Science of Computer Programming 187 (2020) 102344

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Controller synthesis of service contracts with variability

Davide Basile [a,c,*], Maurice H. ter Beek [a], Pierpaolo Degano [b], Axel Legay [d], Gian-Luigi Ferrari [b], Stefania Gnesi [a], Felicita Di Giandomenico [a]

[a] ISTI–CNR, Pisa, Italy
[b] Università di Pisa, Italy
[c] Università di Firenze, Italy
[d] Université Catholique de Louvain, Belgium

ARTICLE INFO

Article history:
Received 8 June 2018
Received in revised form 18 July 2019
Accepted 29 October 2019
Available online 4 November 2019

Keywords:
Supervisory control theory
Contract automata
Service orchestrations
Variability
Behavioural variability

ABSTRACT

Service contracts characterise the desired behavioural compliance of a composition of services. Compliance is typically defined by the fulfilment of all service requests through service offers, as dictated by a given Service-Level Agreement (SLA). Contract automata are a recently introduced formalism for specifying and composing service contracts. Based on the notion of synthesis of the most permissive controller from Supervisory Control Theory, a safe orchestration of contract automata can be computed that refines a composition into a compliant one.

To model more fine-grained SLA and more adaptive service orchestrations, in this paper we endow contract automata with two orthogonal layers of variability: (i) at the structural level, constraints over service requests and offers define different configurations of a contract automaton, depending on which requests and offers are selected or discarded, and (ii) at the behavioural level, service requests of different levels of criticality can be declared, which induces the novel notion of semi-controllability. The synthesis of orchestrations is thus extended to respect both the structural and the behavioural variability constraints. Finally, we show how to efficiently compute the orchestration of all configurations from only a subset of these configurations. A prototypical tool supports the developed theory.

© 2019 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

1. Introduction

Service computing is a well-known paradigm for the creation, publication, discovery and orchestration of *services*, which are autonomous, platform-independent and reusable pieces of software that are loosely coupled into networks of collaborating end-user applications [1,2]. Services are usually programmed with little or no knowledge about clients and other services. They are created and published by possibly mutually distrusted organisations, and may have conflicting goals. Services have to *cooperate* to achieve overall goals and at the same time *compete* to perform specific tasks of their organisation. Ensuring reliability of a composite service is important, e.g. to avoid economic loss. Therefore, understanding and fulfilling a minimal number of behavioural obligations of services is crucial to determine whether the interactive behaviour is consistent with the requirements. Such obligations are usually dictated by a Service-Level Agreement (SLA). Recently, the Service

* Corresponding author at: Università di Firenze, Italy.
E-mail address: davide.basile@isti.cnr.it (D. Basile).

https://doi.org/10.1016/j.scico.2019.102344
0167-6423/© 2019 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

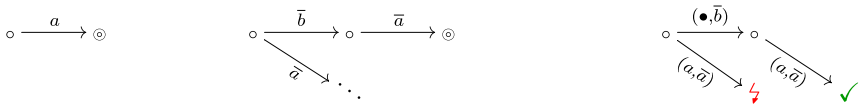


Fig. 1. Two automata (left and middle) and a possible composition of them (right).

Computing Manifesto [3] considers *service design* as one of the four emerging research challenges in service computing for the next 10 years, and calls for formal models supporting it:

> "Service systems have so far been built without an adequate rigorous foundation that would enable reasoning about them. [...] The design of service systems should build upon a formal model of services."

Service contracts offer means to formalise the externally observable behaviour of services in terms of *offers* of the service and *requests* by the service to be matched. The notion of *agreement* characterises compliance of (a composition of) contracts and it is based on the fulfilment of all service requests through corresponding service offers. Behaviour in agreement is implemented by an *orchestration* of services. Orchestrations must dynamically adapt to the discovery of new services, to service updates and to services that are no longer available. Moreover, a precise semantics of service contracts allows us to mechanically verify that an orchestration enjoys certain properties and to assess whether it satisfies a given SLA. We refer the reader to [4,5] for surveys on formal models of *service contracts*.

Contract automata [6] are one such formal model for service contracts. A contract automaton represents either a single service (in which case it is called a *principal*) or a multi-party composition of services. Each principal's goal is to reach an accepting state by matching its request actions (of the form a, b, ...) with corresponding offer actions (of the form $\overline{a}$, $\overline{b}$, ...) of other principals. Service interactions are implicitly controlled by an orchestrator synthesised from the principals, which directs them in such a way that only finite executions in agreement actually occur, i.e. such that each request action a is fulfilled by an offer action $\overline{a}$. Technically, such an orchestration is synthesised as the *most permissive controller* (mpc) known from Supervisory Control Theory (SCT) [7,8]. The goal of this paper is to present a richer framework of contract automata, which allows the user to model more fine-grained SLA and more adaptive service orchestrations.

Consider two automata interacting on a service action a, depicted in Fig. 1 (left and middle parts), and a composition of these automata in Fig. 1 (right part) that models two possibilities of fulfilling service request a from the leftmost automaton by matching it with a service offer $\overline{a}$ of the middle one (i.e. $(a, \overline{a})$). Assume that a must be matched with $\overline{a}$ to obtain an agreement, and that for some reason state $\lightning$ is to be avoided in favour of state ✓. In most automata-based formalisms, including the contract automata of [6,9], this is typically not allowed by the definition of composition and the resulting mpc is empty. Indeed, we would like to be able to express that a must eventually be matched, rather than always. In this paper, we introduce a type of contract automata in which it is possible to orchestrate the composition of the two automata on the left in such a way that the result is the automaton on the right *without state* $\lightning$, i.e. a is only matched with $\overline{a}$ *after* the occurrence of an unmatched service offer $\overline{b}$ of the middle automaton (i.e. $(\bullet, \overline{b})$).

Technically, we extend contract automata with action modalities to distinguish *necessary* from *permitted* service requests (borrowed from [9]) and with two novel orthogonal *variability* mechanisms. Necessary and permitted request actions differ in that the first *must* be fulfilled, while the second *may* also be omitted. The notions of necessary and permitted modalities stem from modal and deontic logic, which trace back to seminal work by Von Wright [10,11]. As in [9], we assume offer actions to always be permitted because a service contract may *always* withdraw its offers not needed to reach an agreement.

The first variability mechanism is defined inside service contracts, i.e. at the *behavioural* level, to declare necessary request actions to be either *urgent* or *lazy*. These modalities drive the orchestrator to fulfil *all the occurrences* of an urgent action, while it is required to fulfil *at least one occurrence* of lazy actions. The simple example above has no urgent action; the only necessary one is the lazy request a. Intuitively, the matching of a lazy request may be delayed whereas this is not the case for urgent requests.

The second variability mechanism concerns constraints that operate on the entire service contract and that are defined at the *structural* level. They are used to define different configurations, which is important because services are typically reused in configurations that vary over time and to adapt them to changing environments [3]. Configurations are characterised by which service actions are *mandatory* and which *forbidden*. The *valid* configurations are those respecting all the structural constraints. We follow the well-established paradigm of Software Product Line Engineering (SPLE), which aims at efficiently managing a family of highly (re)configurable systems to allow for mass customisation [12,13]. To compactly represent a *product line*, i.e. the set of *valid* product configurations, we use a so-called *feature constraint*, a propositional formula φ whose atoms are features [14–16] and we identify features as service actions (offers as well as requests).

