, \vec{q}_t{}') \in T_{\mathcal{A}}^{\Box_l}$. Moreover, given $\mathcal{A}' \subseteq \mathcal{A}$, if $\exists t' = (\vec{q}_{t'}, \vec{a}_{t'}, \vec{q}_{t'}{}') \in T_{\mathcal{A}'}^{\Box_l}$ in $\mathcal{A}'$ such that the following hold:*

1. *$\vec{a}_{t'}$ is a match, $\vec{q}_{t'}, \vec{q}_{t'}{}' \notin Dangling(\mathcal{A}')$, $\vec{q}_{t(j)} = \vec{q}_{t'(j)}$, $\vec{a}_{t(j)} = \vec{a}_{t'(j)} = ?a$, for some $j \in 0 \ldots rank(\mathcal{A})$; and*
2. *there exists a sequence of transitions $t_0, \ldots, t_n$ of $\mathcal{A}'$ such that $\forall i \in 0 \ldots n$, $t_i = (\vec{q}_i, \vec{a}_i, \vec{q}_i{}')$ and the following hold:*
   - *$\vec{q}_0 = \vec{q}_t$;*
   - *$t_n = t'$;*
   - *$\vec{q}_i, \vec{q}_i{}' \notin Dangling(\mathcal{A}')$; and*
   - *if $i < n$, then $\vec{a}_{i(j)} = -$ and $\vec{q}_i{}' = \vec{q}_{i+1}$;*

then t is controllable *in $\mathcal{A}'$ (via t'). Otherwise, t is* uncontrollable *in $\mathcal{A}'$.*

By comparing Definition 2 and Definition 4, we note that only the semi-controllable transitions have been refined, whilst the others are unaltered. Conditions 1 and 2 contain the constraints that are used to decide when a semi-controllable transition is controllable or uncontrollable. The constraints of Condition 1 are also present in Definition 2. The intuition is that a (refined) semi-controllable transition t becomes controllable if (similarly to Definition 2) in a given portion of $\mathcal{A}$, there exists a semi-controllable match transition t', with source and target states not dangling, such that in both t and t' the *same* service, in the *same* local state, does the *same* request. Condition 2 of Definition 4 imposes new further constraints. It requires that t' is *reachable* from the source state of t through a sequence of transitions where the service performing the request is idle.

Consider the Venn diagram in Figure 3. The outermost set *Orchestrations* contains all orchestrations of contract automata that are computed using the notion of semi-controllability of Definition 2. The innermost set *Refined* contains only those orchestrations that are computed using the refined notion of semi-controllability in Definition 4. Intuitively, the refined notion imposes a further constraint on *when* a semi-controllable transition is controllable. As a result, more semi-controllable transitions are uncontrollable than in the previous definition. This explains why *Refined* is contained in *Orchestrations*.

All the examples of semi-controllability available in the literature [13, 8, 9, 6] (e.g., Hotel service) and Figure 2 are orchestrations belonging to the set *Refined* in Figure 3. This means that by updating the notion of semi-controllability, all orchestrations of these examples remain unaltered.

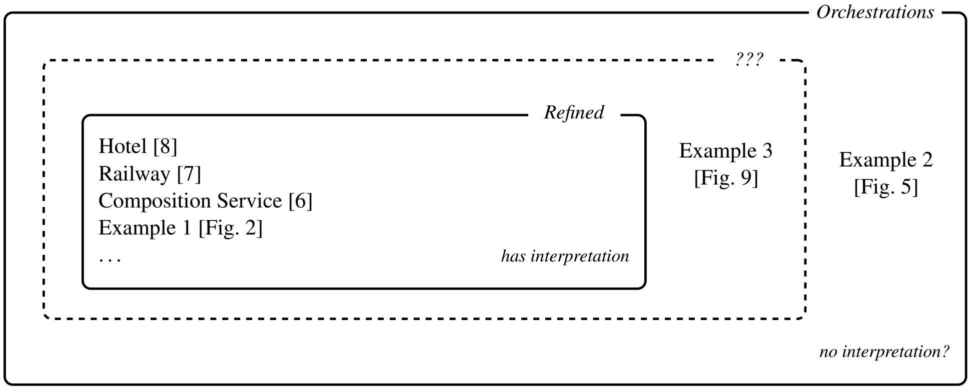

Figure 3: A Venn diagram showing the set of orchestrations of contract automata

Example 2. We now provide an example of an orchestration belonging to *Orchestrations* ∖ *Refined* (cf. Figure 3). We have three principal contracts, namely **A**lice, **B**ob and **C**arl, depicted in Figure 4. The contracts of **B**ob and **C**arl perform two alternative necessary requests. The contract of **A**lice has two branches. In each branch, a request of **B**ob and a request of **C**arl are fulfilled by corresponding offers.

Using the notion of semi-controllability from Definition 2, the synthesis algorithm of Definition 3 takes as input the composed automaton and returns the orchestration of the composition, depicted in Figure 5, which is a contract of rank 3. Indeed, for each necessary request of each service, there *exists* a match transition in the composition where the necessary request is fulfilled by a corresponding offer. In other words, for each necessary request of **B**ob and **C**arl, there exists an execution where the request is matched by a corresponding offer. For example, the composition $\mathbf{A}\mathrm{lice} \otimes \mathbf{B}\mathrm{ob} \otimes \mathbf{C}\mathrm{arl}$ contains the transition $t = [\mathrm{a}_1,\mathrm{b}_0,\mathrm{c}_0] \xrightarrow{[-,?d,-]_\square} [\mathrm{a}_1,\mathrm{b}_2,\mathrm{c}_0]$, which is semi-controllable. According to Definition 2, t is controllable (in $\mathbf{A}\mathrm{lice} \otimes \mathbf{B}\mathrm{ob} \otimes \mathbf{C}\mathrm{arl}$) via $t' = [\mathrm{a}_2,\mathrm{b}_0,\mathrm{c}_0] \xrightarrow{[!d,?d,-]_\square} [\mathrm{a}_4,\mathrm{b}_2,\mathrm{c}_0]$. Since t is controllable and it is not in agreement (i.e., the label of t is a request), this transition is pruned during the synthesis of the orchestration. We note that t is controllable in t' also in all sub-automaton of the composition computed in the various iterations of the synthesis algorithm, and in the final orchestration depicted in Figure 5.

