Logical Methods in Computer Science
Volume 16, Issue 2, 2020, pp. 9:1–9:29
https://lmcs.episciences.org/

Submitted Oct. 05, 2019
Published Jun. 03, 2020

SYNTHESIS OF ORCHESTRATIONS AND CHOREOGRAPHIES: BRIDGING THE GAP BETWEEN SUPERVISORY CONTROL AND COORDINATION OF SERVICES

DAVIDE BASILE [a], MAURICE H. TER BEEK [a], AND ROSARIO PUGLIESE [b]

[a] ISTI–CNR, Pisa, Italy
e-mail address: {davide.basile,maurice.terbeek}@isti.cnr.it

[b] University of Florence, Italy
e-mail address: rosario.pugliese@unifi.it

Abstract. We present a number of contributions to bridging the gap between supervisory control theory and coordination of services in order to explore the frontiers between coordination and control systems. Firstly, we modify the classical synthesis algorithm from supervisory control theory for obtaining the so-called most permissive controller in order to synthesise orchestrations and choreographies of service contracts formalised as contract automata. The key ingredient to make this possible is a novel notion of controllability. Then, we present an abstract parametric synthesis algorithm and show that it generalises the classical synthesis as well as the orchestration and choreography syntheses. Finally, through the novel abstract synthesis, we show that the concrete syntheses are in a refinement order. A running example from the service domain illustrates our contributions.

1. Introduction

Services are ubiquitous in today's society. Examples include finances, healthcare, and tourism (e.g. booking services). Service-oriented computing (SOC) is "the discipline that seeks to develop computational abstractions, architectures, techniques, and tools to support services broadly" [16]. According to this paradigm, services are well-defined, self-contained, and stand-alone software modules that provide some standard business functionality. As such, services can serve as building blocks for the rapid, low-cost development of distributed applications in heterogeneous environments. Services used in composite applications are not limited to new service implementations, but may also include adapted and wrapped fragments of existing applications. The strength of SOC is composing multiple, distributed services into more powerful applications. This reuse through composition provides businesses a means to reduce the cost and risks of developing new applications.

Service composition is thus a key challenge for the full realisation of the SOC paradigm. As such, it can benefit from and contribute to emerging research directions inspired by cloud computing, IoT, social computing, and mobile computing, to name but a few. For instance,

Key words and phrases: Service Contracts, Contract Automata, Controller Synthesis, Orchestration, Choreography.

DOI:10.23638/LMCS-16(2:9)2020

the composition of cloud services requires the coordination of hardware and software resources across various layers. The IoT concept of smart cities concerns the large-scale composition of diverse and heterogeneous digital devices and services to provide multiple real-time, end user customised functionalities. Service composition based on relations in today's large social networks is challenging due to size and complexity of the resultant big data. In mobile environments, service composition is required to consider the intrinsic dynamicity and its effect on QoS aspects concerning security and reliability.

Two approaches are widely adopted for coordinating services by means of service composition: *orchestration* and *choreography*. Intuitively, an orchestration yields the description of a distributed workflow from "one party's perspective" [41], whereas a choreography describes the behaviour of the involved parties from a "global viewpoint" [34]. In an orchestrated model, the service components are coordinated by a special component, the *orchestrator*, which, by interacting with them, dictates the workflow at runtime. In a choreographed model, instead, the service components autonomously execute and interact with each other on the basis of a local control flow expected to comply with their role as specified by the global viewpoint. Ideally, a choreographed model is thought to be more efficient due to the absence of the overhead of communications with the orchestrator. Any choreography can trivially be transformed into an orchestration of services, by adding an idle orchestrator. Similarly, by explicitly adding an orchestrator and its interactions with the service components, and hence the relative overhead, an orchestration of services can be transformed into a choreography.

Despite the key impact that SOC can have on other contemporary computing paradigms, as already mentioned before, the recent 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 that "The design of service systems should build upon a formal model of services". Therefore, the principled design of service-based applications and systems is identified as a primary research challenge for the coming years.

To tackle this challenge, in [10], two orchestrated and choreographed automata-based models of services, called *contract automata*[1] and *communicating (finite-state) machines*, respectively, are studied and related. The goal of both formalisms is to compose the automata such that each service is capable of reaching an accepting (final) state by synchronous/asynchronous one-to-one interactions with the other services in the composition. The main difference relies on the fact that contract automata are oblivious of their partners and an orchestration is synthesised to drive their interactions, whereas communicating machines name the recipient service of each interaction upfront and use FIFO buffers to interact with each other. The model of contract automata was further developed in [8].

The orchestration synthesis was borrowed from the synthesis of the most permissive controller (mpc) from Supervisory Control Theory (SCT) [43, 18], whose aim is to coordinate an ensemble of (local) components into a (global) system that functions correctly. In the context of contract automata, this amounts to refining the composition of service contracts into its largest sub-portion whose behaviour is non-blocking and safe (a notion of service compliance). The adaptation of the mpc synthesis for synthesising an orchestration of services required the introduction of a novel notion of *semi-controllability*. Basically, the assumption of the presence of an unpredictable environment was dropped in favour of a milder notion of predictable necessary service requests to be fulfilled.

[1]Not to be confused with the accidentally homonymous contract automata of [6], which were introduced to formalise legal contracts among two parties expressed in natural language.

In this paper, we contribute to the research efforts on rigorously modelling service orchestration and choreography. More specifically, building on [14], we report on the efforts to relate the mpc synthesis and the orchestration synthesis of contract automata through a polished, homogeneous formalisation. The need for semi-controllability is showcased with intuitive examples and its expressiveness is evaluated with respect to standard SCT notions of controllable and uncontrollable actions. Moreover, we introduce a novel choreography synthesis algorithm and a novel abstract synthesis algorithm. We then show that each of the three synthesis algorithms can be obtained through a different instantiation of this abstract synthesis algorithm. This paper extends [14] in several ways. We include all proofs and as an additional contribution we demonstrate that the different instantiations of the abstract synthesis algorithm are related through a notion of refinement, which allows us to formally prove that the orchestration synthesis is an abstraction of the mpc synthesis. Furthermore, we illustrate each of the synthesis algorithms through a running example from the service domain. Finally, we have also extended the prototypical tool FMCAT[2] with the implementation of the novel choreography synthesis algorithm and then used it to compute all the automata compositions and syntheses shown in our running example.

The paper is organised as follows. Section 2 contains background notions and results concerning contract automata and SCT, and introduces our running example. Section 3 and Section 4 introduce the synthesis of orchestrations and the novel synthesis of choreographies in the setting of (modal service) contract automata. Section 5 demonstrates that each of the introduced synthesis algorithms is an instantiation of a more abstract, parametric synthesis algorithm, and Section 6 shows that these different instantiations are related. Section 7 discusses related work, while Section 8 concludes the paper and provides some hints for future work. Appendix A contains the full proofs of two results only sketched in Section 5.

2. Background

In this section, we provide some background useful to better appreciate our contributions on the crossroads of supervisory control theory and coordination of services formalised as modal service contract automata. We also introduce a running example from the service domain that will be used throughout the paper to illustrate our contributions.

2.1. **Contract Automata.**

A Contract Automaton (CA) represents either a single service (in which case it is called a *principal*) or a multi-party composition of services performing actions. The number of principals of a CA is called its *rank*. The states of a CA are vectors of states of principals. In the following, $\vec{v}$ denotes a vector and $\vec{v}_{(i)}$ denotes its ith element.

The transitions of CA are labelled with actions, which are vectors of elements in the finite set of *basic actions* $\mathsf{L} = \mathsf{R} \cup \mathsf{O} \cup \{\bullet\}$, with $\mathsf{R} \cap \mathsf{O} = \emptyset$ and $\bullet \notin \mathsf{R} \cup \mathsf{O}$. Intuitively, R is the set of *requests* (depicted as non-overlined labels on arcs, e.g. a), O is the set of *offers* (depicted as overlined labels on arcs, e.g. $\overline{a}$) with $\mathsf{O} = \{\, \overline{a} \mid a \in \mathsf{R} \,\}$, and $\bullet$ is a distinguished symbol representing the *idle* action. To establish if a pair of a request and an offer are *complementary*, we use the *involution* function $co : \mathsf{L} \to \mathsf{L}$ defined as follows: $\forall a \in \mathsf{R} : co(a) = \overline{a}$, $\forall \overline{a} \in \mathsf{O} : co(\overline{a}) = a$, and $co(\bullet) = \bullet$. By abusing notation, we let $co(\mathsf{R}) = \mathsf{O}$ and $co(\mathsf{O}) = \mathsf{R}$.

[2]FMCAT is available at `https://github.com/davidebasile/FMCAT`. A video-tutorial showcasing the specification, composition, and syntheses of the contract automata from our running example is available at `https://github.com/davidebasile/FMCAT/tree/master/demoLMCS2020`.

An *action* is a vector $\vec{a}$ of basic actions with either a single offer, or a single request, or a single pair of request-offer that match, i.e. there exist i and j such that $\vec{a}_{(i)}$ is an offer and $\vec{a}_{(j)}$ is the complementary request (formally $co(\vec{a}_{(i)}) = \vec{a}_{(j)}$); all other elements of the vector are $\bullet$, meaning that the corresponding principals remain idle. Such action is called *request*, *offer*, or *match*, respectively. A transition is said to be a request, offer, or match according to its labelling action.

The goal of each principal is to reach an accepting (*final*) state such that all its requests and offers are matched.

In [11], CA are equipped with *modalities*, i.e. *permitted* ($\Diamond$) and *necessary* ($\Box$), that are associated to requests. Offers remain without modalities, i.e. they are interpreted as always permitted, like in the original CA formalism. Matches, on the other hand, inherit the modality of the involved request. The resulting formalism, called Modal Service Contract Automata (MSCA), is formally defined next. Differently from standard SCT, all transitions of MSCA are *observable*, since MSCA model the execution of services in terms of their requests and offers.

Definition 2.1 (MSCA [11])**.** Given a finite set of states $\mathcal{Q} = \{q_1, q_2, \ldots\}$, a *Modal Service Contract Automata (MSCA)* $\mathcal{A}$ of rank n is a septuple $\langle Q, \vec{q_0}, A^{\Diamond}, A^{\Box}, A^o, T, F\rangle$, with set of states $Q \subseteq \mathcal{Q}^n$, initial state $\vec{q_0} \in Q$, set of permitted requests $A^{\Diamond}$ and of necessary request $A^{\Box}$ partitioning the 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 *permitted* transitions $T^{\Diamond}$ and *necessary* transitions $T^{\Box}$, such that: (i) given $t = (\vec{q}, \vec{a}, \vec{q}') \in T$, $\vec{a}$ is either a request, or an offer, or a match; (ii) $\forall i \in 1 \ldots n,\ \vec{a}_{(i)} = \bullet$ implies $\vec{q}_{(i)} = \vec{q}'_{(i)}$; (iii) $t \in T^{\Diamond}$ if and only if $\vec{a}$ is either a request, or a match on $a \in A^{\Diamond}$, or an offer on $\overline{a} \in A^o$; otherwise $t \in T^{\Box}$.

Remarkably, it follows that the set of transitions of an MSCA is finite.

A *principal* is an MSCA of rank 1 such that $A^r \cap co(A^o) = \emptyset$. Unless stated differently, we assume that it is given an MSCA $\mathcal{A} = \langle Q_{\mathcal{A}}, \vec{q_0}_{\mathcal{A}}, A^{\Diamond}_{\mathcal{A}}, A^{\Box}_{\mathcal{A}}, A^o_{\mathcal{A}}, T_{\mathcal{A}}, F_{\mathcal{A}}\rangle$ of rank n. Subscript $\mathcal{A}$ may be omitted if no confusion may arise.

A *step* $(w, \vec{q}) \xrightarrow{\vec{a}} (w', \vec{q}')$ occurs in $\mathcal{A}$ if and only if $w = \vec{a}w'$, $w' \in A^*$, and $(\vec{q}, \vec{a}, \vec{q}') \in T$. Let $\rightarrow^*$ be the reflexive and transitive closure of $\rightarrow$. The *language* of $\mathcal{A}$ is $\mathcal{L}(\mathcal{A}) = \{\, w \mid (w, \vec{q_0}) \xrightarrow{w}{}^*(\varepsilon, \vec{q}),\ \vec{q} \in F \,\}$. A step may be denoted as $\vec{q} \xrightarrow{\vec{a}}$ if w, w', and $\vec{q}'$ are irrelevant, and as $\vec{q} \rightarrow \vec{q}'$ if w, w', and $\vec{a}$ are irrelevant.

Composition of services is rendered through the composition of their MSCA models. This amounts to interleaving or matching the transitions of the component MSCA, forcing the match whenever two components are ready on their respective complementary request/offer actions. In the resulting MSCA, states and actions are vectors of states and actions of the component MSCA, respectively. The composition is non-associative, i.e. pre-existing matches are not rearranged if a new MSCA joins the composition afterwards.

In a composition of MSCA, typically various properties are analysed. We are especially interested in *agreement* and *strong agreement* (which in the literature is also known as progress of interactions, deadlock freedom, compliance or conformance of contracts). In an MSCA in strong agreement, all requests and offers must be matched. Instead, the property of agreement only requires matching all requests. An MSCA admits (strong) agreement if it has a trace satisfying the corresponding property, and it is *safe* if all its traces are such.

The MSCA formalism has its origins in [8], where CA were first introduced, but in this paper we build on the version with modalities from [11] to cater for controllable and

Figure 1: MSCA Client (left) and PrivilegedClient (right)

Figure 2: MSCA Hotel (left) and PrivilegedHotel (right)

uncontrollable (and thus semi-controllable) actions. The branching condition for CA from [10] will be recalled in Section 4 as a condition for obtaining a choreography from an orchestration, and it is satisfied by construction by the output MSCA of the synthesis of the choreography.

Example 2.2. We introduce a running example that will be used throughout the paper to showcase the synthesis of orchestration and choreography. We anticipate, as discussed in detail in Section 4, that a modified version of MSCA is used for the synthesis of a choreography, in which offers can be necessary whilst requests are only permitted.