To effectively use these two variability mechanisms, we refine the classical synthesis algorithm from SCT. We compute orchestrations, in the form of an mpc, of a single *valid* configuration, i.e. such that it includes all mandatory actions and none of the forbidden, besides fulfilling all necessary requests and the maximal number of permitted requests. Here maximal is to be understood in the sense that if the orchestration were to fulfil another permitted request, then this would cause one of the other requirements to no longer be fulfilled. An important technical result of this paper is that we can compute the orchestration of a product line *without* computing the mpc for each of its valid product configurations; it suffices to

compute a small selected subset of valid configurations. This guarantees efficiency and scalability of our novel framework of contract automata.

Summarising, the main contributions of this paper are as follows:

1. A novel formalism for service contracts, called *Featured Modal Contract Automata* (FMCA), which offers support for structural and behavioural variability not available in the literature.
2. The new notion of *semi-controllability* (related to lazy actions), which refines both those of controllability (related to permitted actions) and of uncontrollability (related to urgent actions) used in classical synthesis algorithms from SCT. This notion is fundamental to handle different service requests in the orchestration synthesis for FMCA.
3. A revised algorithm for synthesising an orchestration of services for a single valid product.
4. An algorithm for computing the orchestration of an entire product line by joining the orchestrations of a small selected subset of valid products. It is worth noting that the number of valid products is exponential in the number of features [17], thus only using few of them greatly improves performance.
5. The open-source prototypical tool FMCAT implementing our proposal.[1]

Outline In Section 2, we introduce our running example, a Hotel service product line, and briefly survey and evaluate our tool FMCAT. We formally define FMCA in Section 3, including composition, projection and controllability. In Section 4, we define the synthesis algorithms for FMCA. Two notions needed to efficiently compute the orchestration of an entire product line follow in Section 5, introducing so-called automata refinement, and Section 6, providing a means to (partially) order the products of a product line. In Section 7, we discuss related work, followed by our conclusions in Section 8. All proofs of the results presented in this paper are in Appendix A.

2. Motivating example: hotel reservation systems product line

A (software) *product line* is a set of (software-intensive) products in a portfolio of a manufacturer (or software house) that share common features and that are, ideally, built from a set of reusable (software) components by means of well documented variability [12,13]. A *feature* represents an abstract description of functionality and a feature model typically provides a description of products in terms of features: each *product* is thus uniquely characterised by a set of features. It is well known that a product can be represented by a Boolean assignment to the features (i.e. selected = true and discarded = false) and a feature model can thus be represented by what we call a *feature constraint* in this paper (i.e. a Boolean formula over the features). As a matter of fact, checking if a product respects the constraints expressed by a feature model, i.e. if it is *valid*, reduces to a Boolean satisfiability problem that has efficient solutions [14–16].

2.1. The hotel reservation systems product line and its feature constraint

We illustrate our approach with a product line of a basic franchise of hotel reservation systems. We consider three types of service contracts: a Hotel and two clients, called BusinessClient and EconomyClient, each with different service requests and offers, and with its own feature constraint. A system is a composition of these contracts, e.g. BusinessClient $\otimes$ Hotel $\otimes$ EconomyClient, and the feature constraint of the Hotel service product line is the conjunction of the feature constraints of the individual contracts. The 10 features concern requesting/offering a singleRoom or sharedRoom, privateBathroom or sharedBathroom, payment by card or cash, its confirmation by invoice or receipt, and the possibility of freeCancellation or noFreeCancellation. Furthermore, card and cash are alternative features and there are two additional constraints: invoice is selected in case cash is selected and sharedBathroom in case sharedRoom is. The resulting feature constraint φ of the Hotel service product line is as follows, where $\oplus$ denotes *exclusive or* and $\mathcal{F}$ the set of features:

$$\varphi = (\mathsf{card} \oplus \mathsf{cash}) \ \wedge \ \Big(\bigvee_{f \in \mathcal{F},\, f \notin \{\mathsf{card},\mathsf{cash}\}} f \Big) \wedge (\mathsf{cash} \rightarrow \mathsf{invoice}) \wedge (\mathsf{sharedRoom} \rightarrow \mathsf{sharedBathroom})$$

2.2. The valid products of the hotel reservation systems product line

As already said, a product p assigns a Boolean value to each feature, and the atoms of a feature constraint φ mapped by p to true (false, resp.) are called *mandatory* (*forbidden*, resp.).

Usually, in Software Product Line Engineering (SPLE), each feature is either selected or discarded to configure a product. In other words, all variability is resolved and the interpretation of the atoms of φ is *total*. Here, instead, we consider as valid products also so-called *sub-families* (in SPLE terms), which are defined by a *partial* assignment satisfying φ (see the comment after the 4 products below). This enables us to synthesise the orchestration of an entire product line by considering a few valid products only, rather than computing *all* valid ones. This is one of the main results of our paper, presented in Section 6.

[1] Available (with a short video tutorial) at https://github.com/davidebasile/FMCAT/.

Table 1
Classification of basic actions of FMCA distinguishing among **offers/requests** (||), permitted/necessary (▌) and *lazy/urgent* (|).

offers	**requests**		
permitted		necessary	
		lazy	*urgent*
$\overline{a}$	$a\Diamond$	$a\Box_\ell$	$a\Box_u$

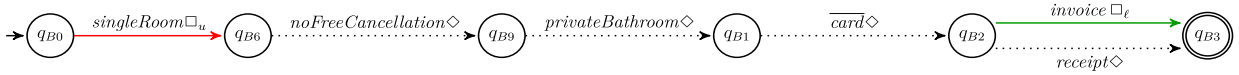


Fig. 2. The automaton for BusinessClient. (For interpretation of the colours in the figure(s), the reader is referred to the web version of this article.)

To give an idea of the impact of our improvement, it suffices to note that when no variability is left, the feature constraint φ characterises 288 product configurations, each representing a different instantiation of all its features. When the assignment is instead partial, as in our case, there are 4860 valid products, but we will show below that only 4 of them suffice to characterise the orchestration of the entire Hotel service product line. It is easy to imagine that for real-world feature models of up to millions of configurations the gain is considerable, confirming the scalability of our approach.

We partially order valid products by stipulating $p_1 \preceq p_2$ if and only if the sets of mandatory and forbidden features of p_2 are included in those of p_1. Accordingly, the *maximal products* are the valid products maximal in $\preceq$ (cf. Definition 17 in Section 6). The 4 maximal products of φ (out of 4860) have the following sets of mandatory (R) and forbidden (F) features.

R : [cash, invoice, sharedBathroom]; F : [card] (product P4854)

R : [cash, invoice]; F : [card, sharedRoom] (product P4857)

R : [card, sharedBathroom]; F : [cash] (product P4858)

R : [card]; F : [cash, sharedRoom] (product P4859)

A partial assignment that interprets the elements of R as true, those of F as false and all the others as "don't care" satisfies φ (cf. Definition 11 in Section 4). Indeed, whichever Boolean value replaces "don't care" leaves the formula satisfied.