Using the refined notion of semi-controllability of Definition 4, the orchestration of $\mathbf{A}\mathrm{lice} \otimes \mathbf{B}\mathrm{ob} \otimes \mathbf{C}\mathrm{arl}$ is empty (i.e., there is no orchestration). Consider again transition t. From state $[\mathrm{a}_1,\mathrm{b}_0,\mathrm{c}_0]$, it is not possible to reach any transition labelled by $[!d,?d,-]_\square$. It follows that t is uncontrollable. Hence, at some iteration i of the orchestration synthesis algorithm in Definition 3, state $[\mathrm{a}_1,\mathrm{b}_0,\mathrm{c}_0]$ becomes forbidden and it is added to the set R_i. At iteration $i+1$, the controllable transition $[\mathrm{a}_0,\mathrm{b}_0,\mathrm{c}_0] \xrightarrow{[!a,-,-]_\square} [\mathrm{a}_1,\mathrm{b}_0,\mathrm{c}_0]$ is pruned because its target state is forbidden. At the next iteration ($i+2$), the initial state $[\mathrm{a}_0,\mathrm{b}_0,\mathrm{c}_0]$ becomes forbidden, because there are semi-controllable transitions not in agreement exiting the initial state (e.g., $[\mathrm{a}_0,\mathrm{b}_0,\mathrm{c}_0] \xrightarrow{[-,?c,-]_\square} [\mathrm{a}_0,\mathrm{b}_1,\mathrm{c}_0]$) that are uncontrollable in the sub-automaton whose transitions are T_{i+2}. Since the initial state is forbidden, it follows that there is no orchestration for $\mathbf{A}\mathrm{lice} \otimes \mathbf{B}\mathrm{ob} \otimes \mathbf{C}\mathrm{arl}$.

Indeed, whenever the state $[\mathrm{a}_1,\mathrm{b}_0,\mathrm{c}_0]$ is reached, although **B**ob and **C**arl are still in their initial state, **B**ob can no longer perform the necessary request $?d$ and **C**arl can no longer perform the request $?f$. In fact, neither **B**ob nor **C**arl can decide internally which necessary request to execute from their current state. For example, there is no trace where the request $?c$ of **B**ob and the request $?f$ of **C**arl are matched.

The orchestrations belonging to *Refined* (i.e., orchestrations computed using the refined notion of semi-controllability given in Definition 4) have an intuitive interpretation when compared to the classic notion of uncontrollability. We recall that uncontrollable transitions are called *urgent* necessary transitions in MSCA, while semi-controllable transitions are called *lazy* necessary transitions.

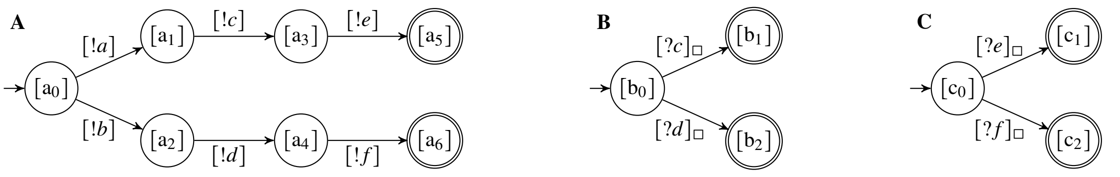

Figure 4: Contracts of **A**lice, **B**ob and **C**arl

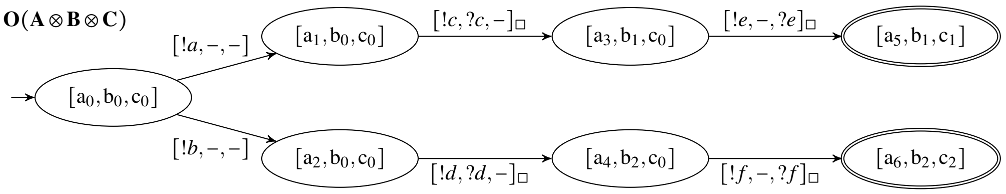

Figure 5: Orchestration $\mathbf{O}(\mathbf{A} \otimes \mathbf{B} \otimes \mathbf{C})$ of **A**lice $\otimes$ **B**ob $\otimes$ **C**arl

Intuitively, an urgent transition cannot be delayed, whereas this is the case for a lazy one. In a concurrent composition of agents, the scheduling of concurrent urgent necessary transitions is *uncontrollable*. Instead, concerning concurrent lazy necessary transitions, each agent *internally* decides its next lazy necessary transition to execute, but the orchestrator schedules when this transition will be executed, i.e., the scheduling is *controllable*. In Example 2, there is no orchestration because, for example, from state $[\mathrm{a}_1,\mathrm{b}_0,\mathrm{c}_0]$ there is no possible scheduling that allows the services to match all their necessary requests. Continuing Example 1, the orchestration in Figure 2 is non-empty because the scheduling of the actions in the orchestration is *controlled* by the orchestrator: one of the two necessary requests is scheduled to be matched only when the server has reached its internal state $[2]$. If instead the clients' necessary request $?a$ is urgent, then there exists no orchestration of the composition of two clients and the server. This is because in this case the scheduling is *uncontrollable*: it is not possible to schedule one of the two clients to have its necessary urgent request to be matched only when the server reaches the state $[2]$. In this case, the server should be ready to match the requests whenever they can be executed, without delaying them.