Figures 1, 3, and 2 show five MSCA of rank 1. These automata model an example of a hotel booking service, where clients and hotels interact by means of a broker for booking hotel rooms. There are two types of clients, `Client` and `PrivilegedClient`. Both clients can either terminate without interactions (final states are drawn as double circles), or they can engage in interactions with the broker to possibly book a room. The first interaction is to ask for a room, by means of the offer $\overline{qry}$ (query). After this action, the clients receive the best room option from the broker, by means of the request *bst* (best). Then, each client can either decide to accept (offer $\overline{ok}$) or refuse (offer $\overline{nok}$) the option offered by the broker. `PrivilegedClient` will be used to showcase a choreography in Section 4. Accordingly, `PrivilegedClient` only differs from `Client` with respect to the first offer $\overline{qry}$, which is declared *necessary*. Basically, `PrivilegedClient` reaches an agreement only if there exists a trace in which its offer is necessarily matched. All other actions are permitted.

Similarly, there are two types of hotels, `Hotel` and `PrivilegedHotel`. Also both hotels can either terminate without interactions, or they can engage in interactions with the broker to possibly have their rooms booked. The first interaction is to receive a request for a room, by means of the request *chk* (check). After this check, a response is sent to the broker through the offer $\overline{rsp}$ (response). Then, each hotel can either receive a booking or a no booking reply by means of requests *bk* (book) or *nbk* (no book), respectively. `PrivilegedHotel` will be used to showcase an orchestration in Section 3. Accordingly, `PrivilegedHotel` only differs from `Hotel` with respect to the request *bk*, declared *necessary*. Basically, `PrivilegedHotel` admits non-empty orchestrations only if there *exists* a trace in which one of its rooms is booked (i.e. the necessary request is matched). All other actions are permitted.

Finally, the `Broker` acts as an intermediary between a client and at least two hotels. The broker starts by receiving a request for a room by a client through the request *qry*. At this point, it starts to interact with the hotels to search for a possible option to propose to the client. This is done by (twice) repeating the offer $\overline{chk}$ (sending a room enquiry) followed

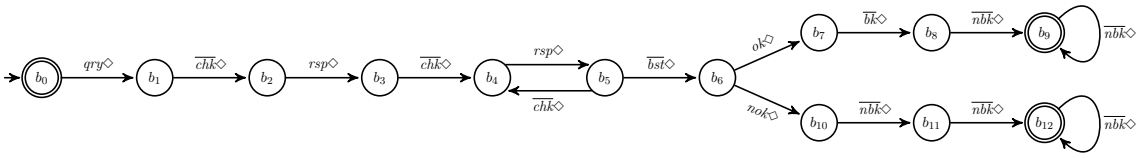

Figure 3: MSCA Broker

by the request *rsp* (receiving the room response by one hotel). Indeed, at least two hotels must be enquired to speak of a best offer. After that, the `Broker` can engage with further hotels, from state b_5, or it can proceed with the best offer $\overline{bst}$ to the client. At this point, it receives through the requests *ok* or *nok* either the acceptance or rejection, respectively, of its offer. If the offer is accepted, `Broker` proceeds to book the room with offer $\overline{bk}$ to the selected hotel (abstracted away in the contract) and replying to all other hotels with a $\overline{nbk}$ offer. Otherwise, if the offer is rejected, `Broker` sends to all hotels waiting for a reply the offer $\overline{nbk}$. All actions of `Broker` are permitted.

2.2. **Supervisory Control Theory.** The aim of Supervisory Control Theory [43, 18] (SCT) is to provide an algorithm to synthesise a finite-state automaton model of a *supervisory controller* from given (component) finite-state automata models of the uncontrolled system and its requirements, themselves expressed as automata. The synthesised supervisory controller, if successfully generated, is such that the controlled system, which is the composition (i.e. synchronous product) of the uncontrolled system and the supervisory controller, satisfies the requirements and is additionally *non-blocking*, *controllable*, and *maximally permissive*.

An automaton is *non-blocking* if from each state at least one of the so-called *marked states* (distinguished stable states representing completed 'tasks' [43], e.g. a final state) can be reached without passing through so-called *forbidden states*, meaning that the system always has the possibility to return to an accepted stable state. The algorithm assumes that marked states and forbidden states are indicated for each component model.

SCT distinguishes between *observable* and *unobservable*, as well as *controllable* and *uncontrollable* actions, where unobservable actions are also uncontrollable. Intuitively, the supervisory controller cannot distinguish one unobservable action from the other, whereas it can take observable actions apart. Moreover, it is not permitted to directly block uncontrollable actions from occurring; the controller is only allowed to disable them by preventing controllable actions from occurring. Intuitively, controllable actions correspond to stimulating the system, while uncontrollable actions correspond to messages provided by the environment, like sensors, which may be neglected but cannot be denied from existing.

Finally, the fact that the resulting supervisory controller is *maximally permissive* (or least restrictive) means that as much behaviour of the uncontrolled system as possible remains present in the controlled system without violating neither the requirements, nor controllability, nor the non-blocking condition.

From the seminal work of Ramadge and Wonham [43], we know that a unique maximally permissive supervisory controller exists, provided that all actions are observable. This is called the *most permissive controller* (*mpc*); it coordinates an ensemble of (local) components into a (global) system that works correctly. The synthesis algorithm suffers from the same

state space explosion problem as model checking [31]. However, SCT has successfully been applied to industrial size case studies [29, 49].

Intuitively, the synthesis algorithm for computing the mpc of a finite-state automaton $\mathcal{A}$ works as follows. The mpc is computed through an iterative procedure that at each step i updates incrementally a set of states R_i containing the *bad* states, i.e. those states that cannot prevent a forbidden state to be eventually reached, and refines an automaton $\mathcal{K}_i$.

The algorithm starts with an automaton $\mathcal{K}_0$ equal to $\mathcal{A}$ and a set R_0 containing all *dangling* states in $\mathcal{A}$, where a state is dangling if it cannot be reached from the initial state or cannot reach a final state. At each step i, the algorithm prunes from $\mathcal{K}_{i-1}$ in a backwards fashion transitions with target state in R_{i-1} or forbidden source state. The set R_i is obtained by adding to R_{i-1} dangling states in $\mathcal{K}_i$ and source states of uncontrollable transitions of $\mathcal{A}$ with target state in R_{i-1}. When no more updates are possible, the algorithm terminates. Termination is ensured since $\mathcal{A}$ is finite-state and has a finite set of transitions, and at each step the subsets of its states R_i cannot decrease while the set of its transitions $T_{\mathcal{K}_i}$ cannot increase. Now, suppose that at its termination the algorithm returns the pair $(\mathcal{K}_s, R_s)$. We have that the mpc is empty, if the initial state of $\mathcal{A}$ is in R_s; otherwise, the mpc is obtained from $\mathcal{K}_s$ by removing the states R_s.

We report below the standard synthesis algorithm, but we homogenise the notation and simplify the formulation, to align the algorithm with those presented in the next sections. For this purpose, we assume the standard mpc synthesis to operate on MSCA where necessary transitions ($T^{\Box}$) are uncontrollable whilst permitted transitions ($T^{\Diamond}$) are controllable.

We use $\langle\ \rangle$ to denote the empty automaton. A state $q \in Q$ is said to be *dangling* if and only if $\nexists\, w$ such that $q_0 \xrightarrow{w}{}^{*} q$ or $q \xrightarrow{w}{}^{*} q_f \in F$. Let *Dangling*($\mathcal{A}$) denote the set of dangling states of $\mathcal{A}$. Given two MSCA $\mathcal{A}$ and $\mathcal{A}'$, we say that $\mathcal{A}'$ is a *sub-automaton* of $\mathcal{A}$, denoted by $\mathcal{A}' \subseteq \mathcal{A}$, whenever the components of $\mathcal{A}'$ are included in the corresponding ones of $\mathcal{A}$. Moreover, given two sets of states R and R', we let $(\mathcal{A}, R) \leq (\mathcal{A}', R')$ if $\mathcal{A}' \subseteq \mathcal{A}$ and $R \subseteq R'$. It is straightforward to show that $(\mathit{MSCA} \times 2^{Q}, \leq)$ is a complete partial order (cpo).

The algorithm to compute the mpc is now defined in terms of the *least fixed point* of a monotone function on the cpo $(\mathit{MSCA} \times 2^{Q}, \leq)$.

Definition 2.3 (Standard synthesis, adapted from [43])**.** Let $\mathcal{A}$ be an MSCA, and let $\mathcal{K}_0 = \mathcal{A}$ and $R_0 = \mathit{Dangling}(\mathcal{K}_0)$. We let the *synthesis function* $f : \mathit{MSCA} \times 2^{Q} \to \mathit{MSCA} \times 2^{Q}$ be defined as follows:

$$
\begin{aligned}
f(\mathcal{K}_{i-1}, R_{i-1}) &= (\mathcal{K}_i, R_i), \text{ with} \\
T_{\mathcal{K}_i} &= T_{\mathcal{K}_{i-1}} \setminus \{\, (\vec{q} \to \vec{q}\,') \in T_{\mathcal{K}_{i-1}} \mid \vec{q}\,' \in R_{i-1} \vee \vec{q} \text{ is forbidden} \,\} \\
R_i &= R_{i-1} \cup \{\, \vec{q} \mid (\vec{q} \to \vec{q}\,') \in T^{\Box}_{\mathcal{A}},\ \vec{q}\,' \in R_{i-1} \,\} \cup \mathit{Dangling}(\mathcal{K}_i)
\end{aligned}
$$

Theorem 2.4 (Standard mpc, adapted from [43])**.** *The synthesis function f is monotone on the cpo $(\mathit{MSCA} \times 2^{Q}, \leq)$ and its* least fixed point *is:*

$$(\mathcal{K}_s, R_s) = \sup(\{\, f^n(\mathcal{K}_0, R_0) \mid n \in \mathbb{N} \,\})$$

The mpc *of $\mathcal{A}$, denoted by $\mathcal{K}_{\mathcal{A}}$, is:*

$$\mathcal{K}_{\mathcal{A}} = \begin{cases} \langle\ \rangle & \text{if } \vec{q}_0 \in R_s \\ \langle Q \setminus R_s, \vec{q}_0, A^{\Diamond}, A^{\Box}, A^{o}, T_{\mathcal{K}_s}, F \setminus R_s \rangle & \text{otherwise} \end{cases}$$

We now want to estimate an upper bound of the complexity of the mpc synthesis algorithm as results from Definition 2.3 and Theorem 2.4. In the worst case, deciding if a state is dangling requires to visit the whole state space. Thus, an upper bound of the

complexity of the procedure for deciding if a state is dangling is $\mathcal{O}(|Q|)$, and the upper-bound complexity for computing the set of dangling states is $\mathcal{O}(|Q|^2)$. At each iteration, in the worst case, the algorithm either removes a single transition from T or adds a single state to R, and each iteration requires to compute the set of dangling states. Thus, an upper bound of the complexity of the mpc synthesis algorithm is $\mathcal{O}((|T|+|Q|)\times|Q|^2)$. To conclude, it is worth noticing that our analysis focusses on the abstract specification of the algorithm while its implementation could be optimised, for example by using parallelism and dedicated data structures, in order to perform better than the complexity sketched above.

Example 2.5. We continue the running example by discussing the synthesis of the mpc for the composition of two clients, the broker, one normal hotel, and one privileged hotel, denoted as

$$\mathcal{A}_1 = \mathsf{Client} \otimes \mathsf{Client} \otimes \mathsf{Broker} \otimes \mathsf{Hotel} \otimes \mathsf{PrivilegedHotel}$$

The property to be enforced is agreement: each request must be matched by a corresponding offer. Basically, this property is an invariant stating that all request transitions are forbidden. Since the synthesis works on forbidden states, we need to preprocess $\mathcal{A}_1$ accordingly. In particular, the algorithm starts from the automaton $\mathcal{A}_1$ from which all permitted requests have been removed. Forbidden states are those featuring an outgoing *necessary* request.

The resulting mpc only consists of the initial (and final) state $(c_0, c_0', b_0, h_0, h_0')$, and its behaviour is empty. Hence, agreement cannot be enforced in $\mathcal{A}_1$ using the standard synthesis algorithm. This is an indication of the fact that standard mpc synthesis is not useful for the scope of synthesising a correct service composition (i.e. in which agreement is satisfied). The reason is that necessary transitions are not to be interpreted as uncontrollable. The notion of uncontrollable transition stems from the necessity of modelling an unpredictable environment, which is not suitable to model necessary service requests. Basically, `PrivilegedHotel` has a necessary request that should be matched in *at least* one trace of the composition. However, by interpreting such necessary request as uncontrollable, the synthesis is enforcing the necessary requests to be satisfied in *every* trace of the composition. Intuitively, this would require that a client is not allowed to refuse to book a room.

As will become clear in the forthcoming sections, $\mathcal{A}_1$ admits a non-empty orchestration in which agreement is enforced, because necessary transitions will not be interpreted as fully uncontrollable.

We have used our tool FMCAT to calculate the automaton $\mathcal{A}_1$ and its mpc synthesis. Their computation time and state-space dimension are reported in Table 1 (on page 17).

3. Synthesis of Orchestrations

In this section, we discuss how we revised the classical synthesis algorithm from SCT to obtain the mpc (cf. Theorem 2.4) and synthesise orchestrations of MSCA.

Originally, MSCA were capable of expressing only permitted requirements, corresponding to actions that are controllable by the orchestrator. Hence, in the synthesis of the orchestration, all transitions labelled by actions violating the property to be enforced were pruned, and all dangling states were removed (cf. [8]).

While permitted requests of MSCA are in one-to-one correspondence with controllable actions, interestingly this is not the case for necessary requests and uncontrollable actions. A necessary (request) action is indeed a weaker constraint than an uncontrollable one. This stems from the fact that traditionally uncontrollable actions relate to an unpredictable

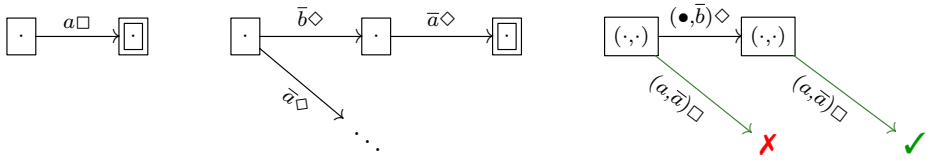

Figure 4: Two MSCA (left and middle) and a possible composition $\mathcal{A}$ of them (right)

environment. However, the interpretation of such actions as *necessary* service requests to be fulfilled in a service contract, as is the case in the setting of MSCA, implies that it suffices that in the synthesised orchestration at least one such synchronisation (i.e. match) actually occurs. This is precisely what is modelled by the notion of *semi-controllable* actions, anticipated in [11] and formally introduced in [12, 13], discussed next.