2.3. FMCA: behavioural representations of the hotel reservation systems

As mentioned above, a feature constraint φ describes all the valid products of a product line. However, it has no immediate operational interpretation, in terms of the actions that the principals involved in a contract have to perform in order to achieve their goals. We address this operational aspect by using the aforementioned formalism of FMCA, which extend the model in [9]. The composition of two FMCA is itself an FMCA (see below) and it represents the composition of two service contracts.

Each FMCA $\mathcal{A}$ is a pair made of a special finite state automaton and a feature constraint φ, which is related to the automaton in the following way. The labels on the arcs of the automaton identify the actions for requests and offers, a subset of which corresponds to all features in φ. We say that $\mathcal{A}$ *respects* a product p whenever all features declared *mandatory* (*forbidden*, resp.) by p correspond to actions that are reachable (unreachable, resp.) from the initial state of $\mathcal{A}$ (cf. Definition 18 in Section 6).

Moreover, each automaton describes a *contract* where the actions corresponding to offers are overlined and those to requests are not. Offers are all considered *permitted*, while requests are either permitted or *necessary* at different degrees, namely *urgent* or *lazy* (cf. Table 1 and Definition 2 in Section 3). In this way, besides expressing that a request has to be matched (by an offer), one can also specify to what extent this must occur, i.e. whether the request must *always* (an *urgent* request) or *eventually* (a *lazy* request) be matched in a contract. The behaviour of $\mathcal{A}$ is the language it accepts.

We now specify the automata of the three service contracts introduced in the beginning of this section. Actually, we will illustrate different behavioural descriptions. Notationally, permitted actions label dotted arcs, suffixed by $\Diamond$, while urgent and lazy requests label (red and green in the pdf) full arcs and are suffixed by $\Box_u$ and $\Box_\ell$, respectively (cf. Table 1 and Fig. 2).

The automaton in Fig. 2 provides the behavioural description of service contract BusinessClient, in which a business client classifies the request for a room as urgent and the request for the invoice as lazy; all other actions have an obvious meaning and are permitted.

The automaton in Fig. 3 describes the Hotel behaviour. It has several points of non-determinism to comply with the requests of a BusinessClient as well as those of an EconomyClient. It also has cyclic behaviour enabling it to start new interactions with different clients. Finally, the hotel offers free breakfast (action $\overline{freebrk}\Diamond$) or requires to fill a captcha (action $captcha\Diamond$) to clients that pay by card and request an invoice (for the sake of example), although not all of these actions have corresponding features.

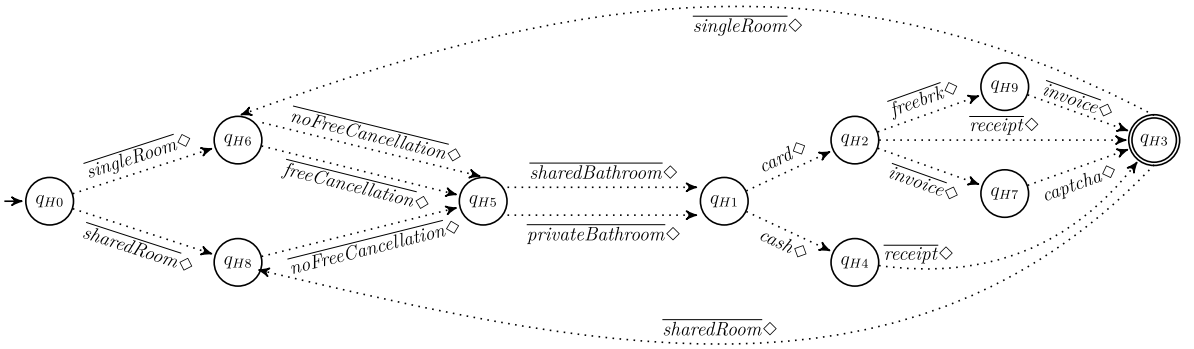


Fig. 3. The automaton for Hotel.



Fig. 4. The automaton for EconomyClient.

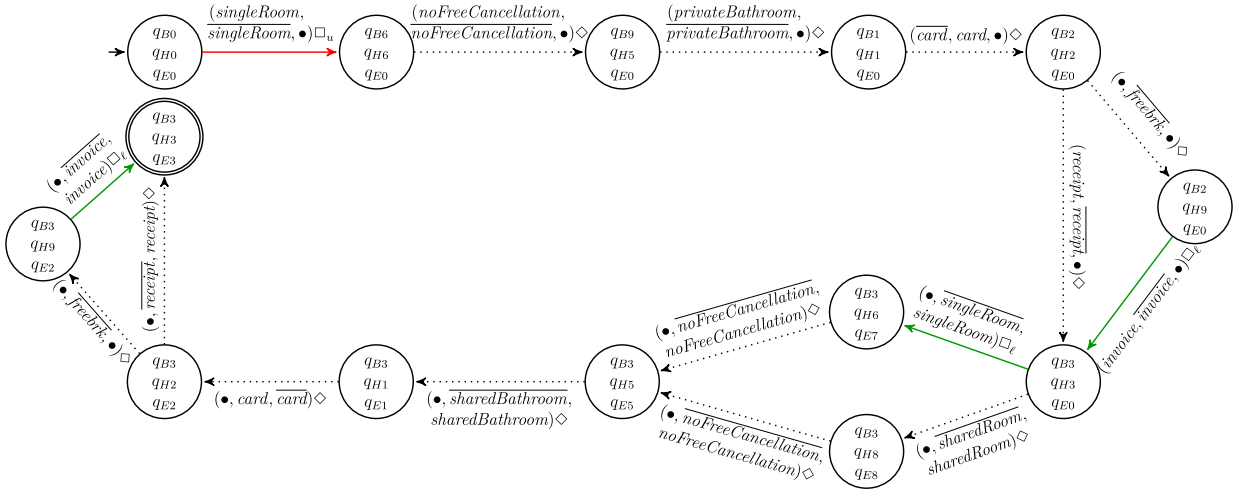


Fig. 5. Orchestration of composition BusinessClient ⊗ Hotel ⊗ EconomyClient of the three automata in Figs. 2, 3 and 4 for the product P4858.

The EconomyClient contract, depicted in Fig. 4, is similar to its business counterpart, but the single room request is *lazy*. Optionally a shared room is requested instead of a single room, and a shared bathroom is requested.

Composition of two FMCA is similar to standard automata composition, except that one's request actions have to be matched by the other's corresponding offers, if available. Inspecting the composition, we can determine whether services are compliant, in the sense that all requests are fulfilled (i.e., matched by offers). Additionally, we can further refine the composition so that only compliant behaviours are possible, as illustrated in Section 2.4.

2.4. Orchestrations of the hotel reservation systems product line

We intuitively show how to synthesise the orchestrations of the products of a product line, again in the form of an FMCA. The orchestration of a product p is defined as the largest sub-portion of $\mathcal{A}$ in *agreement*, i.e. all requests are matched by corresponding offers, and respecting p. Below, we discuss some examples of structural and behavioural variability.

Considering mandatory and forbidden feature constraints We first illustrate structural variability. We start by computing the orchestration of the composition $\mathcal{A} =$ BusinessClient ⊗ Hotel ⊗ EconomyClient for product P4858, i.e. the mandatory features card and sharedBathroom are selected whilst the forbidden feature cash is discarded. The orchestration is in Fig. 5, where all requests are matched by corresponding offers. No interleaving of actions from either component can occur (except for *freebrk*), because as soon as a request is enabled, there is a matching offer available.