4 Research Challenges

In this section, we describe the currently known limits of the synthesis of orchestrations adopting either Definition 2 or Definition 4, we identify a number of research challenges to overcome these limits, and we propose a research roadmap aimed to tackle these challenges effectively.

First, the notion of semi-controllability introduced in [8, 5] and recalled in Definition 2 allows to synthesise orchestrations that may sometimes limit the capability of each service to perform internal choices. The contract automata formalism abstracts from the way that choices are made. Different implementations are possible in which each service may or may not decide the next step in an orchestration [6].

Consider again Example 2. Both **B**ob and **C**arl are able to perform two alternative necessary requests from their initial state. However, as shown in Figure 5, they are forbidden from internally deciding which necessary request is to be executed at runtime. If, for example, **B**ob selects the request $?d$ and **C**arl selects the request $?e$, then it is not possible for **A**lice to match both requests.

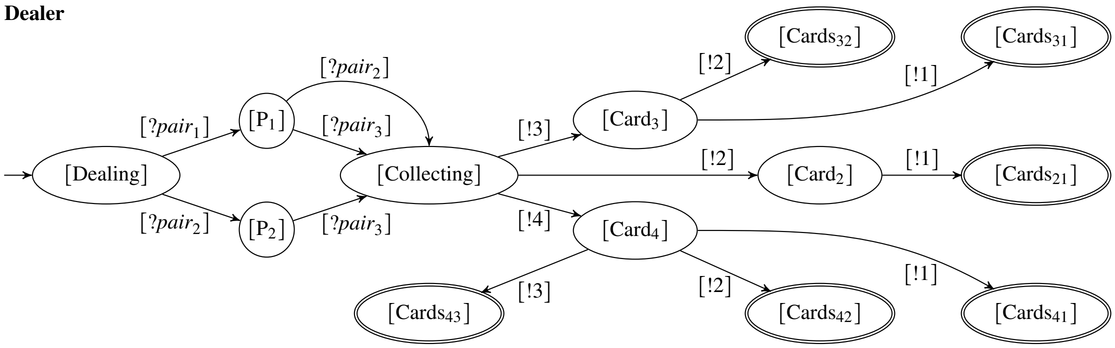

Figure 6: Contract of the **Dealer**

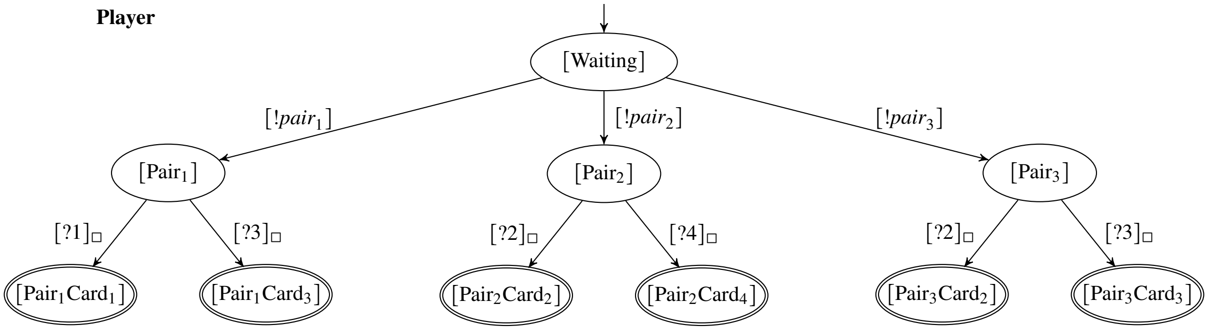

Figure 7: Contract of the **Player**

If we adopt the interpretation given previously (i.e., agents internally choose their necessary transitions and their scheduling is controllable) then we argue that the orchestration computed using Definition 2 is too abstract and should in fact be empty. This is indeed the case if Definition 4 were used instead of Definition 2.

The first research challenge is to identify a concrete application of services that perform necessary requests and whose orchestration belongs to the set *Orchestrations* $\smallsetminus$ *Refined*.

Solving this challenge could help provide an intuitive interpretation of these types of orchestrations. An application should be identified in which each service statically requires that for each necessary request there must exist an execution where this is eventually matched (cf. Definition 2). However, during execution, the choice of which necessary request is to be matched could be external to the service performing the necessary request. Even if the execution of different branches is determined externally, a service contract may still require all branches to be available in the composition. This could be due to the contract's need to enforce certain hyperproperties, such as non-interference or opacity.

Next, we illustrate the second research challenge. All examples of orchestrations currently available in the literature [7, 6, 5, 9, 8] reside inside the set *Refined* (cf. Figure 3). We showed in Example 2 an orchestration **O** not belonging to the set *Refined* and we argued that **O** is too abstract and should in fact be empty. We now provide another example of an orchestration not belonging to the set *Refined*. However, differently from Example 2, in this case the orchestration should not be empty.

Example 3. This example involves a simple card game with two players and a dealer. At the beginning of each round, the dealer chooses a pair of cards to deal to each player (i.e., each player receives a pair of cards). The dealer can select two out of three different pairs of cards:

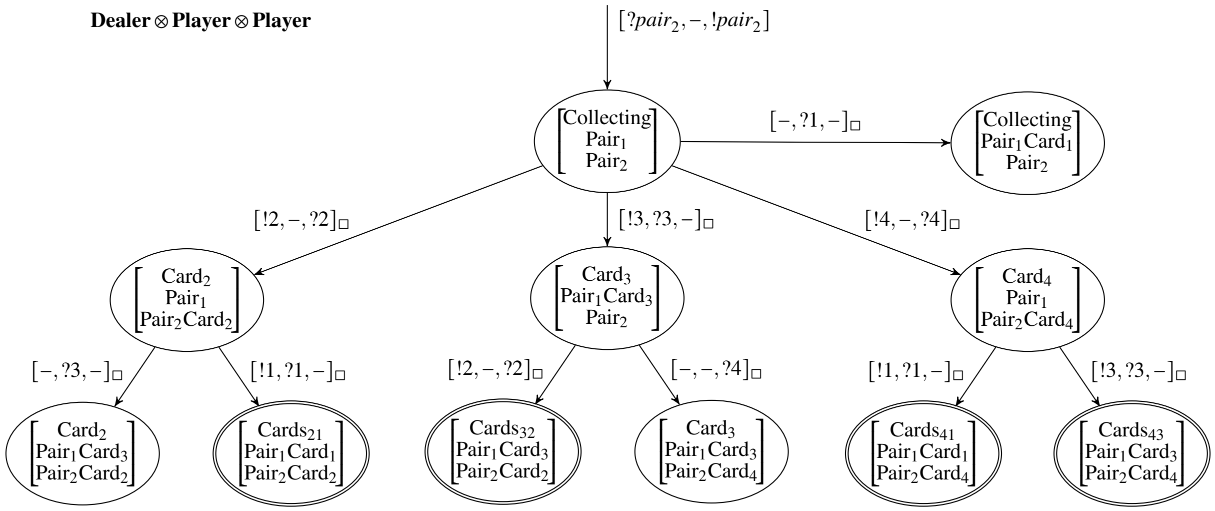

Figure 8: A fragment of the composition of **Dealer ⊗ Player ⊗ Player**

- Pair 1: card 1 and card 3;
- Pair 2: card 2 and card 4;
- Pair 3: card 2 and card 3.

After the dealer has dealt the pairs of cards, each player selects one of the two cards that was received. Once the players have selected their cards, the dealer collects the selected cards from each player. The goal of the game is for the dealer to avoid picking up two cards in ascending or equal order, which would result in the dealer losing. In other words, if the dealer picks up a card that is higher than the other card that was picked up or if two cards of the same value are picked up, the dealer loses. To ensure that the dealer never loses, the dealer has to choose the correct pairs of cards to deal. There are six possible ways to choose the pairs of cards, but only two of them guarantee a strategy for the dealer to collect the cards selected by the players in descending order. The strategy for the dealer consists of dealing to the players (in no particular order) Pair 1 and Pair 2. Indeed, in the remaining cases there exists the possibility that the players *internally* select the same card. In this case, there is no way of rearranging the transitions to avoid the same cards being picked by the dealer.

We modelled this above-mentioned problem as an orchestration of contracts, using the refined notion of semi-controllability. We only model one round of the game. The CA in Figure 6 models the dealer. Note that each request can be matched by either of the two players. Once the dealer has dealt the pairs of cards, the cards selected by the players are collected. Note that the two cards can only be collected in descending order. The CA in Figure 7 models a player. Once the player has received a card, the player decides internally which card to select. This internal decision is modelled as a choice among lazy necessary transitions.

The synthesis algorithm adopting the refined notion of semi-controllability from Definition 4 takes as input the composition of the dealer CA and two players CA and returns an empty orchestration. To explain why the resulting orchestration is empty, consider Figure 8 depicting a portion of the composition of the dealer with two players.

The state $[\text{Collecting}, \text{Pair}_1, \text{Pair}_2]$ is reached when the first player receives $pair_1$ and the second player receives $pair_2$. A symmetric argument holds for state $[\text{Collecting}, \text{Pair}_2, \text{Pair}_1]$, not depicted here.

The transition

$$[\mathrm{Card}_2, \mathrm{Pair}_1, \mathrm{Pair}_2\mathrm{Card}_2] \xrightarrow{[-,?3,-]\square} [\mathrm{Card}_2, \mathrm{Pair}_1\mathrm{Card}_3, \mathrm{Pair}_2\mathrm{Card}_2]$$

is uncontrollable according to Definition 4. Indeed, from state $[\mathrm{Card}_2, \mathrm{Pair}_1, \mathrm{Pair}_2\mathrm{Card}_2]$ it is not possible to reach state $[\mathrm{Collecting}, \mathrm{Pair}_1, \mathrm{Pair}_2]$. This makes the state $[\mathrm{Card}_2, \mathrm{Pair}_1, \mathrm{Pair}_2\mathrm{Card}_2]$ forbidden. Hence, to avoid reaching a forbidden state, the algorithm prunes the transition

$$[\mathrm{Collecting}, \mathrm{Pair}_1, \mathrm{Pair}_2] \xrightarrow{[!2,-,?2]\square} [\mathrm{Card}_2, \mathrm{Pair}_1, \mathrm{Pair}_2\mathrm{Card}_2]$$

which is in fact controllable according to Definition 4. Indeed, from state $[\mathrm{Collecting}, \mathrm{Pair}_1, \mathrm{Pair}_2]$ it is possible to reach the transition

$$[\mathrm{Card}_3, \mathrm{Pair}_1\mathrm{Card}_3, \mathrm{Pair}_2] \xrightarrow{[!2,-,?2]\square} [\mathrm{Cards}_{32}, \mathrm{Pair}_1\mathrm{Card}_3, \mathrm{Pair}_2\mathrm{Card}_2]$$

via a transition in which the second player is idle. However, during the synthesis algorithm also the state $[\mathrm{Card}_3, \mathrm{Pair}_1\mathrm{Card}_3, \mathrm{Pair}_2]$ becomes forbidden due to its outgoing necessary transition, which is uncontrollable according to Definition 2. This in turn causes the pruning of transition

$$[\mathrm{Collecting}, \mathrm{Pair}_1, \mathrm{Pair}_2] \xrightarrow{[!3,?3,-]\square} [\mathrm{Card}_3, \mathrm{Pair}_1\mathrm{Card}_3, \mathrm{Pair}_2]$$

which is controllable. Once the transition has been pruned, the transition

$$[\mathrm{Collecting}, \mathrm{Pair}_1, \mathrm{Pair}_2] \xrightarrow{[!2,-,?2]\square} [\mathrm{Card}_2, \mathrm{Pair}_1, \mathrm{Pair}_2\mathrm{Card}_2]$$

which was previously controllable becomes uncontrollable. This makes the state $[\mathrm{Collecting}, \mathrm{Pair}_1, \mathrm{Pair}_2]$ forbidden. Note, however, that $[\mathrm{Collecting}, \mathrm{Pair}_1, \mathrm{Pair}_2]$ should *not* be forbidden. Indeed, from that state, for each pair of cards selected by the players, the dealer has a strategy to pick them in the correct order:

- if player 1 selects card 1 and player 2 selects card 2, then execute $[!2,-,?2],[!1,?1,-]$;
- if player 1 selects card 1 and player 2 selects card 4, then execute $[!4,-,?4],[!1,?1,-]$;
- if player 1 selects card 3 and player 2 selects card 2, then execute $[!3,?3,-],[!2,-,?2]$;
- if player 1 selects card 3 and player 2 selects card 4, then execute $[!4,-,?4],[!3,?3,-]$.

This example shows that there are cases for which Definition 4 is too restrictive. In this case, the orchestration can be computed using Definition 2, and it is displayed in Figure 9.

To better understand the underlying assumption of Definition 4, we need to decouple the moment in which a service *selects* which transition it will execute from the moment in which a service *executes* that transition. The underlying assumption of Definition 4 is that these two moments are not decoupled.

For example, the first player whose internal state is Pair_1 could select and execute ?3 also from state $[\mathrm{Card}_2, \mathrm{Pair}_1, \mathrm{Pair}_2\mathrm{Card}_2]$, while the strategy described above assumes that the player selects a card in state $[\mathrm{Collecting}, \mathrm{Pair}_1, \mathrm{Pair}_2]$. In fact, the current implementation of the contract automata runtime environment CARE [6] allows the decoupling of these two moments. Once state $[\mathrm{Collecting}, \mathrm{Pair}_1, \mathrm{Pair}_2]$ is reached, the orchestrator interacts with both players and, based on their choices, correctly schedules the transitions of the dealer and the players. This means that the players select their next action in state $[\mathrm{Collecting}, \mathrm{Pair}_1, \mathrm{Pair}_2]$ and afterwards their execution is bounded to the transition they have selected. Summarising, Example 2 has showed that in some cases Definition 2 is too abstract, whereas Example 3 has showed that in some cases Definition 4 is too restrictive.

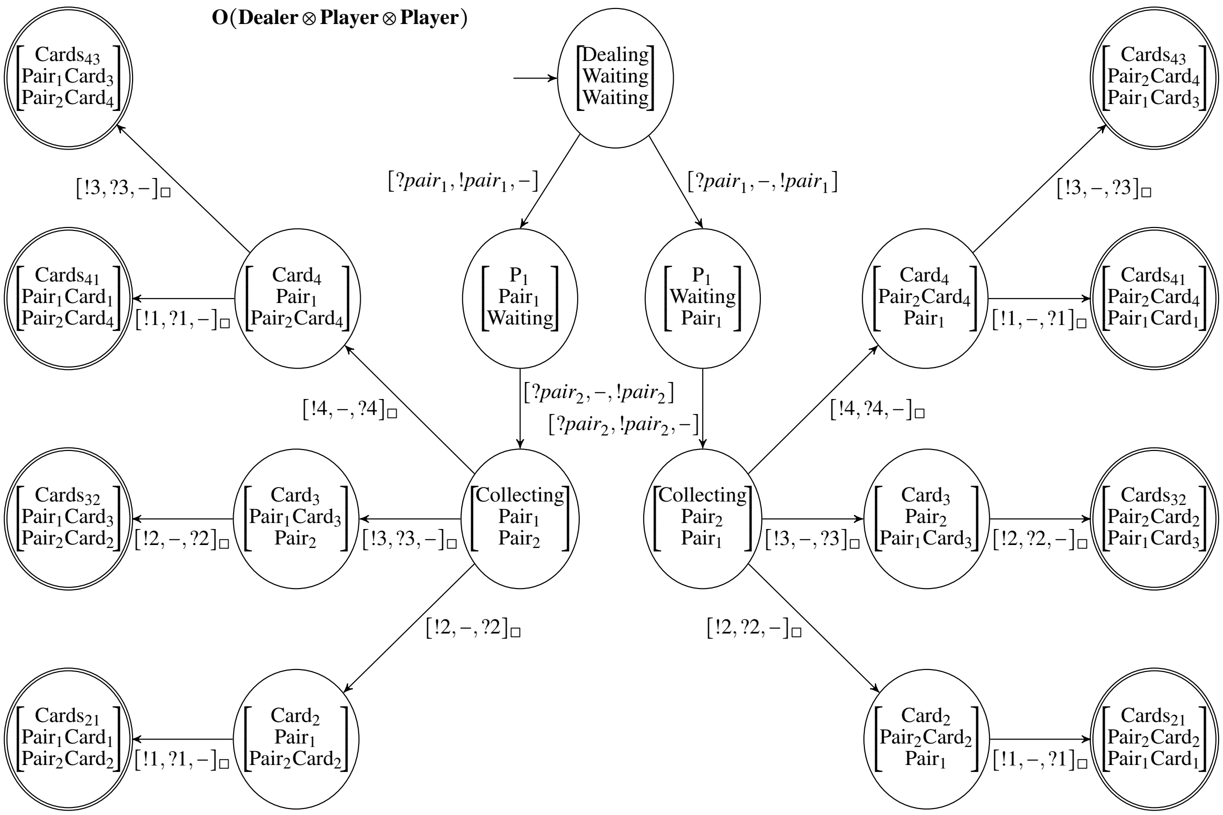

Figure 9: Orchestration **O**(**Dealer** ⊗ **Player** ⊗ **Player**)

The second research challenge is to identify a notion of semi-controllability capable of discarding orchestrations such as the one in Example 2 and providing non-empty orchestrations in scenarios such as the one described in Example 3.