The importance of this novel notion in the synthesis algorithm is showcased by an intuitive example. Consider the two MSCA interacting on the necessary service request a depicted in Fig. 4 (left and middle), and their possible composition $\mathcal{A}$ depicted in Fig. 4 (right). Note that $\mathcal{A}$ models two possibilities of fulfilling request a from the leftmost automaton by matching it with a service offer $\overline{a}$ from the middle one. Note that a similar composition can be obtained in other automata-based formalisms (such as, e.g., (timed) I/O automata [39, 2, 25]). Now assume that a must be matched with $\overline{a}$ to obtain an agreement (i.e. it is *necessary*), and that for some reason the *bad* state ✗ is to be avoided in favour of the *successful* state ✓, i.e. in some sense we would like to express that a must be matched at some point, rather than always. In most automata-based formalisms this is not allowed and the resulting mpc is empty. In the MSCA formalism, it is possible to orchestrate the composition of the two automata on the left in such a way that the result is the automaton $\mathcal{A}$ on the right, but *without the state* ✗ and its incident transition.

In fact, in the MSCA formalism, $\mathcal{A}$ depicts a composition in which the automata on the left can synchronise on a so-called semi-controllable action $a\Box$ either in their initial state or after the middle automaton has performed some other action $\overline{b}\Diamond$, ignoring in this case whether a bad or a successful state is reached in the end. Indeed, the notion of semi-controllability is independent from both the specific formalism being used and the requirement (e.g. agreement in case of MSCA) to be enforced.

As far as we know, we were the first to define a synthesis algorithm, in [13], that is capable of producing a controller that guarantees that *at least* one of these two synchronisations actually occurs. Indeed, in the standard synthesis algorithm (cf. Theorem 2.4), action a can either be *controllable* and hence not necessary as we want, or *uncontrollable* thus requiring that a must *always* be matched, a stronger requirement than the one posed by declaring a as necessary.

To formalise the intuitions above[3], a semi-controllable transition t becomes controllable if 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* principal, in the *same* local state, does the *same* request. Otherwise, t is uncontrollable.

Definition 3.1 (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 $\vec{a}_1$ is an action on $a \in A^{\Diamond} \cup A^{o}$, then t is *controllable* (in $\mathcal{A}$) and part of $T^{\Diamond}$;
- if $\vec{a}_1$ is a request or match on $a \in A^{\Box}$, then t is *semi-controllable* (in $\mathcal{A}$) and part of $T^{\Box}$.

[3]We refer the interested reader to [12, 13] for more complete accounts.

Moreover, given $\mathcal{A}' \subseteq \mathcal{A}$, if t is semi-controllable and $\exists\, t' = (\vec{q_2}, \vec{a_2}, \vec{q_2'}) \in T^{\Box}_{\mathcal{A}'}$ 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 $\vec{a}_{1(i)} = \vec{a}_{2(i)} = a$, then t is *controllable* in $\mathcal{A}'$ (via t'); otherwise, t is *uncontrollable* in $\mathcal{A}'$.

The algorithm for synthesising an orchestration enforcing agreement of MSCA follows. The main adaptation of the mpc synthesis of Theorem 2.4 is that transitions are no longer declared uncontrollable, but instead they can be either controllable or semi-controllable. More importantly, a semi-controllable transition switches from controllable to uncontrollable only after it has been pruned in a previous iteration, in which case its source state becomes bad. Finally, in this case there are no forbidden states but rather forbidden transitions (i.e. requests, according to the property of agreement).

Definition 3.2 (MSCA orchestration synthesis, adapted from [11])**.** Let $\mathcal{A}$ be an MSCA, and let $\mathcal{K}_0 = \mathcal{A}$ and $R_0 = \mathit{Dangling}(\mathcal{K}_0)$. We let the *orchestration synthesis function* $f_o : \mathit{MSCA} \times 2^Q \to \mathit{MSCA} \times 2^Q$ be defined as follows:

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

Theorem 3.3 (MSCA orchestration, adapted from [11])**.** *The orchestration synthesis function f_o is monotone on the cpo $(\mathit{MSCA} \times 2^Q, \leq)$ and its* least fixed point *is:*

$$(\mathcal{K}_s, R_s) = \sup(\{\, f_o^n(\mathcal{K}_0, R_0) \mid n \in \mathbb{N} \,\})$$

The orchestration $\mathcal{K}_{\mathcal{A}}$ *of* $\mathcal{A}$ *is:*

$$
\mathcal{K}_{\mathcal{A}} = \begin{cases} \langle\, \rangle & \textit{if } \vec{q_0} \in R_s \\ \langle Q \setminus R_s, \vec{q_0}, A^{\Diamond}, A^{\Box}, A^o, T_{\mathcal{K}_s} \setminus T', F \setminus R_s \rangle & \textit{otherwise} \end{cases}
$$

where $T' = \{\, t = \vec{q} \to \in \mathcal{K}_s \mid t \textit{ is controllable in } \mathcal{K}_s,\ \vec{q} \in R_s \,\}$.

We now estimate the complexity of the orchestration synthesis algorithm. In the synthesis of the mpc, deciding whether a transition is controllable or uncontrollable has a complexity of $\mathcal{O}(1)$. On the converse, for the orchestration case, deciding whether a semi-controllable transition is controllable or uncontrollable requires in the worst case to check all transitions of the automaton. Accordingly, the procedure for computing the set of uncontrollable transitions has an upper-bound complexity of $\mathcal{O}(|T|^2)$. Since this is the only difference with respect to the mpc synthesis, a first upper bound of the complexity of the orchestration synthesis is $\mathcal{O}((|T| + |Q|) \times |Q|^2 \times |T|^2)$. The computation of the set of dangling states and uncontrollable transitions could be done in parallel through a single visit of the automaton. Thus, the upper-bound complexity of the orchestration synthesis can be lowered to be the same as the complexity of the mpc synthesis, i.e. $\mathcal{O}((|T| + |Q|) \times |Q|^2)$. Finally, we want to underline that our complexity estimation refers to the abstract specification of the algorithm, resulting from Definition 3.2 and Theorem 3.3. As already observed for the mpc synthesis, when implementing the algorithm further optimisations could be achieved that can lower its complexity.

Example 3.4. We further continue the running example by discussing the synthesis of the orchestration for the composition

$$\mathcal{A}_1 = \mathsf{Client} \otimes \mathsf{Client} \otimes \mathsf{Broker} \otimes \mathsf{Hotel} \otimes \mathsf{PrivilegedHotel}$$

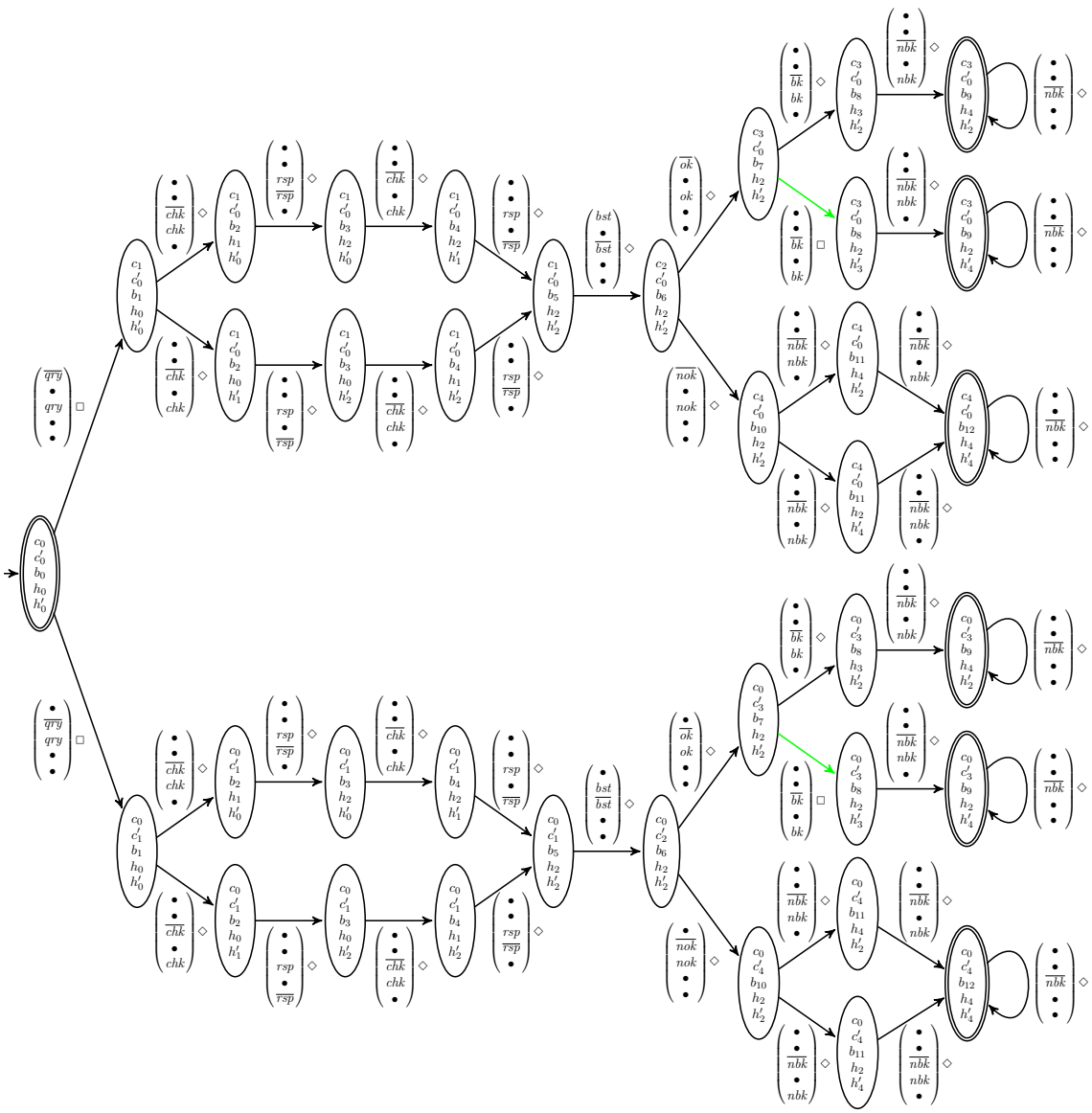

Figure 5: Orchestration of Client ⊗ Client ⊗ Broker ⊗ Hotel ⊗ PrivilegedHotel

The orchestration of $\mathcal{A}_1$ is depicted in Fig. 5 and the time needed to compute it by using FMCAT is reported in Table 1 (on page 17). We recall that the orchestration is the largest sub-portion of the composition that is in agreement, i.e. in which requests are matched by offers.

From the initial (and final) state there are two possible evolutions: either one of the clients is served while the other one does not interact. Without loss of generality, assume that the first client is served. The orchestration continues with the broker enquiring the hotels (in both possible orders). After these enquiries, the reached state is $\vec{q} = (c_1, c'_0, b_5, h_2, h'_2)$. From $\vec{q}$, the broker sends the best offer received from one of the hotels to the client, and the client decides whether or not to accept this best offer. The broker then communicates the selected

choice to the hotels it interacted with. Note that in the orchestration it is possible that the client does not book any room.

We now explain why the mpc is empty (cf. Example 2.5). First, note that $\vec{q}$ must be traversed to reach a final successful state. The composition $\mathcal{A}_1$ (which is not displayed for space limitations) contains the transition $t = \vec{q} \xrightarrow{(\bullet,\bullet,\bullet,\bullet,bk)\Box} (c_1, c'_0, b_5, h_2, h'_3)$. The state $\vec{q}$ is then forbidden, since its outgoing transition t is uncontrollable and cannot be pruned. It follows that all states traversed from the initial state to $\vec{q}$ would eventually become dangling during the mpc synthesis, and thus the mpc is empty.

On the converse, for the case of synthesising the orchestration, we have that t is *semi-controllable* and it is *controllable* via $t' = (c_3, c'_0, b_7, h_2, h'_2) \xrightarrow{(\bullet,\bullet,\overline{bk},\bullet,bk)\Box} (c_1, c'_0, b_8, h_2, h'_3)$. Thus, t is pruned by the orchestration synthesis algorithm. Intuitively, in the orchestration there exists a trace in which the necessary request bk is matched.

This example shows that the notion of semi-controllability is best suited for necessary requests of service contracts. We argue that semi-controllability is not specific to the context of service contracts; rather it is independent of the used formalism and can be applied in other contexts as well. Semi-controllability can be interpreted as the 'existentially quantified' counterpart of the universally quantified notion of uncontrollability, originally stemming from Supervisory Control Theory, in much the same way that Computation Tree Logic allows existential quantification of paths that can only be universally quantified in Linear Temporal Logic.

However, in the next section we will see that the notion of semi-controllability is too relaxed for the case of choreography, which thus demands a revisited version.

3.1. **On encoding semi-controllability.** We now show, by means of an example adapted from [13], that the encoding of an automaton $\mathcal{A}$ with semi-controllable actions into an automaton $\mathcal{A}'$ without, such that the same synthesised orchestrations are obtained, results in an exponential blow-up of the state space. More precisely, the encoding is intended to preserve safety: the orchestration of $\mathcal{A}$ equals that of $\mathcal{A}'$.

The encoding is sketched in Fig. 6. Intuitively, the encoded automaton $\mathcal{A}'$ is obtained by first applying the following construction to the automaton $\mathcal{A}$ from Fig. 4 (right):

> if the synchronisation on a specific semi-controllable action a occurs in n different transitions in $\mathcal{A}$ (two in our example), then the encoding creates an automaton $\mathcal{A}'$ that is the union of $2^n - 1$ automata (three in our example), which are obtained by all possible combinations of pruning a subset of the n semi-controllable transitions of $\mathcal{A}$, minus the one in which all n semi-controllable transitions are pruned;

and then turning all semi-controllable transitions into uncontrollable transitions.

We now explain why, without knowing a priori the set of forbidden and successful states, it is impossible to provide a more efficient encoding and refer to [13, Theorems 3 and 4] for a formal account. Assume, by contradiction, that there exists an encoding that results in a 'smaller' automaton $\mathcal{A}''$, in which one of the $2^n - 1$ combinations of pruned transitions (say, P) is discarded. It then suffices to specify as a counterexample a property in $\mathcal{A}$ such that all source states of transitions in P are forbidden and all target states of the remaining semi-controllable transitions are successful. The synthesis of $\mathcal{A}$ against such a property would prune exactly the semi-controllable transitions in P. However, in the synthesis of $\mathcal{A}''$ such an orchestration would not be present, a contradiction.