The orchestration of $\mathcal{A}$ for product P4859 is the same as in Fig. 5, but with state $\langle q_{B3}, q_{H8}, q_{E8}\rangle$ and its incident transitions removed. Indeed, in this product the feature sharedRoom is discarded, and hence so is the transition labelled by action *sharedRoom* leading to $\langle q_{B3}, q_{H8}, q_{E8}\rangle$.

The Hotel service product line has only two maximal products in $\preceq$ that have non-empty orchestrations, namely P4858 and P4859. Instead, products P4854 and P4857 have empty orchestrations. Indeed, no cash payment is ever performed by either client (recall that for both cash and invoice are mandatory, while card is forbidden). In addition, invoice is unreachable (in the clients' contracts), because card is forbidden.

Note that mandatory and forbidden feature constraints are more demanding than imposing a request action to be necessary: the feature constraints of a product are global to a whole FMCA, while necessary requests are local to its transitions. More in detail, if an action corresponding to a mandatory feature is unreachable, then agreement is violated (see above). Instead, unreachable necessary requests do not spoil contract agreement.

The orchestration of a product line is the union (in automata theory terms) of the orchestrations of its products. A main result of this paper is based on the notion of *canonical* product, requiring it to have non-empty orchestration and to satisfy a further mild condition of its forbidden actions (cf. Definition 19 in Section 6). We can compute the orchestration $\mathcal{O}$ of a product line as the union of those of its canonical products, only, because all the orchestrations of the non-canonical products are sub-automata of $\mathcal{O}$. In our example, the orchestration $\mathcal{O}$ of the Hotel service product line is the union of the orchestrations of the two canonical products P4858 and P4859.

Considering necessary action requests We now continue by illustrating behavioural variability. We first discuss why offers are always permitted. If free breakfast were a necessary offer in Hotel, then we would have the unrealistic scenario in which the hotel contract rejects all clients' contracts. Indeed, no agreement would be reached because in the clients' contracts there is no free breakfast request to match the offer.

Next we consider a composition of the Hotel service with both types of clients and we show how urgent requests can be used for enforcing priorities among service requests. To the best of our knowledge, this is not present in any automata-based formalisms.

In the following, orchestrations always refer to product P4858. If EconomyClient is served *before* BusinessClient, i.e. (EconomyClient $\otimes$ Hotel) $\otimes$ BusinessClient (we will see later that $\otimes$ is non-associative), then both lazy matches

$$t_a = (\vec{q}_{E0,H0,B0}, (\mathit{singleRoom}, \overline{\mathit{singleRoom}}, \bullet)\Box_\ell, \vec{q}_{E7,H6,B0})$$

$$t_b = (\vec{q}_{E0,H0,B0}, (\mathit{sharedRoom}, \overline{\mathit{sharedRoom}}, \bullet)\Box_\ell, \vec{q}_{E8,H8,B0})$$

are present in the composition, so preventing the urgent match

$$t_c = (\vec{q}_{E0,H0,B0}, (\bullet, \overline{\mathit{singleRoom}}, \mathit{singleRoom})\Box_u, \vec{q}_{E0,H6,B6})$$

The absence of the urgent match causes the resulting orchestration to be empty. Intuitively, it should be the converse: the business client should be served before the economy client (i.e. t_c instead of t_a or t_b). In that case, i.e. (BusinessClient $\otimes$ Hotel) $\otimes$ EconomyClient, there does exist a non-empty orchestration respecting a business client's priority (cf. Fig. 5).

2.5. Modelling and analysing with the FMCAT tool

We implemented in Java a prototypical tool called FMCA Tool (FMCAT) that supports the definition of FMCA and that synthesises its orchestration in terms of its mpc. It is open source, and available at https://github.com/davidebasile/FMCAT, including the models used in this paper. FMCAT builds on CAT [18], a tool for contract automata, and it offers the functionalities described below. Our tool exploits FeatureIDE [19], an open-source framework for feature-oriented software development based on Eclipse, offering different feature model editing and management tools.

Modelling with FMCA decouples tasks of software engineers from tasks of experts in formal methods. Indeed, the syntactic description of a product line as a feature model (interpreted as feature constraint) and the description of its semantics as an automaton are separate concerns. By separating them, a software engineer can focus on designing a feature model and its valid products, leaving the task of specifying the operational semantics to an expert in formal modelling. Subsequently, these two aspects can be seamlessly integrated in the same FMCA, making it possible to detect inconsistencies between the syntactic and semantic levels of the same formalism. In general, a software designer would like to minimise the number of product configurations that do not admit safe behaviour (i.e. empty orchestrations). Indeed, valid products with empty orchestrations are such that the syntactic constraints provided by the feature model are not fulfilled by their behavioural descriptions. By inspecting the orchestration of the product line, one can detect all products with no safe behaviour (in the form of compliant services). If there are any, one either amends the feature model or their behavioural description so as to obtain non-empty orchestrations.

Consider once more the above example. The feature constraint φ of the Hotel service product line forces an invoice to be emitted in case of cash payments, and identifies cash and card payments as two alternative features. When the cash feature is selected, it requires also the invoice feature to be implemented while at the same time the card feature must not be implemented. However, in the behavioural description of the Hotel service in Fig. 3, an invoice is emitted only for credit

Table 2
Results about the expressiveness of lazy transitions.

	Num. states	Num. lazy transitions	Num. states of encoding
BusinessClient ⊗ Hotel ⊗ EconomyClient	343	143	$343 \times (2^{143}) + 1$
BusinessClient ⊗ Hotel	54	9	$54 \times (2^{9}) + 1$
EconomyClient ⊗ Hotel	47	10	$47 \times (2^{10}) + 1$
(BusinessClient ⊗ Hotel) ⊗ EconomyClient	340	144	$340 \times (2^{144}) + 1$
(EconomyClient ⊗ Hotel) ⊗ BusinessClient	278	106	$278 \times (2^{106}) + 1$

card payments, whereas a mere receipt is provided in case of cash payments. As a result, the maximal products P4854 and P4857 are not canonical because they require the cash feature to be implemented. One can fix this inconsistency between φ and the behavioural description by either removing the constraint cash $\rightarrow$ invoice from φ or by swapping $\overline{receipt}$ with $\overline{invoice}$ in the Hotel automaton.

Note that for detecting possible inconsistencies of this kind it suffices to only inspect the maximal products (here only 4), instead of all product configurations (here 288) with a total Boolean assignment.

2.6. Evaluation

To further corroborate our proposal, we provide an evaluation of the two main innovations proposed by FMCA: behavioural and structural variability.

Set-up of the evaluation We briefly evaluate FMCA using FMCAT (build June 2019). The evaluation was carried out on a machine with Processor Intel(R) Core(TM) i7-8700 CPU at 3.20 GHz, 3192 MHz, 6 Core(s), 12 Logical Processor(s) with 16 GB of RAM, running 64-bit Windows Version 10.0.17134 build 17134.

Evaluating behavioural variability First we evaluate behavioural variability, and in particular the gain in expressiveness due to the novel notion of semi-controllability. We will informally sketch an encoding of an FMCA into one without lazy transitions and estimate the differences in the state spaces of the two models.