The resulting, currently unknown set of orchestrations that would be identified by the notion of semi-controllability that solves this challenge is depicted in Figure 3 with dashed lines.

We continue by discussing further research challenges for the orchestration synthesis of contract automata. An important aspect is the ability to scale to large orchestrations when many service contracts are composed. We note that computing Definition 4 is harder than computing Definition 2, due to the additional constraint of reachability which requires a visit of the automaton. Decoupling the moment in which a service selects a choice from the moment in which the selected choice is executed, could further increase the hardness of deciding when a lazy necessary transition is controllable.

Consider again the CA in Figure 1. From their initial state, both **B**ob and **C**arl have two choices. If, instead of two principals, we had ten principals whose behaviour is similar to that of **B**ob and **C**arl, then there would be 2^{10} possible combinations of (internal) choices the services could make.

The third research challenge is to provide scalable solutions for synthesising orchestrations.

Generally speaking, the behaviour of an orchestration that belongs to the unknown dotted set of Figure 3 must be a sub-automaton of an orchestration computed using Definition 2 and a super-automaton of an orchestration computed using Definition 4. Indeed, Definition 2 can be used as an upper bound and Definition 4 as a lower bound to approximate the behaviour of such an orchestration.

Finally, we discuss the last research challenge identified in this paper. We previously formalised the notion of lazy necessary request that is semi-controllable according to either Definition 2 or Definition 4. We noted that Definition 2 may exclude the case in which, in the presence of a choice, a service may internally select its necessary transition. Instead, Definition 4 may exclude the case in which, in the presence of a choice, the moment in which the service internally selects its necessary transition is decoupled from the moment in which the selected necessary transition is executed. In other words, we identified two *requirements* that an orchestration of services should satisfy: *independence* and *decoupling* of choices.

The fourth research challenge is to consolidate a set of requirements that a desirable orchestration of service contracts must satisfy.

The requirements that would solve this challenge should be established incrementally, as discussed in this paper. Formal definitions of necessary service transitions and practical examples are useful to identify the ideal set of requirements that an orchestration of services should satisfy. Of course, these requirements are entangled with the underlying execution support of an orchestration of services, which was recently proposed in [6].

4.1 Research Roadmap

We have presented a series of research challenges associated with the orchestration of contract automata. We now propose a potential research roadmap aimed at tackling these challenges effectively. However, it is necessary to further examine the concepts described below to determine their validity.

Specifying Choices We propose to concretise the selection of the next transition to execute at contract automata level, distinguishing between internal and external selections. Presently, this distinction is abstracted away within contracts and handled by the underlying execution support. Our rationale is that abstracting from the selection process may lead to scalability challenges. Specifically, if a transition is selected internally, it must always be available, whereas an externally selected transition can be removed from the orchestration. In essence, internal selection imposes stricter requirements than external selection. Consequently, treating all selections as internal to ensure independence of choice leads to larger state spaces. For instance, the issue highlighted in Example 2 arises due to the presence of externally selected transitions. By allowing contracts to specify which transitions are internally or externally selected, we can potentially reduce the state space, as compared to considering all choices as internal.

Pursuing the above has important implications. Firstly, it necessitates updating accordingly the underlying execution support, `CARE`, to align it with the contract automata specifications. This entails reducing the implementation freedom for each choice to adhere to the contract's explicit selection of the next transition. By explicating choices within contracts, we establish the interpretation of necessary requests discussed in this paper. In this interpretation, a service *internally* decides to perform a necessary request, but the scheduling of the execution of the request is controlled by the orchestrator. In other words, optional actions are externally selected, whereas necessary actions are internally selected. Consequently, by explicitly stating choices in contracts, we can address the first research challenge. Indeed, all necessary requests would be internally selected. Scenarios like the one outlined in the context of the first research challenge (i.e., external necessary requests) would be practically ruled out.

Another implication relates to the fourth research challenge, which entails consolidating a set of requirements for effective orchestrations. Notably, if choices are explicitly specified in contracts, the requirement of independence of choice can be removed.

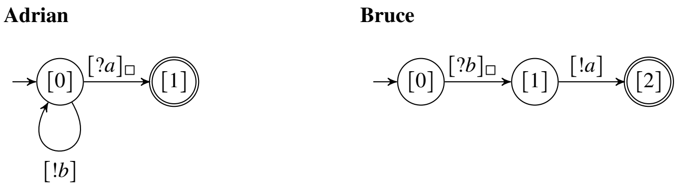

Figure 10: Two contracts whose orchestration requires further investigation

Implementing the Decoupling of Choices The second research challenge, as mentioned previously, revolves around the absence of the decoupling of choice requirement in both Definitions 2 and 4. This requirement suggests a potential implementation of semi-controllable transitions and may help identify the currently unknown set of orchestrations in Figure 3. Currently, a semi-controllable transition is defined as a transition that can be either controllable or uncontrollable based on a global condition of the automaton. However, decoupling the moment when a service internally selects a transition from the moment when the transition is executed might require splitting a semi-controllable transition into two distinct transitions.

Reasoning in this way suggests that a semi-controllable transition could potentially be represented as two consecutive transitions. The first transition would be uncontrollable, capturing the internal selection, while the subsequent transition would be controllable and responsible for executing the action. For example, consider a semi-controllable transition $\mathtt{[q]}\xrightarrow{[?a]_{\Box_l}}\mathtt{[q']}$, which would be split into two transitions: $t_1 = \mathtt{[q]}\xrightarrow{[\tau]_{\Box_u}}\mathtt{[i]}$ and $t_2 = \mathtt{[i]}\xrightarrow{[?a]}\mathtt{[q']}$. Here, t_1 represents an uncontrollable silent transition to an intermediate, non-final state, while t_2 is controllable and executes the action. This approach suggests that the orchestrator cannot control the internal selection made with t_1, but it can control and schedule the execution of the action indicated by t_2. Moreover, an important consequence of the fact that the intermediate state is non-final, is that t_2 must eventually be executed.