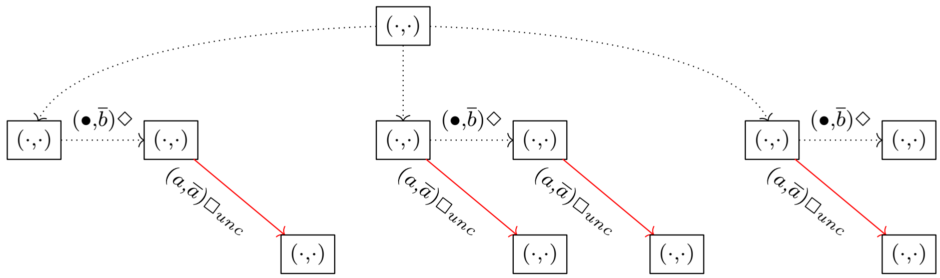

Figure 6: Automaton $\mathcal{A}'$ uses uncontrollable transitions to encode automaton $\mathcal{A}$ from Fig. 4

4. Synthesis of Choreographies

In the previous section, we have seen that the orchestration of MSCA is similar to a most permissive controller. The orchestrator is however implicit, in the sense that its interactions with the principals are hidden. Basically, one could assume that before interacting, each principal expects a message from the orchestrator and answers with an acknowledgement after the interaction terminates. The main intuition behind switching from an orchestrated to a choreographic coordination of contracts is that there is no longer the need for such 'hidden' interactions. Ideally, the principals moving autonomously are able to accomplish the behaviour foreseen by the synthesis, which in this case acts as a global type. Differently from the traditional choreographic approach, where the starting point is a global type, in MSCA the global type is synthesised automatically.

The requirements for ensuring that the synthesised automaton is a (form of) choreography were studied in [10, 37]. Roughly, they amount to the so-called *branching condition* requiring that principals perform their offers/outputs independently of the other principals in the composition. To formalise this notion, we let $snd(\vec{a}) = i$ when $\vec{a}$ is a match action or an offer action and $\vec{a}_{(i)} \in \mathsf{O}$.

Definition 4.1 (Branching condition [10])**.** An MSCA $\mathcal{A}$ satisfies the *branching condition* if and only if the following holds for each pair of states $\vec{q}_1$, $\vec{q}_2$ reachable in $\mathcal{A}$:

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

The branching condition is related to a phenomenon known as 'state sharing' in other coordination models (cf., e.g., [45]) according to which system components can influence potential synchronisations through their local (component) states even if they are not involved in the actual global (system) transition.

In [10], it is proved that the synthesised automaton corresponds to a well-behaving choreography if and only if it satisfies the branching condition and is strongly safe. Notably, in case the two conditions are not satisfied, that paper does not provide any algorithm for automatically synthesising a choreography; rather, the contracts have to be manually amended. Instead, in the remainder of this section, we introduce a novel algorithm for automatically synthesising a well-behaving choreography. Note that, differently from the orchestration and the controller synthesis, in this case there could be more than one possible choreography (cf. Example 4.6).

The property to be enforced during the synthesis is strong agreement: all offers and requests have to be matched, because all messages have to be read (i.e. offers matched). Moreover, in the case of choreography, service contract requests are always permitted whereas

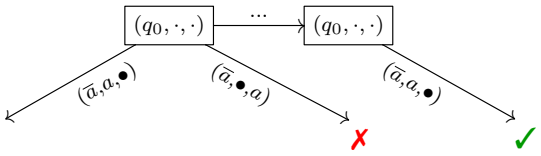

Figure 7: Fragment of a possible service composition

service contract offers can be necessary. That is, the roles of service requests and offers are swapped with respect to the case of orchestration.

In principle, the synthesis could trivially introduce a coordinator component and its interactions to coordinate the principals. However, this would reduce the choreography to a centralised coordination of contracts. To prevent this, the synthesis can only remove and never add behaviour. Hence, a choreography can only be synthesised if all principals are capable of interacting on their own without resorting to a central coordinator.

Similarly to orchestration synthesis, indicating transitions as either controllable or uncontrollable does not suffice for synthesising a choreography. Moreover, the notion of semi-controllability introduced for the orchestration case does not suffice for expressing necessary offers. Indeed, orchestration synthesis does not ensure the branching condition to be satisfied by the synthesised automaton, as the following example shows.

Example 4.2. In Fig. 7, a fragment of a service composition is shown. Two global states are depicted, and in both the first service, say *Alice*, is in its initial local state (say, q_0). *Alice* performs an output (i.e. offer) $\overline{a}$ that can be directed to either *Bob* (second service) or *Carol* (third service), from the initial global state, or only to *Bob* from the other state. It is possible to reach either a successful (✓) or a bad (✗) state, left unspecified for the moment. Notably, the output of *Alice* is neither controllable, nor uncontrollable, nor semi-controllable by the synthesis.

Now assume that the $\overline{a}$ is controllable and from the initial global state both interactions eventually lead to a bad state (✗). In this case, those transitions are pruned by the synthesis, and the resulting automaton is erroneously approved. Indeed, *Alice* has no mean to understand when her output $\overline{a}$ is enabled, because she has not changed state. The branching condition, which is necessary for obtaining a well-behaving choreography, would be violated. Note that this would happen also if $\overline{a}$ were semi-controllable. In fact, to satisfy the branching condition, the synthesis should remove all outputs $\overline{a}$.

Conversely, assume that the $\overline{a}$ is uncontrollable and that it is possible from the initial global state to reach a successful state (✓) if the message $\overline{a}$ is received by *Bob*. In this case, it would not be possible to prune the transition from the initial state leading to ✗, because it is also uncontrollable. The synthesis would thus be empty, an erroneous rejection, because a choreography exists in which *Alice* autonomously interacts with *Bob*.

In conclusion, a necessary action is rendered neither as uncontrollable nor as semi-controllable, and permitted actions require extra pruning operations during the synthesis. A novel notion of semi-controllability for a necessary action is required, which is weaker than uncontrollable but stronger than the semi-controllable notion used in the synthesis of orchestration.

Basically, for the choreography synthesis, a (semi-controllable) necessary transition $t = (\vec{q} \xrightarrow{\vec{a}_1}) \in T^{\square}$ is detected to be uncontrollable if and only if no necessary transition

$t' = (\vec{q} \xrightarrow{\vec{a}_2}) \in T^{\Box}$ exists from the same source state such that in both t and t' the same offer is provided by the same principal, but possibly with different receivers. We now define this formally.

Definition 4.3. Let $\mathcal{A}$ be an MSCA and let $t = (\vec{q}, \vec{a}_1, \vec{q}_1') \in T_{\mathcal{A}}$. Then:
- if $\vec{a}_1$ is an action on $a \in A^{\Diamond}$, then t is *controllable* (in $\mathcal{A}$);
- if $\vec{a}_1$ is an offer or match on $a \in A^{\Box}$, then t is *semi-controllable* (in $\mathcal{A}$).

Moreover, given $\mathcal{A}' \subseteq \mathcal{A}$, if t is semi-controllable and $\exists\, t' = (\vec{q}, \vec{a}_2, \vec{q}_2') \in T^{\Box}_{\mathcal{A}'}$ such that $\vec{a}_2$ is a match, $\vec{q}, \vec{q}_2' \notin \mathit{Dangling}(\mathcal{A}')$, and $\vec{a}_{1(i)} = \vec{a}_{2(i)}$ where $i = \mathit{snd}(\vec{a}_1)$, then t is *controllable* in $\mathcal{A}'$ (via t'); otherwise, t is *uncontrollable* in $\mathcal{A}'$.

Hence, again a necessary transition is a particular type of transition that switches from being controllable to uncontrollable in case a condition on the global automaton is not met. Note that this condition is stronger than the one required for the case of orchestration (semi-controllability), because for the case of choreography transitions t and t' in Definition 4.3 share the source state. Moreover, also in this case it can be shown that the encoding of this type of semi-controllable transition into an uncontrollable one would result in an exponential growth of the state space of the model.

Similarly to the orchestration synthesis in Definition 3.2, when a semi-controllable transition previously removed by the synthesis switches from controllable to uncontrollable, its source state is detected to be bad. Apart from the different notion of semi-controllability, another difference with respect to the orchestration synthesis is that the transitions violating the branching condition must also be removed. Depending on which transitions violating the branching condition are pruned at a certain iteration, different choreographies can be obtained (cf. Example 4.6). Indeed, a maximal choreography is not always guaranteed to exist (as is the case for the running example). A concrete implementation should fix the criterion under which transitions are selected for the set $\hat{T}_{\mathcal{K}_i,R_i}$ (cf. Definition 4.4).

Finally, according to the property of strong agreement, both request and offer transitions are forbidden. The formalisation is provided next.

Definition 4.4 (MSCA choreography synthesis)**.** Let $\mathcal{A}$ be an MSCA, and let $\mathcal{K}_0 = \mathcal{A}$ and $R_0 = \mathit{Dangling}(\mathcal{K}_0)$. We let a *choreography synthesis function* $f_c : \mathit{MSCA} \times 2^Q \to \mathit{MSCA} \times 2^Q$ be defined as follows:

$$f_c(\mathcal{K}_{i-1}, R_{i-1}) = (\mathcal{K}_i, R_i), \text{ with}$$
$$T_{\mathcal{K}_i} = T_{\mathcal{K}_{i-1}} \setminus (\{\, (\vec{q} \to \vec{q}') = t \in T_{\mathcal{K}_{i-1}} \mid \vec{q}' \in R_{i-1} \vee t \textit{ is a request or offer} \,\} \cup \hat{T}_{\mathcal{K}_{i-1},R_{i-1}})$$
$$R_i = R_{i-1} \cup \{\, \vec{q} \mid (\vec{q} \to) \in T_{\mathcal{A}} \textit{ is uncontrollable in } \mathcal{K}_i \,\} \cup \mathit{Dangling}(\mathcal{K}_i)$$

where, at each iteration i,

$$\hat{T}_{\mathcal{K}_i,R_i} \subseteq T_{bc} = \{\, (\vec{q}_1 \xrightarrow{\vec{a}}) \in T_{\mathcal{K}_i} \mid \exists\, \vec{q}_2 : (\mathit{snd}(\vec{a}) = j \wedge \vec{q}_{1(j)} = \vec{q}_{2(j)}) \wedge (\vec{q}_2 \xrightarrow{\vec{a}}) \notin T_{\mathcal{K}_i} \wedge \vec{q}_1, \vec{q}_2 \notin R_i \,\}$$

and whenever $f_c(\mathcal{K}_i, R_i) = (\mathcal{K}_i, R_i)$ then $T_{bc} = \emptyset$.

Theorem 4.5 (MSCA choreography)**.** *A choreography synthesis function f_c is monotone on the cpo $(\mathit{MSCA} \times 2^Q, \leq)$ and its* least fixed point *is:*

$$(\mathcal{K}_s, R_s) = \sup(\{\, f_c^n(\mathcal{K}_0, R_0) \mid n \in \mathbb{N} \,\})$$

A choreography $\mathcal{K}_{\mathcal{A}}$ *of* $\mathcal{A}$ *is:*

$$\mathcal{K}_{\mathcal{A}} = \begin{cases} \langle\, \rangle & \textit{if } \vec{q}_0 \in R_s \\ \langle Q \setminus R_s, \vec{q}_0, A^{\Diamond}, A^{\Box}, A^o, T_{\mathcal{K}_s} \setminus T', F \setminus R_s \rangle & \textit{otherwise} \end{cases}$$

where $T' = \{\, t = \vec{q} \to \in \mathcal{K}_s \mid t \text{ is controllable in } \mathcal{K}_s,\ \vec{q} \in R_s \,\}$.

Moreover, $\mathcal{K}_{\mathcal{A}}$ *satisfies the branching condition.*

Proof. The algorithm terminates because at each iteration either some transition is pruned or a state becomes forbidden, and both sets of transitions and states are finite. We now prove that the synthesised automaton is (i) non-blocking, (ii) controllable, (iii) strongly safe, and (iv) satisfies the branching condition. In case $\mathcal{K}_{\mathcal{A}} = \langle\ \rangle$, the properties hold trivially, thus we assume that the synthesised controller is non-empty.

For (i), trivially all dangling states are pruned, so it is always possible to reach a final state. Similarly, bad states (i.e. states in the set R_s) are never traversed by construction, i.e. transitions with target in R_s are pruned.

For (ii), by construction all uncontrollable transitions have source state in R_s, and thus are not reachable. Note that by Definition 4.3 uncontrollable transitions are necessary requirements that are not met and thus are always removed by the synthesis.

For (iii), all transitions eventually violating strong safety are requests or offers and are pruned by the synthesis.

For (iv), the transitions violating the branching condition are

$$\{\, (\vec{q}_1 \xrightarrow{\vec{a}}) = t \in T_{\mathcal{K}_{\mathcal{A}}} \mid \exists\, \vec{q_2} : (snd(\vec{a}) = i \wedge \vec{q_1}_{(i)} = \vec{q_2}_{(i)}) \wedge (\vec{q}_2 \xrightarrow{\vec{a}}) \notin T_{\mathcal{K}_{\mathcal{A}}} \,\}$$

and these are pruned by definition. □

Returning to Example 4.2, the erroneously accepted case is removed because, during the synthesis, the operation of pruning the transitions leading to bad states causes the removal of the remaining transition. Thus, the obtained choreography is empty. Similarly, the erroneously rejected case is not possible because, assuming that the output from the initial state is necessary, this necessary action is not rendered as uncontrollable as long as the output is matched by some other principal from the same initial state.

We now estimate also the complexity of the choreography synthesis. With respect to the orchestration synthesis, in the choreography synthesis at each iteration a transition violating the branching condition can be removed. In the worst case, deciding if a transition violates the branching condition requires to check all other transitions. Hence, an upper bound of the procedure for selecting a transition violating the branching condition is $\mathcal{O}(|T|^2)$. Note that, in the unlikely event that all transitions share the same source state, the upper-bound complexity for computing the set of uncontrollable transitions is the same as in the case of orchestration synthesis. Thus, a first upper bound of the complexity of the choreography synthesis algorithm is $\mathcal{O}((|T| + |Q|) \times |Q|^2 \times |T|^4)$. We can refine this first approximation to $\mathcal{O}((|T| + |Q|) \times |Q|^2)$. Indeed, similar to the case of orchestration synthesis, at each iteration in a single traversal of the automaton it is possible to compute the set of dangling states, the set of uncontrollable transitions, and the set of transitions violating the branching condition. Also in this case, as for the other syntheses, our complexity estimation refers to the abstract specification of the algorithm resulting from Definition 4.4 and Theorem 4.5. The implementation of the algorithm could be optimised to perform even better.