As stated in Section 1, while permitted and urgent actions are related to the notions of controllability and uncontrollability of SCT [7,8], respectively, lazy actions are related to a novel notion called *semi-controllability*. We recall that a lazy request must eventually be matched, rather than always, as is the case for urgent requests. Indeed, a lazy request allows to model a frequent scenario in contracts where the satisfaction of a necessary request can be delayed, as in case of the room request of EconomyClient. While the synthesis algorithm cannot prune "bad" urgent transitions, it can prune "bad" lazy transitions as long as there exists another lazy transition where the same request is matched (cf. Section 4). Note that in general it is not possible to know a priori whether a transition is bad or not, because this depends on the given requirements to enforce (e.g. agreement, forbidden and required actions).

Hence, the encoding $\mathcal{A}'$ of an FMCA $\mathcal{A}$ containing n lazy transitions is the union of 2^n automata that are obtained by all possible combinations of pruning a subset of the n lazy transitions of $\mathcal{A}$ and turning the remaining lazy transitions into urgent. One of such combinations will prune exactly the same transitions pruned by the synthesis and thus the resulting orchestration of $\mathcal{A}'$ will contain the orchestration of the original automaton $\mathcal{A}$, among others that are not maximally permissive. In the worst-case scenario, the number of states of the encoding $\mathcal{A}'$ will be the number of states of the original automaton $\mathcal{A}$ multiplied by 2^n, plus an additional new initial state.

FMCAT has been equipped with a functionality that estimates the worst-case number of states of the encoding discussed above. Table 2 reports the results of the tool for the compositions discussed in this section. For each automaton we display the number of states, the number of lazy transitions and an estimation of the number of states of the encoding. As expected, the results show that in the worst case there is an exponential growth in the number of states of the automata, which quickly makes their analysis and usage non-tractable.

Concluding, the possibility offered by FMCA of primitively expressing necessary requests that must *eventually* be satisfied allows to reduce the number of states by an exponential factor with respect to other formalisms that only support controllable and uncontrollable actions.

Evaluating structural variability We now evaluate the structural variability of FMCA, and in particular the introduction of a partial order of products to reduce the number of configurations used to compute the orchestration (i.e. the most permissive controller) of the family. While the literature contains other attempts at synthesising the most permissive controller of a product line [20,21] (cf. Section 7), these require to compute the most permissive controller for each product (in which all variability has been resolved) without ordering the products. On the other hand, FMCA only considers canonical products. As shown in the remaining part of this section, ordering the products allows to improve the performance and reduce the state space. To evaluate the benefits introduced by FMCA, FMCAT was equipped with a functionality for computing the most permissive controller of a family with or without taking advantage of such partial order of products.

The results are displayed in Table 3. The table reports for each automaton the number of states and the time needed by FMCAT to compute it. It also reports the number of configurations for which an orchestration has been synthesised

Table 3
Results of the evaluation performed with FMCAT.

	Num. states	Time (ms)	Num. configurations	Used configurations
BusinessClient ⊗ Hotel ⊗ EconomyClient	343	71	–	–
BusinessClient ⊗ Hotel	54	9	–	–
EconomyClient ⊗ Hotel	47	6	–	–
(BusinessClient ⊗ Hotel) ⊗ EconomyClient	340	55	–	–
(EconomyClient ⊗ Hotel) ⊗ BusinessClient	278	22	–	–
$\mathcal{K}_{\text{BusinessClient}\otimes\text{Hotel}\otimes\text{EconomyClient}_{\text{P4858}}}$	14	875	1	P4858
$\mathcal{K}_{\text{BusinessClient}\otimes\text{Hotel}\otimes\text{EconomyClient}_{\text{P4859}}}$	13	580	1	P4859
$\mathcal{K}_{(\text{EconomyClient}\otimes\text{Hotel})\otimes\text{BusinessClient}_{\text{P4858}}}$	0	337	1	P4858
$\mathcal{K}_{(\text{BusinessClient}\otimes\text{Hotel})\otimes\text{EconomyClient}_{\text{P4858}}}$	14	598	1	P4858
$\mathcal{O}_{\text{BusinessClient}\otimes\text{Hotel}\otimes\text{EconomyClient}}$	28	3220	4	P4858, P4859
$\mathcal{O}_{(\text{EconomyClient}\otimes\text{Hotel})\otimes\text{BusinessClient}}$	0	950	4	–
$\mathcal{O}_{(\text{BusinessClient}\otimes\text{Hotel})\otimes\text{EconomyClient}}$	28	2796	4	P4858, P4859
$\mathcal{O}_{\text{BusinessClient}\otimes\text{Hotel}\otimes\text{EconomyClient}}$ without PO	55	129600	288	P86, P94, P182, P190
$\mathcal{O}_{(\text{EconomyClient}\otimes\text{Hotel})\otimes\text{BusinessClient}}$ without PO	0	60186	288	–
$\mathcal{O}_{(\text{BusinessClient}\otimes\text{Hotel})\otimes\text{EconomyClient}}$ without PO	55	130994	288	P86, P94, P182, P190

and the configurations for which the orchestration is non-empty. These last two columns are relevant when computing the orchestration of the family. The first six rows report the various compositions used in this section. The next four rows report the orchestration of these compositions for a specific product (denoted as $\mathcal{K}_{\mathcal{A}_p}$, where $\mathcal{A}$ is the composition and p is the product). These are the orchestrations discussed in Section 2.4. The next three rows report the orchestration of the family for the various compositions, by exploiting the partial order of products. In this case, the computation only requires to analyse the four canonical products. As previously discussed, products P4858 and P4859 are the only two canonical products used in the orchestration of the family. Finally, the remaining three rows display once again the orchestration of the whole family, but this time without exploiting the partial order. In this last case, computing the orchestration requires to analyse the 288 products originally generated by FeatureIDE, where those with non-empty orchestrations are:

$$\mathsf{R} : \mathcal{F} \setminus \mathsf{F}; \mathsf{F} : [\mathsf{cash}, \mathsf{receipt}, \mathsf{sharedRoom}, \mathsf{freeCancellation}] \quad \text{(product P86)}$$

$$\mathsf{R} : \mathcal{F} \setminus \mathsf{F}; \mathsf{F} : [\mathsf{cash}, \mathsf{receipt}, \mathsf{freeCancellation}] \quad \text{(product P94)}$$

$$\mathsf{R} : \mathcal{F} \setminus \mathsf{F}; \mathsf{F} : [\mathsf{cash}, \mathsf{sharedRoom}, \mathsf{freeCancellation}] \quad \text{(product P182)}$$

$$\mathsf{R} : \mathcal{F} \setminus \mathsf{F}; \mathsf{F} : [\mathsf{cash}, \mathsf{freeCancellation}] \quad \text{(product P190)}$$

These four products are indeed those generating the orchestration of the family without exploiting the partial order. In particular, the orchestrations for products P182 and P190 are identical to those of products P4858 and P4859. However, the orchestration of the family also includes two non-relevant products, namely P86 and P94. Indeed, the orchestrations of products P86 and P94 are included in those of products P182 and P190, and thus are not significant for characterising the orchestration of the family.