Further exploration is required to determine whether this interpretation of semi-controllability solves the third research challenge. In particular, there are still corner cases that require further investigation. For instance, consider the contracts in Figure 10. Although an orchestration could be obtained by matching the necessary request ?b of Bruce first and only afterwards the necessary request ?a of Adrian, this orchestration is not supported by the notion of semi-controllability outlined above. In this orchestration, Adrian internally selects the request ?a and the orchestrator schedules the request of Adrian to be matched later after Adrian matches the request of Bruce.

Furthermore, we envision the establishment of a clear separation between optional and necessary transitions on the one hand and controllable and uncontrollable transitions on the other. All necessary requests should be categorised as lazy/semi-controllable, thus effectively excluding urgent necessary requests from contracts. This implies that contract automata with optional and necessary transitions should be transformed into automata with solely controllable and uncontrollable transitions, which are known as plant automata in supervisory control theory. It is worth noting that all uncontrollable transitions will serve as silent moves to represent the internal selection of a necessary transition.

Experimental Validation of Performance The third research challenge highlights the issue of scalability and proposes the adoption of Definition 2 as an upper bound for the set of orchestrations. However, it remains unclear whether the synthesis process using Definition 2 is faster compared to synthesising using the mapped plant automaton as suggested earlier. Definition 2 necessitates a visit of the automaton at each iteration of the synthesis process to determine whether a semi-controllable transition is controllable or uncontrollable. This requirement is not present in a plant automaton consisting solely of controllable and uncontrollable transitions. On the other hand, the suggested mapping approach increases the state space of the automata by introducing an additional state for each necessary transition. As a result, it is essential to conduct further experimental research to assess the effectiveness of utilising Definition 2 as an upper bound for the set of orchestrations. This research should involve measuring the performance and efficiency of the synthesis process when employing Definition 2 and comparing it with the approach based on the mapped plant automaton.

5 Conclusion

We have presented a number of research challenges related to the orchestration synthesis of contract automata. Initially, we proposed a novel refined definition of semi-controllability and compared it to the current definition through illustrative examples. We identified various sets of orchestrations, as showed in Figure 3. Additionally, we informally discussed two prerequisites that the orchestration of contracts should satisfy: independence and decoupling of choices. Furthermore, we evaluated the current formal definitions of semi-controllability based on these requirements, which generated a series of research questions regarding the orchestration synthesis of contract automata, to be addressed in future work, possibly by following the proposed research roadmap.

Acknowledgements We would like to thank the reviewers for their useful comments. This work has been partially supported by the Italian MIUR PRIN 2017FTXR7S project IT MaTTerS (Methods and Tools for Trustworthy Smart Systems) and the Italian MUR PRIN 2020TL3X8X project T-LADIES (Typeful Language Adaptation for Dynamic, Interacting and Evolving Systems).

References

[1] Eugene Asarin, Oded Maler, Amir Pnueli & Joseph Sifakis (1998): *Controller Synthesis for Timed Automata*. *IFAC Proc. Vol.* 31(18), pp. 447–452, doi:10.1016/S1474-6670(17)42032-5.

[2] Lorenzo Bacchiani, Mario Bravetti, Marco Giunti, João Mota & António Ravara (2022): *A Java typestate checker supporting inheritance*. *Sci. Comput. Program.* 221, doi:10.1016/j.scico.2022.102844.

[3] Franco Barbanera, Ivan Lanese & Emilio Tuosto (2022): *On Composing Communicating Systems*. In Clément Aubert, Cinzia Di Giusto, Larisa Safina & Alceste Scalas, editors: *Proceedings of the 15th Interaction and Concurrency Experience (ICE'22)*, *EPTCS* 365, pp. 53–68, doi:10.4204/EPTCS.365.4.

[4] Davide Basile & Maurice H. ter Beek (2021): *A Clean and Efficient Implementation of Choreography Synthesis for Behavioural Contracts*. In Ferruccio Damiani & Ornela Dardha, editors: *Proceedings of the 23rd IFIP WG 6.1 International Conference on Coordination Models and Languages (COORDINATION'21)*, *LNCS* 12717, Springer, pp. 225–238, doi:10.1007/978-3-030-78142-2_14.

[5] Davide Basile & Maurice H. ter Beek (2022): *Contract Automata Library*. *Sci. Comput. Program.* 221, doi:10.1016/j.scico.2022.102841. Available at `https://github.com/contractautomataproject/ContractAutomataLib`.

[6] Davide Basile & Maurice H. ter Beek (2023): *A Runtime Environment for Contract Automata*. In Marsha Chechik, Joost-Pieter Katoen & Martin Leucker, editors: *Proceedings of the 25th International Symposium on Formal Methods (FM'23)*, *LNCS* 14000, Springer, pp. 550–567, doi:10.1007/978-3-031-27481-7_31.

[7] Davide Basile, Maurice H. ter Beek, Laura Bussi & Vincenzo Ciancia (2023): *A Toolchain for Strategy Synthesis with Spatial Properties*. *Int. J. Softw. Tools Technol. Transf.*

[8] Davide Basile, Maurice H. ter Beek, Pierpaolo Degano, Axel Legay, Gian Luigi Ferrari, Stefania Gnesi & Felicita Di Giandomenico (2020): *Controller synthesis of service contracts with variability*. *Sci. Comput. Program.* 187, doi:10.1016/j.scico.2019.102344.

[9] Davide Basile, Maurice H. ter Beek & Rosario Pugliese (2020): *Synthesis of Orchestrations and Choreographies: Bridging the Gap between Supervisory Control and Coordination of Services*. *Log. Methods Comput. Sci.* 16(2), pp. 9:1–9:29, doi:10.23638/LMCS-16(2:9)2020.