Example 4.6. We once more continue the running example by discussing the choreography synthesis of the running example for the composition

$$\mathcal{A}_2 = \mathsf{Client} \otimes \mathsf{PrivilegedClient} \otimes \mathsf{Broker} \otimes \mathsf{Hotel} \otimes \mathsf{Hotel}$$

The choreography of $\mathcal{A}_2$ is depicted in Fig. 8 and the time needed to compute $\mathcal{A}_2$ and its choreography by using FMCAT is reported in Table 1. Note that differently from $\mathcal{A}_1$ in

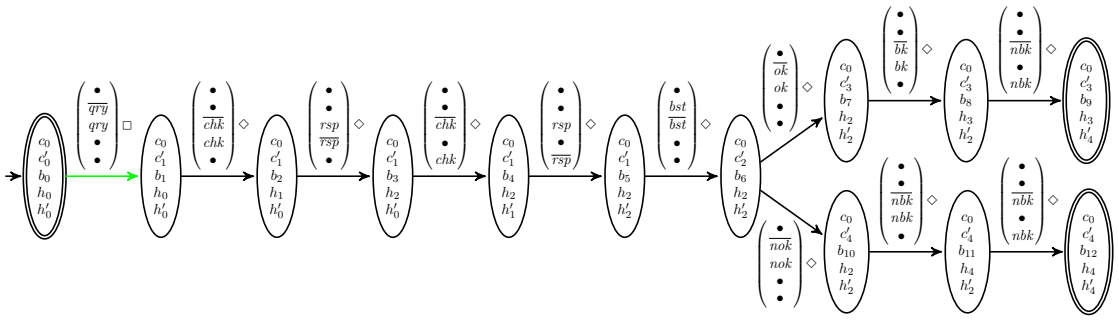

Figure 8: Choreography of Client ⊗ PrivilegedClient ⊗ Broker ⊗ Hotel ⊗ Hotel

	Num. states composition	Time (ms)	Num. states mpc	Time (ms)	Num. states orchestration	Time (ms)	Num. states choreography	Time (ms)
$\mathcal{A}_1$	2934	65594	1	4070	37	715216	–	–
$\mathcal{A}_2$	2934	66243	–	–	–	–	13	459311

Table 1: Results of computing the compositions $\mathcal{A}_1$ and $\mathcal{A}_2$ and their syntheses.[4]

Example 2.5 and Example 3.4, in $\mathcal{A}_2$ there is a privileged client and no privileged hotel. Indeed, `PrivilegedHotel` is not a valid contract for the choreography case. The choreography does not need the overhead of interactions with the orchestrator and, most importantly, the synthesis of choreography does not introduce any additional behaviour. Indeed, with due adjustment of necessary transitions of `PrivilegedClient` and `Hotel`, the choreography could be considered a sub-automaton of the orchestration.

We now use the example to discuss the differences between orchestration and choreography, and in particular the requirement that the *branching condition* is satisfied. In the orchestration, from the initial state either one of the two clients can interact. This decision is internally taken by the orchestrator (whose communications are abstracted away in the orchestration). On the converse, in the choreography only the `PrivilegedClient` is allowed to interact. This is because the clients are not able to decide on their own which one of them should start the interactions. This can be explained as follows. If both clients were allowed to interact, a deadlock could be reached upon the following steps. Initially, `PrivilegedClient` offers $\overline{qry}$. Afterwards, for the interactions to continue such offer must be received by some principal (i.e. the underlining choreographed model is synchronous [10]). In this case, `Broker` receives the offer qry. At this point, `Client` is allowed to offer its $\overline{qry}$ message. The interactions are now deadlocked, because `Broker` cannot receive such message, nor can any other contract. This is an example of violation of the branching condition. Consider the initial state $\vec{q}_0 = (c_0, c_0', b_0, h_0, h_0')$ and state $\vec{q}_1 = (c_0, c_1', b_1, h_0, h_0')$. In the orchestration, the branching condition is violated because from state $\vec{q}_0$ the match $(\overline{qry}, \bullet, qry, \bullet, \bullet)$ is allowed, while it is not in state $\vec{q}_1$, and in both states `Client` is in c_0. During the choreography synthesis, the match $(\overline{qry}, \bullet, qry, \bullet, \bullet)$ from state $\vec{q}_0$ is pruned. Likewise, in the choreography the broker enquires the hotels in a fixed order, whereas in the orchestration all possible orders are allowed, or else the branching condition would be violated.

[4]The evaluation was carried out on a machine with Processor Intel(R) Core(TM) i7-8500Y CPU at 1.50 GHz, 1601 Mhz, 2 Core(s), 4 Logical Processor(s) with 16 GB of RAM, running 64-bit Windows 10.

Note that an alternative choreography can be obtained by swapping the order in which the hotels are enquired by the broker. Indeed, from state $\vec{q}_1$ the orchestration allows both matches $(\bullet, \bullet, \overline{chk}, chk, \bullet)$ and $(\bullet, \bullet, \overline{chk}, \bullet, chk)$. During the choreography synthesis, both these outgoing matches are violating the branching condition. By pruning one of them, the other is automatically amended, because the states causing violation of the branching condition become dangling. In particular, the synthesis prunes the transition $(\bullet, \bullet, \overline{chk}, \bullet, chk)$ in favour of $(\bullet, \bullet, \overline{chk}, chk, \bullet)$.

Finally, concerning semi-controllability, note that it is not possible to have a choreography in which `PrivilegedClient` is not served in favour of `Client`, because the $\overline{qry}$ offer of `PrivilegedClient` is *necessary* and thus must be matched.

5. Abstract Synthesis

In Section 2, Section 3, and Section 4, we have presented three slightly different synthesis algorithms, and in the previous section we have illustrated their differences. As said before, to bridge the gap between standard synthesis and orchestration and choreography syntheses, the controllable and uncontrollable actions from SCT are related to permitted and necessary modalities, respectively, of MSCA.

The main intuition for this is that the SCT assumption of an unpredictable environment responsible for the uncontrollable transitions is not realistic in the case of coordination of services whose behaviour is known and observable. As a result, necessary actions are not in correspondence with uncontrollable actions, but rather require the introduction of a milder notion of controllability. The condition under which a controllable transition becomes uncontrollable varies depending on the particular synthesis algorithm (orchestration or choreography). Conversely, in the standard mpc synthesis such information is local, i.e. a transition is declared to be uncontrollable.

In this section, we discuss an abstract synthesis algorithm that generalises the previous algorithms by abstracting away the conditions under which a transition is pruned or a state is deemed bad, thus encapsulating and extrapolating the notion of controllability and safety. These two conditions, called *pruning predicate* (ϕ_p) and *forbidden predicate* (ϕ_f) are parameters to be instantiated by the corresponding instance of the synthesis algorithm (e.g. orchestration or choreography). Predicate ϕ_p is used for selecting the transitions to be pruned. Depending on the specific instance, non-local information about the automaton or the set of bad states is needed by ϕ_p. Therefore, ϕ_p takes as input the current transition to be checked, the automaton, and the set of bad states. If ϕ_p evaluates to true, then the corresponding transition will be pruned. Predicate ϕ_f is used for deciding whether a state becomes bad. The input parameters are the same as ϕ_p. However, ϕ_f only inspects necessary transitions ($T^{\Box}$). If ϕ_f evaluates to true, then the source state is deemed bad and added to the set R_i. The abstract synthesis algorithm is formally defined below.

Definition 5.1 (Abstract synthesis)**.** Let $\mathcal{A}$ be an MSCA, and let $\mathcal{K}_0 = \mathcal{A}$ and $R_0 = \mathit{Dangling}(\mathcal{K}_0)$. Given two predicates $\phi_p, \phi_f : T \times \mathit{MSCA} \times Q \to \mathsf{Bool}$, we let the *abstract synthesis function* $f_{(\phi_p,\phi_f)} : \mathit{MSCA} \times 2^Q \to \mathit{MSCA} \times 2^Q$ be defined as follows:

$$
\begin{aligned}
&f_{(\phi_p,\phi_f)}(\mathcal{K}_{i-1}, R_{i-1}) = (\mathcal{K}_i, R_i), \text{ with}\\
&\qquad T_{\mathcal{K}_i} = T_{\mathcal{K}_{i-1}} \setminus \{\, t \in T_{\mathcal{K}_{i-1}} \mid \phi_p(t, \mathcal{K}_{i-1}, R_{i-1}) = \mathit{true} \,\}\\
&\qquad R_i = R_{i-1} \cup \{\, \vec{q} \mid (\vec{q} \to) = t \in T^{\Box}_{\mathcal{A}}\,,\ \phi_f(t, \mathcal{K}_{i-1}, R_{i-1}) = \mathit{true} \,\} \cup \mathit{Dangling}(\mathcal{K}_i)
\end{aligned}
$$

As in the previous cases, the mpc relative to the pair (ϕ_p, ϕ_f) is obtained by computing the least fixed point $(\mathcal{K}_s, R_s)$ of $f_{(\phi_p,\phi_f)}$ and removing the states R_s from $\mathcal{K}_s$.

Theorem 5.2 (Abstract controller synthesis)**.** *The abstract synthesis function $f_{(\phi_p,\phi_f)}$ is monotone on the cpo $(MSCA \times 2^Q, \leq)$ and its* least fixed point *is:*

$$(\mathcal{K}_s^{(\phi_p,\phi_f)}, R_s^{(\phi_p,\phi_f)}) = \sup(\{\, f^n_{(\phi_p,\phi_f)}(\mathcal{K}_0, R_0) \mid n \in \mathbb{N} \,\})$$

The abstract controller *of $\mathcal{A}$ for predicates (ϕ_p, ϕ_f), denoted by $\mathcal{K}_{\mathcal{A}}^{(\phi_p,\phi_f)}$, is:*

$$\mathcal{K}_{\mathcal{A}}^{(\phi_p,\phi_f)} = \begin{cases} \langle\ \rangle & \text{if } \vec{q}_0 \in R_s^{(\phi_p,\phi_f)} \\ \langle Q \setminus R_s^{(\phi_p,\phi_f)}, \vec{q}_0, A^\Diamond, A^\Box, A^o, T_{\mathcal{K}_s^{(\phi_p,\phi_f)}}, F \setminus R_s^{(\phi_p,\phi_f)} \rangle & \text{otherwise} \end{cases}$$

Proof. The algorithm terminates because at each iteration either some transition is pruned or a state becomes forbidden, and both sets of transitions and states are finite. We now prove that the synthesised automaton is (i) non-blocking, (ii) controllable, (iii) most-permissive, and (iv) safe. In case $\mathcal{K}_{\mathcal{A}}^{(\phi_p,\phi_f)} = \langle\ \rangle$, the properties hold trivially, thus we assume that the synthesised controller is non-empty.

For (i), trivially all dangling states are pruned, so it is always possible to reach a final state. Similarly, bad states (i.e. states in the set R_s) are never traversed by construction, i.e. transitions with target in R_s are pruned.

For (ii) and (iv), the forbidden predicate ϕ_f codifies exactly when controllability or safety is violated by a state. By construction, it is never the case that such a state is reached.

For (iii), by construction a state is deemed bad or a transition is pruned exactly when either forbidden or pruning predicates are satisfied, respectively. Thus, maximality follows by the fact that each controller greater than the one synthesised will admit some forbidden state or transition. □

In the remainder of this section, we show how to instantiate the abstract synthesis function to the standard synthesis function, to the orchestration synthesis function, or to the choreography synthesis function, and prove their correspondences.

Theorem 5.3 (Abstract mpc synthesis)**.** *The standard synthesis function of Definition 2.3 coincides with the instantiation of the abstract synthesis function of Definition 5.1 where, for a generic transition $t = (\vec{q}, \vec{a}, \vec{q}')$, predicates ϕ_p and ϕ_f are defined as follows:*

$$\phi_p^{mpc}(t, \mathcal{K}, R) = (\vec{q}' \in R) \vee (\vec{q} \text{ is forbidden})$$
$$\phi_f^{mpc}(t, \mathcal{K}, R) = (\vec{q}' \in R)$$

Proof. Let $\mathcal{K}_{\mathcal{A}}^{mpc}$ and $\mathcal{K}_{\mathcal{A}}^{abs}$ be the controllers computed through Theorems 2.4 and 5.2, respectively. The proof proceeds by induction on the fixed point iterations and by case analysis.

For the base case, by definition $\mathcal{K}_0^{mpc} = \mathcal{K}_0^{abs} = \mathcal{A}$ and $R_0^{abs} = R_0^{mpc} = \mathit{Dangling}(\mathcal{K}_0)$.

For the inductive case, let i be a fixed point iteration. Assuming $\mathcal{K}_{i-1}^{mpc} = \mathcal{K}_{i-1}^{abs}$ and $R_{i-1}^{mpc} = R_{i-1}^{abs}$, we prove $\mathcal{K}_i^{mpc} = \mathcal{K}_i^{abs}$ and $R_i^{mpc} = R_i^{abs}$.

The equivalence $\mathcal{K}_i^{mpc} = \mathcal{K}_i^{abs}$ follows because at the ith iteration, ϕ_p^{mpc} detects exactly the same transitions that are pruned by the mpc synthesis algorithm.

For the equivalence $R_i^{mpc} = R_i^{abs}$, we have $R_i^{mpc} = R_{i-1}^{mpc} \cup \mathit{Dangling}(\mathcal{K}_i^{mpc}) \cup \{\, \vec{q} \mid (\vec{q}, \vec{a}, \vec{q}') \in T^{\Box}_{\mathcal{K}_i^{mpc}},\ \vec{q}' \in R_{i-1}^{mpc} \,\}$ and $R_i^{abs} = R_{i-1}^{abs} \cup \mathit{Dangling}(\mathcal{K}_i^{abs}) \cup \{\, \vec{q} \mid (\vec{q} \xrightarrow{a}) = t \in T^{\Box}_{\mathcal{A}},\ \phi_f^{mpc}(t, \mathcal{K}_{i-1}^{abs}, R_{i-1}^{abs}) = \mathit{true} \}$.

Since $\mathcal{K}_i^{mpc} = \mathcal{K}_i^{abs}$, also the dangling states are equivalent. It remains to prove that

$$\{\, \vec{q} \mid (\vec{q}, \vec{a}, \vec{q}') \in T^{\Box}_{\mathcal{K}_i^{mpc}},\ \vec{q}' \in R_{i-1}^{mpc} \,\} = \{\, \vec{q} \mid (\vec{q} \xrightarrow{a}) = t \in T^{\Box}_{\mathcal{A}},\ \phi_f^{mpc}(t, \mathcal{K}_{i-1}^{abs}, R_{i-1}^{abs}) = \mathit{true} \}.$$

This equivalence is straightforward by the definition of ϕ_f^{mpc} and the inductive hypothesis. □

Note that in Theorem 5.3 the predicates do not use any non-local information related to the parameter $\mathcal{K}$. For both orchestration and choreography, two different semi-controllability conditions are used to decide whether a state has become forbidden. These conditions are translated into the corresponding forbidden predicates.