Concluding, the experiments empirically show that FMCA, and in particular their partial order of products, reduces both the state space of the orchestration of the family and the time needed to compute its orchestration.

3. Featured modal contract automata

We now formally define Featured Modal Contract Automata (FMCA), borrowing the following notation from [6,22]. In our framework, we distinguish basic actions belonging to the sets of *requests* $\mathsf{R} = \{a, b, c, \ldots\}$ and *offers* $\mathsf{O} = \{\overline{a}, \overline{b}, \overline{c}, \ldots\}$ where $\mathsf{R} \cap \mathsf{O} = \emptyset$. The alphabet of *basic actions* is defined as $\Sigma = \mathsf{R} \cup \mathsf{O} \cup \{\bullet\}$ where $\bullet \notin \mathsf{R} \cup \mathsf{O}$ is a distinguished element representing the *idle* move. We define the involution $co : \Sigma \to \Sigma$ such that (abusing notation) $co(\mathsf{R}) = \mathsf{O}$, $co(\mathsf{O}) = \mathsf{R}$ and $co(\bullet) = \bullet$.

Let $\vec{v} = (e_1, ..., e_n)$ be a vector of *rank* $r_v = n \geq 1$, and let $\vec{v}_{(i)}$ denote the ith element with $1 \leq i \leq r_v$. By $\vec{v}_1 \vec{v}_2 \cdots \vec{v}_m$ we denote the concatenation of m vectors $\vec{v}_i$. From now onwards, we stipulate that in an action vector $\vec{a}$ there is either a single offer or a single request, or a single pair of request-offer that match, i.e. there exists exactly i, j such that $\vec{a}_{(i)}$ is an offer and $\vec{a}_{(j)}$ is the complementary request; all the other elements of the vector contain the symbol $\bullet$, meaning that the corresponding principals remain idle. In the following, let $\bullet^m$ denote a vector of rank m, all elements of which are $\bullet$.

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

- $\vec{a} = \bullet^{n_1} \alpha \bullet^{n_2}$, $n_1, n_2 \geq 0$ $n_1 + n_2 + 1 = n$, then $\vec{a}$ is a *request (action) on* α if $\alpha \in \mathsf{R}$, whereas $\vec{a}$ is an *offer (action) on* α if $\alpha \in \mathsf{O}$
- $\vec{a} = \bullet^{n_1} \alpha \bullet^{n_2} co(\alpha) \bullet^{n_3}$, $n_1, n_2, n_3 \geq 0$, $n_1 + n_2 + n_3 + 2 = n$, then $\vec{a}$ is a *match (action) on* α, where $\alpha \in \mathsf{R} \cup \mathsf{O}$

Actions $\vec{a}$ and $\vec{b}$ are *complementary*, denoted by $\vec{a} \bowtie \vec{b}$, iff the following holds: (i) $\exists \alpha \in \mathsf{R} \cup \mathsf{O}$ s.t. $\vec{a}$ is either a request or an offer on α; (ii) $\vec{a}$ is an offer (request, resp.) on α implies that $\vec{b}$ is a request (offer, resp.) on $co(\alpha)$.

The actions and states of contract automata are vectors of basic actions and states of principals, respectively. The alphabet of an FMCA consists of vectors, each element of which intuitively records the execution of basic actions of principals in the contract.

An FMCA declares a product line of service contracts through (i) *permitted* and *necessary* transitions; and (ii) a feature constraint φ identifying all valid products (cf. Section 4). Modalities (i.e. permitted and necessary) classify requests and matches, while all offers are permitted. Offers and permitted requests reflect optional behaviour and can thus be discarded in the orchestration.

We further partition the set of necessary requests into *urgent* and *lazy* requests. Since these requests *must* be matched to reach an agreement among contracts, they express another layer of variability that specifies if a necessary request must *always* or *eventually* be matched in a contract. This extension leads to an increasing degree of controllability, as formally shown in Section 3.2. Table 1 in Section 2 depicts the different types of basic actions.

The definition of an FMCA follows, which is essentially a finite state automaton with an alphabet of basic actions appropriately partitioned, plus a propositional logic formula used to characterise the product line.

Definition 2 *(Featured modal contract automata).* Assume as given a finite set of states $\mathfrak{Q} = \{q_1, q_2, \ldots\}$. Then a *featured modal contract automaton* $\mathcal{A}$ of rank $n \geq 1$ is a tuple $\langle Q, \vec{q_0}, A^{\Diamond}, A^{\Box_u}, A^{\Box_\ell}, A^o, T, \varphi, F \rangle$, where

- $Q = Q_1 \times \cdots \times Q_n \subseteq \mathfrak{Q}^n$
- $\vec{q_0} \in Q$ is the initial state
- $A^{\Diamond}, A^{\Box_u}, A^{\Box_\ell} \subseteq \mathsf{R}$ are (pairwise disjoint) finite sets of permitted, urgent and lazy requests, resp.; we denote the set of requests by $A^r = A^{\Diamond} \cup A^{\Box_u} \cup A^{\Box_\ell}$
- $A^o \subseteq \mathsf{O}$ is the finite set of offers
- $T \subseteq Q \times A \times Q$, where $A = (A^r \cup A^o \cup \{\bullet\})^n$, is the set of transitions partitioned into *permitted* transitions $T^{\Diamond}$ and *necessary* transitions $T^{\Box}$, constrained as follows. Given $t = (\vec{q}, \vec{a}, \vec{q}\,') \in T$,
  - $\vec{a}$ is either a request or an offer or a match
  - $\forall i \in 1 \ldots n,\ \vec{a}_{(i)} = \bullet$ implies $\vec{q}_{(i)} = \vec{q}\,'_{(i)}$
  - $t \in T^{\Diamond}$ iff $\vec{a}$ is either a request or a match on $a \in A^{\Diamond}$ or an offer on $\overline{a} \in A^o$; otherwise $t \in T^{\Box}$
- φ is a propositional logic formula, whose atoms belong to $\mathsf{R} \cup \mathsf{O}$
- $F \subseteq Q$ is the set of final states

A *principal* FMCA (or just *principal*) has rank 1 and $A^r \cap co(A^o) = \emptyset$.

For brevity, unless stated differently, we assume a fixed FMCA $\mathcal{A} = \langle Q_{\mathcal{A}}, q\vec{o}_{\mathcal{A}}, A^{\Diamond}_{\mathcal{A}}, A^{\Box_u}_{\mathcal{A}}, A^{\Box_\ell}_{\mathcal{A}}, A^o_{\mathcal{A}}, T_{\mathcal{A}}, \varphi_{\mathcal{A}}, F_{\mathcal{A}} \rangle$ of rank n. Subscript $\mathcal{A}$ may be omitted when no confusion can arise. Moreover, if not stated otherwise, each operation f on one of the elements of the tuple (e.g. union $f(A^r)$) is intended to homomorphically act on its elements (e.g. $f(A^{\Diamond})$, $f(A^{\Box_u})$, and $f(A^{\Box_\ell})$). Also, let $T^{\Diamond \cup \Box}$ be a shorthand for $T^{\Diamond} \cup T^{\Box}$ and likewise for other transition sets. Finally, we call a transition t *request*, *offer* or *match* if its label is such. An FMCA recognises a language over (annotated) actions.