[10] Davide Basile, Pierpaolo Degano & Gian Luigi Ferrari (2016): *Automata for Specifying and Orchestrating Service Contracts*. *Log. Methods Comput. Sci.* 12(4), pp. 6:1–6:51, doi:10.2168/LMCS-12(4:6)2016.

[11] Davide Basile, Pierpaolo Degano, Gian Luigi Ferrari & Emilio Tuosto (2014): *From Orchestration to Choreography through Contract Automata*. In Ivan Lanese, Alberto Lluch Lafuente, Ana Sokolova & Hugo Torres Vieira, editors: *Proceedings of the 7th Interaction and Concurrency Experience (ICE'14)*, *EPTCS* 166, pp. 67–85, doi:10.4204/EPTCS.166.8.

[12] Davide Basile, Pierpaolo Degano, Gian Luigi Ferrari & Emilio Tuosto (2016): *Relating two automata-based models of orchestration and choreography*. *J. Log. Algebr. Methods Program.* 85(3), pp. 425–446, doi:10.1016/j.jlamp.2015.09.011.

[13] Davide Basile, Felicita Di Giandomenico, Stefania Gnesi, Pierpaolo Degano & Gian Luigi Ferrari (2017): *Specifying Variability in Service Contracts*. In: *Proceedings of the 11th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'17)*, ACM, pp. 20–27, doi:10.1145/3023956.3023965.

[14] Maurice H. ter Beek, Josep Carmona, Rolf Hennicker & Jetty Kleijn (2017): *Communication Requirements for Team Automata*. In Jean-Marie Jacquet & Mieke Massink, editors: *Proceedings of the 19th International Conference on Coordination Models and Languages (COORDINATION'17)*, *LNCS* 10319, Springer, pp. 256–277, doi:10.1007/978-3-319-59746-1_14.

[15] Maurice H. ter Beek, G. Cledou, R. Hennicker & J. Proença (2023): *Can we Communicate? Using Dynamic Logic to Verify Team Automata*. In M. Chechik, J.-P. Katoen & M. Leucker, editors: *Proceedings of the 25th International Symposium on Formal Methods (FM'23)*, *LNCS* 14000, Springer, pp. 122–141, doi:10.1007/978-3-031-27481-7_9.

[16] Athman Bouguettaya, Munindar P. Singh, Michael N. Huhns, Quan Z. Sheng, Hai Dong, Qi Yu, Azadeh Ghari Neiat, Sajib Mistry, Boualem Benatallah, Brahim Medjahed, Mourad Ouzzani, Fabio Casati, Xumin Liu, Hongbing Wang, Dimitrios Georgakopoulos, Liang Chen, Surya Nepal, Zaki Malik, Abdelkarim Erradi, Yan Wang, M. Brian Blake, Schahram Dustdar, Frank Leymann & Michael P. Papazoglou (2017): *A Service Computing Manifesto: The Next 10 Years*. *Commun. ACM* 60(4), pp. 64–72, doi:10.1145/2983528.

[17] Alberto Camacho, Meghyn Bienvenu & Sheila A. McIlraith (2019): *Towards a Unified View of AI Planning and Reactive Synthesis*. In: *Proceedings of the 29th International Conference on Automated Planning and Scheduling (ICAPS'18)*, AAAI, pp. 58–67, doi:10.1609/icaps.v29i1.3460.

[18] Christos G. Cassandras & Stéphane Lafortune (2006): *Introduction to Discrete Event Systems*. Springer, doi:10.1007/978-0-387-68612-7.

[19] `CATApp`. Available at `https://github.com/contractautomataproject/ContractAutomataApp`.

[20] Rüdiger Ehlers, Stéphane Lafortune, Stavros Tripakis & Moshe Y. Vardi (2017): *Supervisory control and reactive synthesis: a comparative introduction*. *Discret. Event Dyn. Syst.* 27(2), pp. 209–260, doi:10.1007/s10626-015-0223-0.

[21] Paolo Felli, Nitin Yadav & Sebastian Sardina (2017): *Supervisory Control for Behavior Composition*. *IEEE Trans. Autom. Control* 62(2), pp. 986–991, doi:10.1109/TAC.2016.2570748.

[22] Dimitrios Kouzapas, Ornela Dardha, Roly Perera & Simon J. Gay (2018): *Typechecking protocols with Mungo and StMungo: A session type toolchain for Java*. *Sci. Comput. Program.* 155, pp. 52–75, doi:10.1016/j.scico.2017.10.006.

[23] Michael Luttenberger, Philipp J. Meyer & Salomon Sickert (2020): *Practical synthesis of reactive systems from LTL specifications via parity games*. *Acta Inform.* 57(1-2), pp. 3–36, doi:10.1007/s00236-019-00349-3.

[24] Peter J. Ramadge & Walter M. Wonham (1987): *Supervisory Control of a Class of Discrete Event Processes*. *SIAM J. Control Optim.* 25(1), pp. 206–230, doi:10.1137/0325013.

[25] Robert E. Strom & Shaula Yemini (1986): *Typestate: A Programming Language Concept for Enhancing Software Reliability*. *IEEE Trans. Softw. Eng.* 12(1), pp. 157–171, doi:10.1109/TSE.1986.6312929.

[26] André Trindade, João Mota & António Ravara (2020): *Typestates to Automata and back: a tool*. In Julien Lange, Anastasia Mavridou, Larisa Safina & Alceste Scalas, editors: *Proceedings of the 13th Interaction and Concurrency Experience (ICE'20)*, *EPTCS* 324, pp. 25–42, doi:10.4204/EPTCS.324.4.

[27] Nobuko Yoshida, Fangyi Zhou & Francisco Ferreira (2021): *Communicating Finite State Machines and an Extensible Toolchain for Multiparty Session Types*. In Evripidis Bampis & Aris Pagourtzis, editors: *Proceedings of the 23rd International Symposium on Fundamentals of Computation Theory (FCT'21)*, *LNCS* 12867, Springer, pp. 18–35, doi:10.1007/978-3-030-86593-1_2.