Theorem 5.4 (Abstract orchestration synthesis)**.** *The orchestration synthesis function of Definition 3.2 coincides with the instantiation of the abstract synthesis function of Definition 5.1 where, for a generic transition $t = (\vec{q}, \vec{a}, \vec{q}')$, predicates ϕ_p and ϕ_f are defined as follows:*

$$\begin{aligned}
\phi_p^{orc}(t, \mathcal{K}, R) &= (t \textit{ is a request }) \vee (\vec{q}' \in R) \\
\phi_f^{orc}(t, \mathcal{K}, R) &= \nexists\, (\vec{q_2} \xrightarrow{\vec{a}_2} \vec{q_2}') \in T^{\Box}_{\mathcal{K}} \,:\, (\vec{a}_2 \textit{ is a match }) \wedge (\vec{q}_2, \vec{q_2}' \notin \mathit{Dangling}(\mathcal{K})) \\
&\qquad \wedge (\vec{q}_{(i)} = \vec{q_2}_{(i)}) \wedge (\vec{a}_{(i)} = \vec{a_2}_{(i)} = a)
\end{aligned}$$

Proof (sketch). The proof is analogous to that of Theorem 5.3 but relying on Theorem 3.2 instead of Theorem 2.4. The full proof can be found in the appendix. □

The pruning predicate of Theorem 5.4 does not use any information coming from the global automaton $\mathcal{K}$, whereas this is no longer the case for the forbidden predicate that indeed specifies the semi-controllability condition for the necessary transitions of an orchestration (cf. Definition 3.1).

Theorem 5.5 (Abstract choreography synthesis)**.** *The choreography synthesis function of Definition 4.4 coincides with the instantiation of the abstract synthesis function of Definition 5.1, where given a generic transition $t = (\vec{q}, \vec{a}, \vec{q}')$, the predicates ϕ_p and ϕ_f are defined as follows, where $\hat{T}_{\mathcal{K},R}$ is defined in Definition 4.4:*

$$\begin{aligned}
\phi_p^{cor}(t, \mathcal{K}, R) &= (t \textit{ is a request or offer}) \vee (\vec{q}' \in R) \vee t \in \hat{T}_{\mathcal{K},R} \\
\phi_f^{cor}(t, \mathcal{K}, R) &= \nexists\, (\vec{q} \xrightarrow{\vec{a}_2} \vec{q_2}') \in T^{\Box}_{\mathcal{K}} \,:\, (\vec{a}_2 \textit{ is a match}) \wedge (\vec{q}, \vec{q_2}' \notin \mathit{Dangling}(\mathcal{K})) \\
&\qquad \wedge (\vec{a}_{(i)} = \vec{a_2}_{(i)} = \overline{a})
\end{aligned}$$

Proof (sketch). The proof is analogous to that of Theorem 5.3 but relying on Theorem 4.4 instead of Theorem 2.4. The full proof can be found in the appendix. □

Notably, in Theorem 5.5 both predicates require global information on the whole automaton. Similarly to Theorem 5.4, the forbidden predicate codifies the semi-controllability condition of Definition 4.3. Moreover, the pruning predicate removes all transitions violating the branching condition (cf. Definition 4.1).

6. A Partial Order on Controllers

In Theorem 5.3, Theorem 5.4, and Theorem 5.5, we have proved that the three previously presented synthesis algorithms are instantiations of the abstract synthesis algorithm of Definition 5.1. This abstraction provides us the mean to formally relate the various algorithms presented so far, as detailed in this section.

To begin with, we define a partial order on predicates. Intuitively, a pair (ϕ_{p_2}, ϕ_{f_2}) is greater than another pair (ϕ_{p_1}, ϕ_{f_1}) if and only if (ϕ_{p_2}, ϕ_{f_2}) is (pairwise) entailed by (ϕ_{p_1}, ϕ_{f_1}).

Definition 6.1 (Partial order on predicates)**.** Let $\mathcal{A}$ be an MSCA and let $\texttt{Pr}$ be the set of pairs of pruning and forbidden predicates of Definition 5.1 with $(\phi_{p_1}, \phi_{f_1}), (\phi_{p_2}, \phi_{f_2}) \in \texttt{Pr}$. The partial order on predicates $(\texttt{Pr}, \leq)$ is defined as:

$$(\phi_{p_1}, \phi_{f_1}) \leq (\phi_{p_2}, \phi_{f_2}) \text{ iff } \forall i \in \mathbb{N} \,.\, (\phi_{p_1}(t, \mathcal{K}_i^1, R_i^1) \Rightarrow (\phi_{p_2}(t, \mathcal{K}_i^2, R_i^2) \vee t \notin \mathcal{K}_i^2)) \\ \wedge (\phi_{f_1}(t, \mathcal{K}_i^1, R_i^1) \Rightarrow (\phi_{f_2}(t, \mathcal{K}_i^2, R_i^2)) \vee \vec{q} \in \mathit{Dangling}(\mathcal{K}_i^2)),$$

where $t = (\vec{q}, \vec{a}, \vec{q}')$.

By Definition 5.1, we know that such predicates are used to refine an MSCA during the synthesis. Indeed, states and transitions are removed when such predicates are satisfied by them. The partial order on predicates induces an ordering on the various abstract controllers, as the following result shows.

Proposition 6.2 (Ordering controllers)**.** *Let $\mathcal{A}$ be an MSCA and let $(\phi_{p_1}, \phi_{f_1}), (\phi_{p_2}, \phi_{f_2}) \in \texttt{Pr}$ be such that $(\phi_{p_1}, \phi_{f_1}) \leq (\phi_{p_2}, \phi_{f_2})$. Then:*

$$\mathcal{K}_{\mathcal{A}}^{(\phi_{p_2}, \phi_{f_2})} \subseteq \mathcal{K}_{\mathcal{A}}^{(\phi_{p_1}, \phi_{f_1})}$$

Proof. By Definition 5.1, both $\mathcal{K}_{\mathcal{A}}^{(\phi_{p_1}, \phi_{f_1})}$ and $\mathcal{K}_{\mathcal{A}}^{(\phi_{p_2}, \phi_{f_2})}$ are sub-automata of $\mathcal{A}$, and they only differ in the sets of states and transitions.

By contradiction, assume that there exists a transition t in $T_{\mathcal{K}_{\mathcal{A}}^{(\phi_{p_2}, \phi_{f_2})}} \setminus T_{\mathcal{K}_{\mathcal{A}}^{(\phi_{p_1}, \phi_{f_1})}}$. By Definition 5.1, let i be the iteration where t is removed from $T_{\mathcal{K}_i^1}$. By hypothesis, it holds that $\phi_{p_1}(t, \mathcal{K}_i^1, R_i^1) \Rightarrow \phi_{p_2}(t, \mathcal{K}_i^2, R_i^2) \vee t \notin \mathcal{K}_i^2$, hence by Definition 5.1, t must also have been removed from $T_{\mathcal{K}_i^2}$ or it is not present, a contradiction.

Similarly, assume that there exists a state $\vec{q}$ in $Q_{\mathcal{K}_{\mathcal{A}}^{(\phi_{p_2}, \phi_{f_2})}} \setminus Q_{\mathcal{K}_{\mathcal{A}}^{(\phi_{p_1}, \phi_{f_1})}}$. By Definition 5.1, let i be the iteration where $\vec{q}$ is added to $R_{\mathcal{K}_i^1}$. By hypothesis, it holds that $\phi_{f_1}(t, \mathcal{K}_i^1, R_i^1) \Rightarrow \phi_{f_2}(t, \mathcal{K}_i^2, R_i^2) \vee \vec{q} \in \mathit{Dangling}(\mathcal{K}_i^2)$, hence by Definition 5.1, $\vec{q}$ must also have been added to $R_{\mathcal{K}_i^2}$. Finally, $Q_{\mathcal{K}_{\mathcal{A}}^{(\phi_{p_2}, \phi_{f_2})}} = Q_{\mathcal{A}} \setminus R_s^2$ and $R_{\mathcal{K}_i^2} \subseteq R_s^2$, thus a contradiction is reached. □

This result has an immediate application in performing abstraction of syntheses, in the sense that the lesser the pair of predicates the more abstract (in refinement terms) the corresponding synthesised automaton. This can be useful to perform partial syntheses and skip unnecessary checks or even potentially undecidable computations. For example, if $\mathcal{K}^{(\phi_{p_1}, \phi_{f_1})} = \langle\,\rangle$, for a given pair (ϕ_{p_1}, ϕ_{f_1}), then by Proposition 6.2 we know that for all (ϕ_{p_i}, ϕ_{f_i}) such that $(\phi_{p_1}, \phi_{f_1}) \leq (\phi_{p_i}, \phi_{f_i})$ it will hold that $\mathcal{K}^{(\phi_{p_i}, \phi_{f_i})} = \langle\,\rangle$.

While the orchestration synthesis of Definition 3.2 is enforcing agreement, the mpc synthesis of Definition 2.3 is enforcing a generic predicate modelled as forbidden states. Whenever the mpc synthesis is also enforcing agreement, as an instantiation of Proposition 6.2, we can prove that the two syntheses are related. Moreover, agreement identifies forbidden

transitions as those labelled by requests. On the converse, the mpc synthesis identifies forbidden states rather than forbidden transitions. Therefore, to enable a comparison of the mpc and the orchestration synthesis, we need to (i) transform the automaton such that the predicate on forbidden transitions (i.e. agreement in this case) can be expressed by means of forbidden states and (ii) instantiate the generic predicate expressed by forbidden states. For point (i), the synthesis of the mpc is applied to the automaton $\mathcal{A}'$ obtained from the original automaton $\mathcal{A}$ by erasing controllable forbidden transitions. For point (ii), forbidden states are those states that are sources of uncontrollable forbidden transitions. This is what the following lemma states.

Lemma 6.3 (Orchestration vs. mpc synthesis)**.** *Given an MSCA $\mathcal{A}$, let $\mathcal{A}'$ be obtained from $\mathcal{A}$ by removing all controllable request transitions and considering as forbidden the states of $\mathcal{A}'$ with outgoing uncontrollable request transitions. Let $\mathcal{K}^{orc}_{\mathcal{A}}$ and $\mathcal{K}^{mpc}_{\mathcal{A}'}$ be the orchestration and mpc of Definitions 3.2 and 2.3, respectively. Then:*

$$\mathcal{K}^{mpc}_{\mathcal{A}'} \subseteq \mathcal{K}^{orc}_{\mathcal{A}}$$

Proof. By Theorems 5.4 and 5.3, $\mathcal{K}^{orc}_{\mathcal{A}}$ and $\mathcal{K}^{mpc}_{\mathcal{A}'}$ are equivalent to $\mathcal{K}^{(\phi^{orc}_p, \phi^{orc}_f)}_{\mathcal{A}}$ and $\mathcal{K}^{(\phi^{mpc}_p, \phi^{mpc}_f)}_{\mathcal{A}'}$, respectively. Moreover, both controllers are sub-automata of $\mathcal{A}$, and they only differ in the sets of states and transitions.

Recall that, given $t = (\vec{q}, \vec{a}, \vec{q}')$, $\phi^{mpc}_p(t, \mathcal{K}^{mpc}_i, R^{mpc}_i) = (\vec{q}' \in R^{mpc}_i) \vee (\vec{q}$ is forbidden), $\phi^{mpc}_f(t, \mathcal{K}^{mpc}_i, R^{mpc}_i) = (\vec{q}' \in R^{mpc}_i)$ and $\phi^{orc}_p(t, \mathcal{K}^{orc}_i, R^{orc}_i) = (t$ is a request $) \vee (\vec{q}' \in R^{orc}_i)$, $\phi^{orc}_f(t, \mathcal{K}^{orc}_i, R^{orc}_i) = \nexists\, (\vec{q_2} \xrightarrow{\vec{a}_2} \vec{q_2}') \in T^{\square}_{\mathcal{K}^{orc}_i} : (\vec{a}_2$ is a match $) \wedge (\vec{q}_2, \vec{q_2}' \notin \mathit{Dangling}(\mathcal{K}^{orc}_i)) \wedge (\vec{q}_{(i)} = \vec{q2}_{(i)}) \wedge (\vec{a}_{(i)} = \vec{a2}_{(i)} = a)$.

We proceed by induction on i. For the base case, it holds that $\mathcal{K}_{\mathcal{A}'_0} \subseteq \mathcal{K}_0$ and $\mathit{Dangling}(\mathcal{K}_0) \subseteq \mathit{Dangling}(\mathcal{K}_{\mathcal{A}'_0})$. By hypothesis, $\phi^{orc}_p(t, \mathcal{K}^{orc}_0, R^{orc}_0)$ is true. Then either t is a request or $\vec{q}' \in \mathit{Dangling}(\mathcal{K}_0)$. If t is a request, then t has been already pruned. Otherwise, $\vec{q}' \in \mathit{Dangling}(\mathcal{K}_0)$ (or both), and so it is in $\mathit{Dangling}(\mathcal{K}_{\mathcal{A}'_0})$ and the pruning predicate of the mpc is satisfied. Similarly, by hypothesis $\phi^{orc}_f(t, \mathcal{K}^{orc}_0, R^{orc}_0)$ is true. Since no transitions have been pruned in $\mathcal{K}_0$, it must be the case that the source state of t is in $\mathit{Dangling}(\mathcal{K}_0)$, and so it is in $\mathit{Dangling}(\mathcal{K}_{\mathcal{A}'_0})$.

For the inductive step, the implication on the pruning predicate is satisfied by noticing that $R^{orc}_{i-1} \subseteq R^{mpc}_{i-1}$. The implication on the forbidden predicate is satisfied because trivially $t \notin T^{\square}_{\mathcal{K}^{orc}_i}$, and hence $t \notin T^{\square}_{\mathcal{K}^{mpc}_i}$, and this is because either the target is dangling or the source is forbidden. In both cases the forbidden predicate of the mpc is satisfied. □

Thus, for example, given an MSCA $\mathcal{A}$, from $\mathcal{K}^{orc}_{\mathcal{A}} = \langle\ \rangle$ we can conclude that $\mathcal{K}^{mpc}_{\mathcal{A}} = \langle\ \rangle$ by Lemma 6.3, without actually computing it.