Definition 3. Let $\bigcirc \in \{\Diamond, \Box_u, \Box_\ell\}$. A step $(w, \vec{q}) \xrightarrow{\vec{a}\bigcirc} (w', \vec{q}')$ occurs iff $w = \vec{a} \bigcirc w'$, $w' \in (A \cup \{\bigcirc\})^*$ and $(\vec{q}, \vec{a}, \vec{q}\,') \in T^{\bigcirc}$. We write $\vec{q} \xrightarrow{\vec{a}\bigcirc}$ when w, w' and $\vec{q}\,'$ are immaterial and $(w, \vec{q}) \to (w', \vec{q}\,')$ when $\vec{a}\bigcirc$ is immaterial. Let $\to^*$ be the reflexive and transitive closure of transition relation $\to$. The language of $\mathcal{A}$ is $\mathscr{L}(\mathcal{A}) = \{\, w \mid (w, \vec{q_0}) \xrightarrow{w}{}^*(\varepsilon, \vec{q}),\ \vec{q} \in F \,\}$.

By an abuse of notation, the modalities can be attached to either basic actions or to their action vector (thus, e.g., $(a\Box_\ell, \overline{a}) \equiv (a, \overline{a})\Box_\ell$).

3.1. Composing FMCA

The FMCA operators of composition are crucial for generating (at binding time) an ensemble of services. By adding new services to an existing composition, it is possible to dynamically update the product line (i.e. both its feature model and its behaviour) and to synthesise, if possible, a composition satisfying all requests defined by the service contracts (cf. Section 4).

A set of FMCA is *composable* if and only if the conjunction of their feature constraints leads to no contradiction.

Definition 4 *(Composable).* A set $Set = \{\, \mathcal{A}_i \mid i \in 1 \ldots n \,\}$ of FMCA is *composable* iff $(\bigwedge_{\mathcal{A}_i \in Set}\ \varphi_{\mathcal{A}_i}) \not\models \mathit{false}$.

We now formally define our first (non-associative) composition operation, intuitively presented in Section 2. Its operands $\mathcal{A}_i$, $i \in 1 \ldots n$ are either principals or composite services. Intuitively, product composition $\otimes$ partially interleaves the actions of all operands, with one restriction: if two operands $\mathcal{A}_i$ and $\mathcal{A}_j$ are ready to execute two complementary actions (i.e.

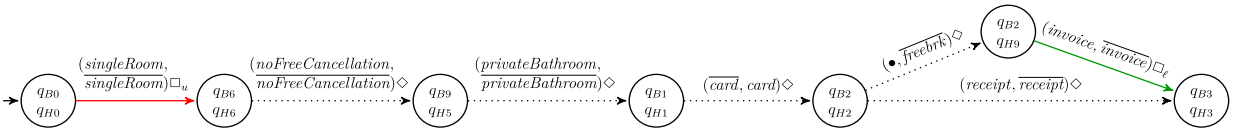


Fig. 6. Orchestration of composition BusinessClient ⊗ Hotel of the two automata in Figs. 2 and 3 for the canonical product P4859.

$\vec{a}_i \bowtie \vec{a}_j$) then only their match is allowed and their interleavings are prevented. Moreover, the generated match will inherit the modality of the request. This was shown in Section 2 and is further illustrated in Example 1 following the definition.

In detail, the transitions of the composite service are generated as follows.

Case (1) in Definition 5 generates match transitions starting from two transitions with complementary actions (i.e. $\vec{a}_i \bowtie \vec{a}_j$), coming from two operands $\mathcal{A}_i$ and $\mathcal{A}_j$. If, e.g., $(\vec{q}_j, \vec{a}_j, \vec{q}\,'_j) \in T^{\square}$, then the resulting match transition is marked as necessary (i.e. $(\vec{q}, \vec{c}, \vec{q}\,') \in T^{\square}$). If both complementary actions are permitted, then their resulting match transition t is marked as permitted. All other principals not involved in t remain idle.

Case (2) in Definition 5 generates all interleaved transitions if and only if no complementary actions can be executed from the composed source state $\vec{q}$. An operand $\mathcal{A}_i$ executes its transition $t = (\vec{q}_i, \vec{a}_i, \vec{q}\,'_i)$ while all others remain idle. The composed transition is marked as necessary (permitted) only if t is necessary (permitted, resp.). Note that condition $\vec{a}_i \bowtie \vec{a}_j$ excludes pre-existing match transitions of the operands being involved in new matches. Recall that we implicitly assume the set of labels of an FMCA of rank m to be $A \subseteq (A^r \cup A^o \cup \{\bullet\})^m$.

Definition 5 *(Composition).* Let $\mathcal{A}_i$ be a composable FMCA of rank r_i, $i \in 1, \ldots, n$, and let $\bigcirc \in \{\diamond, \square\}$. The *product composition* $\bigotimes_{i\in 1\ldots n} \mathcal{A}_i$ is the FMCA $\mathcal{A}$ of rank $m = \sum_{i\in 1\ldots n} r_i$, where

- $Q = Q_1 \times \cdots \times Q_n$, with $\vec{q}_0 = \vec{q}_{01} \cdots \vec{q}_{0n}$
- $A^r = \bigcup_{i\in 1\ldots n} A^r_i$, $A^o = \bigcup_{i\in 1\ldots n} A^o_i$,
- $T^{\bigcirc} \subseteq Q \times A \times Q$ s.t. $(\vec{q}, \vec{c}, \vec{q}\,') \in T^{\bigcirc}$ iff, when $\vec{q} = \vec{q}_1 \cdots \vec{q}_n \in Q$,
  1. either $\exists\, 1 \le i < j \le n$ s.t. $(\vec{q}_i, \vec{a}_i, \vec{q}\,'_i) \in T^{\bigcirc}_i$, $(\vec{q}_j, \vec{a}_j, \vec{q}\,'_j) \in T^{\bigcirc \cup \diamond}_j$, $\vec{a}_i \bowtie \vec{a}_j$ and
     $\begin{cases} \vec{c} = \bullet^u \vec{a}_i \bullet^v \vec{a}_j \bullet^z, \text{with } u = r_1 + \cdots + r_{i-1},\ v = r_{i+1} + \cdots + r_{j-1},\ z = r_{j+1} + \cdots + r_n,\ |\vec{c}| = m \\ \text{and } \vec{q}\,' = \vec{q}_1 \cdots \vec{q}_{i-1}\ \vec{q}\,'_i\ \vec{q}_{i+1} \cdots \vec{q}_{j-1}\ \vec{q}\,'_j\ \vec{q}_{j+1} \cdots \vec{q}_n \end{cases}$
  2. or $\exists\, 1 \le i \le n$ s.t. $(\vec{q}_i, \vec{a}_i, \vec{q}\,'_i) \in T^{\bigcirc}_i$ and
     $\begin{cases} \vec{c} = \bullet^u \vec{a}_i \bullet^v, \text{with } u = r_1 + \cdots + r_{i-1},\ v = r_{i+1} + \cdots + r_n, |\vec{c}| = m, \\ \vec{q}\,' = \vec{q}_1 \cdots \vec{q}_{i-1}\ \vec{q}\,'_i\ \vec{q}_{i+1} \cdots \vec{q}_n \text{ and } \forall j \ne i,\ 1 \le j \le n \text{ s.t. } (\vec{q}_j, \vec{a}_j, \vec{q}\,'_j) \in T^{\bigcirc \cup \diamond}_j,\ \vec{a}_i \bowtie \vec{a}_j \text{ does not hold} \end{cases}$
- $\varphi = \bigwedge_{i\in 1\ldots n} \varphi_i$
- $F = \{\, \vec{q}_1 \cdots \vec{q}_n \in Q \mid \vec{q}_i \in F_i,\ i \in 1 \ldots n \,\}$