Example 6.4. Concluding the running example, one can observe that the mpc of $\mathcal{A}_1$ is a sub-automaton (formed of only the initial and final state) of the orchestration of $\mathcal{A}_1$.

7. Related Work

Our contributions to bridging the gap between SCT and coordination of services concern adaptations of the classical synthesis algorithm from SCT in order to synthesise orchestrations and choreographies of service contracts formalised as MSCA. In the literature, there exist many formalisms for modelling and analysing (service) contracts, ranging from behavioural

type systems, including behavioural contracts [21, 1, 36] and session types [17, 32, 27, 20, 40], to automata-based formalisms, including interface automata [26] and (timed) (I/O) automata [39, 2, 25]. Foundational models for service contracts and session types are surveyed in [44, 7, 33].

The MSCA formalism used in this paper differs fundamentally from these models, which typically study notions of contract compliance involving only two parties, since MSCA primitively support *multi-party* compliance of contracts that *compete* on offering or requesting the same service. Furthermore, the above models do not consider modalities of services whereas MSCA provide primitive support for *permitted* and *necessary* service actions, which resulted in the introduction of a novel notion of *semi-controllability* in the context of SCT. Modal Transition Systems (MTS) and their extensions [35], as adopted for instance in Software Product Line Engineering (SPLE [42, 3]), like modal I/O automata [38] and MTS with variability constraints [47], do natively distinguish may and must modalities, but the other differences remain. In particular, they cannot explicitly handle dynamic composition by allowing new services that join composite services to intercept already matched actions.

We are only aware of two other applications of SCT to MTS. In [24], there is no direct relation between may/must and controllable/uncontrollable, and the modal automaton (i.e. MTS with final states) is seen as a predicate that is satisfied if the plant automaton (i.e. the system to be refined against the predicate) is a sort of alternate refinement of the predicate. Similarly, in [28], the control objectives (i.e. the predicate) is a modal automaton, non-blockingness is not considered, and another modal automaton describes which actions are controllable and which are uncontrollable in the plant automaton. In this paper, the predicate is an invariant (i.e. forbidden states and forbidden transitions are given), the modal automaton (i.e. MSCA) is the plant, and a necessary transition induces different notions of controllability according to the adopted coordination paradigm.

SCT was first applied to SPLE in [48] by showing how the `CIF 3` toolset [50] can automatically synthesise a single (global, family) model representing an automaton for each of the valid products of a product line from (i) a feature constraint with attributes (e.g. cost), (ii) behavioural component models associated with the features, and (iii) additional behavioural requirements like state invariants, action orderings, and guards on actions (reminiscent of the Featured Transition Systems of [22]). The resulting `CIF 3` model satisfies all feature-related constraints as well as all given behavioural requirements. Since `CIF 3` allows the export of such models in a format accepted by the `mCRL2` model checker [23], the latter can be used to verify arbitrary behavioural properties expressed in the modal μ-calculus with data or its feature-oriented variant of [46]. An important advantage is that both `CIF 3` and `mCRL2` can be used off-the-shelf, meaning that no additional tools are required. Differently from our approach, all actions are controllable and orchestration is not considered. In [9], the prototypical tool CAT supporting orchestration synthesis for CA is presented.

The only approach by others to bridge the gap between SCT and coordination of services that we are aware of is that of [5], where services are formalised as so-called Service Labelled Transition Systems (SLTS), which are a kind of guarded automata with data. To this aim, SCT is adapted to deal with conditions and variables as well as with a means to enforce services based on runtime information. However, service composition through SLTS is based on the standard synchronous product, whilst the contract composition expresses competing contracts. More importantly, in [5], input actions are considered uncontrollable whereas output actions are controllable, in the standard view of a service interacting with the environment. Our contribution induces novel notions of controllability to express necessary

requirements that are semi-controllable. The standard controller synthesis algorithm is used in [30] to synthesise adapters between services. These adapters act like proxies and are used to enforce properties such as deadlock-freedom. Compared to our work, the interactions between services are driven by their contracts rather than by adapters. The standard controller synthesis algorithm cannot be applied to synthesise a correct composition of contracts.

We conclude this section by describing two recent extensions of MSCA, developed for different purposes, and for which we also defined adapted synthesis algorithms. In [12], we presented Featured Modal Contract Automata (FMCA). Technically, we extended MSCA with a variability mechanism concerning structural constraints that operate on the service contract, used to define different configurations. This reflects the fact that services are typically reused in configurations that vary over time and need to dynamically adapt to changing environments [51]. Configurations were characterised by which service actions are mandatory and which forbidden. The valid configurations were defined as those respecting all structural constraints. We followed the well-established paradigm of SPLE, which aims at efficiently managing a product line (family) of highly (re)configurable systems to allow for mass customisation [42, 3]. To compactly represent a product line, i.e. the set of valid product configurations, we used a so-called feature constraint, a propositional formula φ whose atoms are features [15], and we identified features as service actions (offers as well as requests). A valid product then distinguishes a set of mandatory and a set of forbidden actions. Consequently, we defined an algorithm to compute the FMCA $\mathcal{K}_{\mathcal{A}_p}$ as the mpc for a valid product p of an FMCA $\mathcal{A}$. The main adaptation of the synthesis algorithm for MSCA was to consider as bad states also those that cannot prevent a forbidden action to be eventually executed and to discard the transitions labelled with actions forbidden by p. Moreover, if some action that is mandatory in p is unavailable in the automaton that results from the fixed point iteration, then the mpc results empty. In [12], we also presented an evaluation of FMCA with the prototypical tool FMCAT. Building on CAT [9], FMCAT can synthesise the orchestration of an FMCA in terms of its mpc. The results clearly show the gain in expressiveness due to the notion of semi-controllability, as well as the reduction of the number of configurations needed to compute the orchestration due to the introduction of a partial order of products of FMCA. This inspired us to consider semi-controllability also in MSCA and to develop a partial order of controllers for MSCA in this paper.

In [13], we presented Timed Service Contract Automata (TSCA) as an extension of the FMCA from [12] with real-time constraints. Formally, a configuration of a TSCA is a triple consisting of a recognised trace, a state, and a valuation of clocks. The (finite) behaviour recognised by a TSCA are traces of alternating time and discrete transitions, i.e. in a given configuration either time progresses (a silent action in the languages recognised by TSCA) or a discrete step to a new configuration is performed. Consequently, we defined an algorithm to compute the orchestration synthesis of TSCA. To respect the timing constraints, we used the notion of zones from timed games [4, 19]. The resulting synthesis algorithm resembles a timed game, but it differs from classical timed game algorithms [4, 19, 25] by combining two separate games, viz. *reachability* games (to ensure that marked states must be reachable) and *safety* games (to ensure that forbidden states are never traversed). A TSCA might be such that all bad configurations are unreachable (i.e. it is safe), while at the same time no final configuration is reachable (i.e. the resulting orchestration is empty).

8. Conclusion

This paper presents our recent efforts, originally published in [14], concerning bridging the gap between the most permissive controller synthesis from Supervisory Control Theory with synthesis algorithms of orchestrations and choreographies for a formal model of service contracts called Modal Service Contract Automata. This includes a novel algorithm capable of synthesising a safe non-blocking composition of service contracts that is directly translatable into a choreographed formalism. A further contribution is an abstract synthesis algorithm that generalises the synthesis of the choreography, as well as that of the orchestration and that of the most permissive controller. This paper includes the proofs of all statements from [14]. Furthermore, it contains a formal demonstration that the different synthesis algorithms are related through a notion of refinement, which allows us to formally prove that, under mild assumptions, the orchestration synthesis is an abstraction of the mpc synthesis. Finally, the paper includes an extensive running example from the service domain that illustrates our contributions.

The properties to be enforced in the algorithms presented in this paper are all invariants specified through either forbidden states or forbidden transitions. Future work is needed to investigate the abstract syntheses under other non-invariant properties. Another avenue for future research is to investigate the different features of micro-services with respect to services, and to study what is needed to adapt the formalism of (timed/modal service) contract automata and our results to deal with micro-services.

Acknowledgments

We acknowledge useful comments from the reviewers and funding from the MIUR PRIN 2017FTXR7S project IT MaTTerS (Methods and Tools for Trustworthy Smart Systems).

References

[1] L. Acciai, M. Boreale, and G. Zavattaro. Behavioural contracts with request-response operations. *Sci. Comp. Program.*, 78(2):248–267, 2013. doi:10.1016/j.scico.2011.10.007.

[2] R. Alur and D. Dill. A Theory of Timed Automata. *Theoret. Comp. Sci.*, 126(2):183–235, 1994. doi:10.1016/0304-3975(94)90010-8.

[3] S. Apel, D. S. Batory, C. Kästner, and G. Saake. *Feature-Oriented Software Product Lines: Concepts and Implementation*. Springer, 2013. doi:10.1007/978-3-642-37521-7.

[4] E. Asarin, O. Maler, A. Pnueli, and J. Sifakis. Controller Synthesis for Timed Automata. *IFAC Proc. Vol.*, 31(18):447–452, 1998. doi:10.1016/S1474-6670(17)42032-5.

[5] F. Atampore, J. Dingel, and K. Rudie. Automated Service Composition Via Supervisory Control Theory. In *WODES*, pages 28–35. IEEE, 2016. doi:10.1109/WODES.2016.7497822.

[6] S. Azzopardi, G. J. Pace, F. Schapachnik, and G. Schneider. Contract automata: An operational view of contracts between interactive parties. *Artif. Intell. Law*, 24(3):203–243, 2016. doi:10.1007/s10506-016-9185-2.

[7] M. Bartoletti, T. Cimoli, and R. Zunino. Compliance in Behavioural Contracts: A Brief Survey. In *Programming Languages with Applications to Biology and Security*, volume 9465 of *LNCS*, pages 103–121. Springer, 2015. doi:10.1007/978-3-319-25527-9_9.

[8] D. Basile, P. Degano, and G. L. Ferrari. Automata for Specifying and Orchestrating Service Contracts. *Log. Meth. Comp. Sci.*, 12(4:6):1–51, 2016. doi:10.2168/LMCS-12(4:6)2016.

[9] D. Basile, P. Degano, G. L. Ferrari, and E. Tuosto. Playing with Our CAT and Communication-Centric Applications. In *FORTE*, volume 9688 of *LNCS*, pages 62–73. Springer, 2016. doi:10.1007/978-3-319-39570-8_5.

[10] D. Basile, P. Degano, G. L. Ferrari, and E. Tuosto. Relating two automata-based models of orchestration and choreography. *J. Log. Algebr. Meth. Program.*, 85(3):425–446, 2016. doi:10.1016/j.jlamp.2015.09.011.

[11] D. Basile, F. Di Giandomenico, S. Gnesi, P. Degano, and G. L. Ferrari. Specifying Variability in Service Contracts. In *VaMoS*, pages 20–27. ACM, 2017. doi:10.1145/3023956.3023965.

[12] D. Basile, M. H. ter Beek, P. Degano, A. Legay, G. L. Ferrari, S. Gnesi, and F. Di Giandomenico. Controller synthesis of service contracts with variability. *Science of Computer Programming*, 187, 2020. doi:10.1016/j.scico.2019.102344.

[13] D. Basile, M. H. ter Beek, and A. Legay. Timed service contract automata. *Innovations Syst. Softw. Eng.*, 2020. doi:10.1007/s11334-019-00353-3.

[14] D. Basile, M. H. ter Beek, and R. Pugliese. Bridging the Gap Between Supervisory Control and Coordination of Services: Synthesis of Orchestrations and Choreographies. In *COORDINATION*, volume 11533 of *LNCS*, pages 129–147. Springer, 2019. doi:10.1007/978-3-030-22397-7_8.

[15] D. S. Batory. Feature Models, Grammars, and Propositional Formulas. In *SPLC*, volume 3714 of *LNCS*, pages 7–20. Springer, 2005. doi:10.1007/11554844_3.

[16] A. Bouguettaya, M. Singh, M. Huhns, Q. Z. Sheng, H. Dong, Q. Yu, A. G. Neiat, S. Mistry, B. Benatallah, B. Medjahed, M. Ouzzani, F. Casati, X. Liu, H. Wang, D. Georgakopoulos, L. Chen, S. Nepal, Z. Malik, A. Erradi, Y. Wang, B. Blake, S. Dustdar, F. Leymann, and M. Papazoglou. A Service Computing Manifesto: The Next 10 Years. *Commun. ACM*, 60(4):64–72, 2017. doi:10.1145/2983528.

[17] R. Bruni, I. Lanese, H. C. Melgratti, and E. Tuosto. Multiparty Sessions in SOC. In *COORDINATION*, volume 5052 of *LNCS*, pages 67–82. Springer, 2008. doi:10.1007/978-3-540-68265-3_5.

[18] C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Springer, 2006. doi:10.1007/978-0-387-68612-7.

[19] F. Cassez, A. David, E. Fleury, K. G. Larsen, and D. Lime. Efficient On-the-Fly Algorithms for the Analysis of Timed Games. In *CONCUR*, volume 3653 of *LNCS*, pages 66–80. Springer, 2005. doi:10.1007/11539452_9.

[20] G. Castagna, M. Dezani-Ciancaglini, and L. Padovani. On Global Types and Multi-Party Sessions. *Log. Meth. Comp. Sci.*, 8(1:24):1–45, 2012. doi:10.2168/LMCS-8(1:24)2012.

[21] G. Castagna, N. Gesbert, and L. Padovani. A Theory of Contracts for Web Services. *ACM Trans. Program. Lang. Syst.*, 31(5):19:1–19:61, 2009. doi:10.1145/1538917.1538920.

[22] A. Classen, M. Cordy, P. - Y. Schobbens, P. Heymans, A. Legay, and J. - F. Raskin. Featured Transition Systems: Foundations for Verifying Variability-Intensive Systems and Their Application to LTL Model Checking. *IEEE Trans. Softw. Eng.*, 39(8):1069–1089, 2013. doi:10.1109/TSE.2012.86.

[23] S. Cranen, J. F. Groote, J. J. A. Keiren, F. P. M. Stappers, E. P. de Vink, W. Wesselink, and T. A. C. Willemse. An Overview of the mCRL2 Toolset and Its Recent Advances. In *TACAS*, volume 7795 of *LNCS*, pages 199–213. Springer, 2013. doi:10.1007/978-3-642-36742-7_15.

[24] P. Darondeau, J. Dubreil, and H. Marchand. Supervisory Control for Modal Specifications of Services. *IFAC Proc. Vol.*, 43(12):418–425, 2010. doi:10.3182/20100830-3-DE-4013.00069.

[25] A. David, K. G. Larsen, A. Legay, U. Nyman, and A. Wąsowski. Timed I/O Automata: A Complete Specification Theory for Real-time Systems. In *HSCC*, pages 91–100. ACM, 2010. doi:10.1145/1755952.1755967.

[26] L. de Alfaro and T. Henzinger. Interface Automata. In *ESEC/FSE*, pages 109–120. ACM, 2001. doi:10.1145/503209.503226.

[27] M. Dezani-Ciancaglini and U. de'Liguoro. Sessions and Session Types: An Overview. In *WS-FM*, volume 6194 of *LNCS*, pages 1–28. Springer, 2010. doi:10.1007/978-3-642-14458-5_1.

[28] G. Feuillade and S. Pinchinat. Modal Specifications for the Control Theory of Discrete Event Systems. *Discrete Event Dyn. Syst.*, 17(2):211–232, 2007. doi:10.1007/s10626-006-0008-6.

[29] S. T. J. Forschelen, J. M. van de Mortel-Fronczak, R. Su, and J. E. Rooda. Application of supervisory control theory to theme park vehicles. *Discrete Event Dyn. Syst.*, 22(4):511–540, 2012. doi:10.1007/s10626-012-0130-6.

[30] C. Gierds, A. J. Mooij, and K. Wolf. Reducing Adapter Synthesis to Controller Synthesis. *IEEE Trans. Services Computing*, 5(1):72–85, 2012. doi:10.1109/TSC.2010.57.

[31] P. Gohari and W. M. Wonham. On the complexity of supervisory control design in the RW framework. *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, 30(5):643–652, 2000. doi:10.1109/3477.875441.

[32] K. Honda, N. Yoshida, and M. Carbone. Multiparty Asynchronous Session Types. In *POPL*, pages 273–284. ACM, 2008. doi:10.1145/1328438.1328472.
[33] H. Hüttel, I. Lanese, V. T. Vasconcelos, L. Caires, M. Carbone, P. - M. Deniélou, D. Mostrous, L. Padovani, A. Ravara, E. Tuosto, H. Torres Vieira, and G. Zavattaro. Foundations of Session Types and Behavioural Contracts. *ACM Comput. Surv.*, 49(1):3:1–3:36, 2016. doi:10.1145/2873052.
[34] N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, Y. Lafon, and C. Barreto. Web Services Choreography Description Language v1.0. https://www.w3.org/TR/ws-cdl-10/, 2005.
[35] J. Křetínský. 30 Years of Modal Transition Systems: Survey of Extensions and Analysis. In *Models, Algorithms, Logics and Tools*, volume 10460 of *LNCS*, pages 36–74. Springer, 2017. doi:10.1007/978-3-319-63121-9_3.
[36] C. Laneve and L. Padovani. An algebraic theory for web service contracts. *Form. Asp. Comp.*, 27(4):613–640, 2015. doi:10.1007/s00165-015-0334-2.
[37] J. Lange, E. Tuosto, and N. Yoshida. From Communicating Machines to Graphical Choreographies. In *POPL*, pages 221–232. ACM, 2015. doi:10.1145/2676726.2676964.
[38] K. G. Larsen, U. Nyman, and A. Wąsowski. Modal I/O Automata for Interface and Product Line Theories. In *ESOP*, volume 4421 of *LNCS*, pages 64–79. Springer, 2007. doi:10.1007/978-3-540-71316-6_6.
[39] N. Lynch and M. Tuttle. An Introduction to Input/Output Automata. *CWI Q.*, 2:219–246, 1989. URL: https://ir.cwi.nl/pub/18164/18164A.pdf.
[40] J. Michaux, E. Najm, and A. Fantechi. Session types for safe Web service orchestration. *J. Log. Algebr. Program.*, 82(8):282–310, 2013. doi:10.1016/j.jlap.2013.05.004.
[41] C. Peltz. Web Services Orchestration and Choreography. *IEEE Comp.*, 36(10):46–52, 2003. doi:10.1109/MC.2003.1236471.
[42] K. Pohl, G. Böckle, and F. J. van der Linden. *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer, 2005. doi:10.1007/3-540-28901-1.
[43] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. Control Optim.*, 25(1):206–230, 1987. doi:10.1137/0325013.
[44] M. H. ter Beek, A. Bucchiarone, and S. Gnesi. Web Service Composition Approaches: From Industrial Standards to Formal Methods. In *ICIW*. IEEE, 2007. doi:10.1109/ICIW.2007.71.
[45] M. H. ter Beek, J. Carmona, R. Hennicker, and J. Kleijn. Communication Requirements for Team Automata. In *COORDINATION*, volume 10319 of *LNCS*, pages 256–277. Springer, 2017. doi:10.1007/978-3-319-59746-1_14.
[46] M. H. ter Beek, E. P. de Vink, and T. A. C. Willemse. Family-Based Model Checking with mCRL2. In *FASE*, volume 10202 of *LNCS*, pages 387–405. Springer, 2017. doi:10.1007/978-3-662-54494-5_23.
[47] M. H. ter Beek, A. Fantechi, S. Gnesi, and F. Mazzanti. Modelling and analysing variability in product families: Model checking of modal transition systems with variability constraints. *J. Log. Algebr. Meth. Program.*, 85(2):287–315, 2016. doi:10.1016/j.jlamp.2015.11.006.
[48] M. H. ter Beek, M. A. Reniers, and E. P. de Vink. Supervisory Controller Synthesis for Product Lines Using CIF 3. In *ISoLA*, volume 9952 of *LNCS*, pages 856–873. Springer, 2016. doi:10.1007/978-3-319-47166-2_59.
[49] R. J. M. Theunissen, D. A. van Beek, and J. E. Rooda. Improving evolvability of a patient communication control system using state-based supervisory control synthesis. *Adv. Eng. Inform.*, 26(3):502–515, 2012. doi:10.1016/j.aei.2012.02.009.
[50] D. A. van Beek, W. J. Fokkink, D. Hendriks, A. Hofkamp, J. Markovski, J. M. van de Mortel-Fronczak, and M. A. Reniers. CIF 3: Model-Based Engineering of Supervisory Controllers. In *TACAS*, volume 8413 of *LNCS*, pages 575–580. Springer, 2014. doi:10.1007/978-3-642-54862-8_48.
[51] Q. Yi, X. Liu, A. Bouguettaya, and B. Medjahed. Deploying and managing Web services: issues, solutions, and directions. *VLDB J.*, 17(3):735–572, 2008. doi:10.1007/s00778-006-0020-3.

Appendix A. Proofs

We provide the proofs of Theorem 5.4 and Theorem 5.5 only sketched in Section 5.

Theorem 5.4 (Abstract orchestration synthesis)**.** *The orchestration synthesis function of Definition 3.2 coincides with the instantiation of the abstract synthesis function of Definition 5.1 where, for a generic transition $t = (\vec{q}, \vec{a}, \vec{q}')$, predicates ϕ_p and ϕ_f are defined as follows:*

$$\begin{aligned}\phi_p^{orc}(t, \mathcal{K}, R) &= (t \textit{ is a request }) \vee (\vec{q}' \in R)\\ \phi_f^{orc}(t, \mathcal{K}, R) &= \nexists\, (\vec{q_2} \xrightarrow{\vec{a}_2} \vec{q_2}') \in T^{\Box}_{\mathcal{K}} \,:\, (\vec{a}_2 \textit{ is a match }) \wedge (\vec{q}_2, \vec{q}_2' \notin \textit{Dangling}(\mathcal{K}))\\ &\qquad \wedge (\vec{q}_{(i)} = \vec{q_2}_{(i)}) \wedge (\vec{a}_{(i)} = \vec{a_2}_{(i)} = a)\end{aligned}$$

Proof. The proof is analogous to that of Theorem 5.3 but relying on Theorem 3.2 instead of Theorem 2.4. The full proof follows.

Let $\mathcal{K}^{orc}_{\mathcal{A}}$ and $\mathcal{K}^{abs}_{\mathcal{A}}$ be the controllers computed through Theorems 3.2 and 5.2, respectively. The proof proceeds by induction on the fixed point iterations and by case analysis.

For the base case, by definition $\mathcal{K}^{orc}_0 = \mathcal{K}^{abs}_0 = \mathcal{A}$ and $R^{abs}_0 = R^{orc}_0 = \textit{Dangling}(\mathcal{K}_0)$.

For the inductive case, let i be a fixed point iteration. Assuming $\mathcal{K}^{orc}_{i-1} = \mathcal{K}^{abs}_{i-1}$ and $R^{orc}_{i-1} = R^{abs}_{i-1}$, we prove $\mathcal{K}^{orc}_i = \mathcal{K}^{abs}_i$ and $R^{orc}_i = R^{abs}_i$.

The equivalence $\mathcal{K}^{orc}_i = \mathcal{K}^{abs}_i$ follows because at the ith iteration ϕ^{orc}_p detects exactly the same transitions that are pruned by the orchestration synthesis algorithm.

For the equivalence $R^{orc}_i = R^{abs}_i$, we have $R^{orc}_i = R^{orc}_{i-1} \cup \{\, \vec{q} \mid (\vec{q} \rightarrow) \in T^{\Box}_{\mathcal{A}} \textit{ is uncontrollable in } \mathcal{K}^{orc}_i \,\} \cup \textit{Dangling}(\mathcal{K}^{orc}_i)$, and $R^{abs}_i = R^{abs}_{i-1} \cup \textit{Dangling}(\mathcal{K}^{abs}_i) \cup \{\, \vec{q} \mid (\vec{q} \xrightarrow{a}) = t \in T^{\Box}_{\mathcal{A}},\ \phi^{orc}_f(t, \mathcal{K}^{abs}_{i-1}, R^{abs}_{i-1}) = \textit{true}\}$.

Since $\mathcal{K}^{orc}_i = \mathcal{K}^{abs}_i$, also the dangling states are equivalent. It remains to prove that $\{\, \vec{q} \mid (\vec{q} \rightarrow) \in T^{\Box}_{\mathcal{A}} \textit{ is uncontrollable in } \mathcal{K}^{orc}_i \,\} = \{\, \vec{q} \mid (\vec{q} \rightarrow) = t \in T^{\Box}_{\mathcal{A}},\ \phi^{orc}_f(t, \mathcal{K}^{abs}_{i-1}, R^{abs}_{i-1}) = \textit{true}\}$. This equivalence is straightforward by the definition of ϕ^{orc}_f, Definition 3.1, and the inductive hypothesis. □

Theorem 5.5 (Abstract choreography synthesis)**.** *The choreography synthesis function of Definition 4.4 coincides with the instantiation of the abstract synthesis function of Definition 5.1, where given a generic transition $t = (\vec{q}, \vec{a}, \vec{q}')$, the predicates ϕ_p and ϕ_f are defined as follows, where $\hat{T}_{\mathcal{K},R}$ is defined in Definition 4.4:*

$$\begin{aligned}\phi_p^{cor}(t, \mathcal{K}, R) &= (t \textit{ is a request or offer}) \vee (\vec{q}' \in R) \vee t \in \hat{T}_{\mathcal{K},R}\\ \phi_f^{cor}(t, \mathcal{K}, R) &= \nexists\, (\vec{q} \xrightarrow{\vec{a}_2} \vec{q_2}') \in T^{\Box}_{\mathcal{K}} \,:\, (\vec{a}_2 \textit{ is a match}) \wedge (\vec{q}, \vec{q}_2' \notin \textit{Dangling}(\mathcal{K}))\\ &\qquad \wedge (\vec{a}_{(i)} = \vec{a_2}_{(i)} = \overline{a})\end{aligned}$$

Proof. The proof is analogous to that of Theorem 5.3 but relying on Theorem 4.4 instead of Theorem 2.4. The full proof follows.

Let $\mathcal{K}^{cor}_{\mathcal{A}}$ and $\mathcal{K}^{abs}_{\mathcal{A}}$ be the controllers computed through Theorems 4.4 and 5.2, respectively. The proof proceeds by induction on the fixed point iterations and by case analysis.

For the base case, by definition $\mathcal{K}^{cor}_0 = \mathcal{K}^{abs}_0 = \mathcal{A}$ and $R^{abs}_0 = R^{cor}_0 = \textit{Dangling}(\mathcal{K}_0)$.

For the inductive case, let i be a fixed point iteration. Assuming $\mathcal{K}^{cor}_{i-1} = \mathcal{K}^{abs}_{i-1}$ and $R^{cor}_{i-1} = R^{abs}_{i-1}$, we prove $\mathcal{K}^{cor}_i = \mathcal{K}^{abs}_i$ and $R^{cor}_i = R^{abs}_i$.

The equivalence $\mathcal{K}^{cor}_i = \mathcal{K}^{abs}_i$ follows because at the ith iteration ϕ^{cor}_p detects exactly the same transitions that are pruned by the choreography synthesis algorithm (and takes the same non-deterministic choices).

For the equivalence $R_i^{cor} = R_i^{abs}$, we have $R_i^{cor} = R_{i-1}^{cor} \cup \{\, \vec{q} \mid (\vec{q} \to) \in T_{\mathcal{A}}^{\Box} \textit{ is uncontrollable in } \mathcal{K}_i^{cor} \,\} \cup \textit{Dangling}(\mathcal{K}_i^{cor})$, and $R_i^{abs} = R_{i-1}^{abs} \cup \textit{Dangling}(\mathcal{K}_i^{abs}) \cup \{\, \vec{q} \mid (\vec{q} \xrightarrow{a}) = t \in T_{\mathcal{A}}^{\Box},\ \phi_f^{cor}(t, \mathcal{K}_{i-1}^{abs}, R_{i-1}^{abs}) = \textit{true} \}$.

Since $\mathcal{K}_i^{cor} = \mathcal{K}_i^{abs}$, also the dangling states are equivalent. It remains to prove that $\{\, \vec{q} \mid (\vec{q} \to) \in T_{\mathcal{A}}^{\Box} \textit{ is uncontrollable in } \mathcal{K}_i^{cor} \,\} = \{\, \vec{q} \mid (\vec{q} \to) = t \in T_{\mathcal{A}}^{\Box},\ \phi_f^{cor}(t, \mathcal{K}_{i-1}^{abs}, R_{i-1}^{abs}) = \textit{true} \}$. This equivalence is straightforward by the definition of ϕ_f^{cor}, Definition 4.3, and the inductive hypothesis. □