Example 1. In Figs. 2 and 3, two principals discussed in Section 2 are depicted. A sub-portion of their composition BusinessClient ⊗ Hotel, which is their non-empty orchestration, is shown in Fig. 6. The outgoing transition $\vec{q}_{B0,H0} \xrightarrow{(singleRoom,\, \overline{singleRoom})\square_u}$ is an example of an urgent match between the urgent request $singleRoom\square_u$ of the first principal and the permitted offer $\overline{singleRoom}\diamond$ of the second. Recall that a match excludes any interleavings of principals in the composition. For example, in this composition neither of the transitions $\vec{q}_{B0,H0} \xrightarrow{(singleRoom\square_u,\, \bullet)}$ and $\vec{q}_{B0,H0} \xrightarrow{(\bullet,\, \overline{singleRoom}\diamond)}$ are allowed because of $(singleRoom\square_u, \bullet) \bowtie (\bullet, \overline{singleRoom}\diamond)$.

In the following, we assume every FMCA $\mathcal{A}$ of rank $r_{\mathcal{A}} > 1$ to be composed by FMCA with the composition operators described in this section. We now define the projection operator $\prod^i(\mathcal{A})$, which retrieves the principal $\mathcal{A}_i$ involved in $\mathcal{A}$ and identifies its original transitions and feature constraint. This operator is now formally defined (recall from Definition 2 that the set A^r is the union of all requests, permitted and necessary, while $T^{\diamond}$ and $T^{\square}$ partition the set T). Note that $\varphi = \bigwedge_{i\in 1\ldots n} \varphi_i$ as dictated by Definition 5.

Definition 6 *(Projection).* Let $\mathcal{A}$ be of rank n. The *projection* $\prod^i(\mathcal{A}) = \langle \prod^i(Q), \vec{q}_{0(i)}, \prod^i(A^{\diamond}), \prod^i(A^{\square_u}), \prod^i(A^{\square_\ell}), \prod^i(A^o), \prod^i(T), \prod^i(\varphi), \prod^i(F) \rangle$ on the ith principal, $i \in 1 \ldots n$, is s.t.

- $\prod^i(Q) = \{\, \vec{q}_{(i)} \mid \vec{q} \in Q \,\}$
- $\prod^i(A^r) = \{\, a \mid a \in A^r,\ (q, a, q') \in \prod^i(T) \,\}$
- $\prod^i(A^o) = \{\, \overline{a} \mid \overline{a} \in A^o,\ (q, \overline{a}, q') \in \prod^i(T) \,\}$
- $\prod^i(T^{\diamond}) = \{\, (\vec{q}_{(i)}, \vec{a}_{(i)}, \vec{q}\,'_{(i)}) \mid ((\vec{q}, \vec{a}, \vec{q}\,') \in T^{\diamond} \wedge \vec{a}_{(i)} \notin \bullet) \vee ((\vec{q}, \vec{a}, \vec{q}\,') \in T^{\square} \wedge \vec{a}_{(i)} \in \mathsf{O}) \,\}$
- $\prod^i(T^{\square}) = \{\, (\vec{q}_{(i)}, \vec{a}_{(i)}, \vec{q}\,'_{(i)}) \mid ((\vec{q}, \vec{a}, \vec{q}\,') \in T^{\square} \wedge \vec{a}_{(i)} \in \mathsf{R}) \,\}$
- $\prod^i(\bigwedge_{i\in 1\ldots n} \varphi_i) = \varphi_i$

– $\prod^i(F) = \{\, \vec{q}_{(i)} \mid \vec{q} \in F \,\}$

The associative composition operator $\boxtimes$ is defined below on top of the operators $\otimes$ and $\prod$. First, the corresponding principals of the operands are extracted by $\prod$ and then they are recomposed all together in a single step by $\otimes$. This causes all pre-existing matches to be rearranged.

Definition 7 *(A-composition).* Let $\mathcal{A}_1, \mathcal{A}_2$ be two composable FMCA of rank m and n, resp., and let $I = \{\, \prod^i(\mathcal{A}_1) \mid 0 < i \leq m \,\} \cup \{\, \prod^j(\mathcal{A}_2) \mid 0 < j \leq n \,\}$. Then the *a-product composition* of $\mathcal{A}_1$ and $\mathcal{A}_2$ is $\mathcal{A}_1 \boxtimes \mathcal{A}_2 = \bigotimes_{\mathcal{A}_i \in I} \mathcal{A}_i$.

Note that $\boxtimes$ models a *dynamic* composition policy: new services joining composite services can intercept already matched actions.

Hence, by changing operators of composition or the order of composition different composite FMCA can be obtained, as exemplified below.

Example 2. Consider again the example from Section 2 and compositions (EconomyClient $\otimes$ Hotel) $\boxtimes$ BusinessClient and EconomyClient $\otimes$ (Hotel $\otimes$ BusinessClient). In both, the transition $\vec{q}_{E0,H0,B0} \xrightarrow{(\bullet, \overline{singleroom}, singleroom\Box_u)}$ is allowed, whereas it is not in the composition (EconomyClient $\otimes$ Hotel) $\otimes$ BusinessClient, which has an empty orchestration.

3.2. Controllability

We base our algorithm for orchestration synthesis on that used to obtain the most permissive controller (mpc) in Supervisory Control Theory (SCT).

The purpose of SCT is to synthesise the mpc that enforces only "good" computations in finite state automata in which *forbidden* states are never traversed while *marked* (i.e. final) states represent the successful termination of a task. To this aim, SCT distinguishes *controllable* actions (those that the controller can disable) and *uncontrollable* actions (those that are always enabled), besides partitioning actions into *observable* and *unobservable* (obviously uncontrollable). If all actions are observable, then an mpc exists which never blocks a good computation, if any. Ideally, the actions that ruin an orchestration of service contracts should be removed by the synthesis algorithm. However, this can only be done for actions that are controllable in the orchestration.

Besides the classical controllable and uncontrollable actions, we introduce the new semi-controllable ones. Moreover, we call a transition controllable/uncontrollable if its action label is such.

All permitted actions are fully controllable. Urgent actions (i.e. requests and matches) are uncontrollable. Note that in these cases controllability and uncontrollability can be checked *locally* on a single transition.

Lazy actions (i.e. requests and matches) are *semi-controllable*. Semi-controllable transitions may lead to either controllable or uncontrollable transitions, depending on a *global* condition to be checked on the whole automaton resulting from a composition. If this condition is satisfied, then the semi-controllable transition is also controllable, otherwise it is uncontrollable. This condition states that the request (labelling the semi-controllable transition) must be matched in *at least one* transition in the automaton. Note that this is not the case for urgent actions that are uncontrollable *in every* transition in which they appear. The synthesis algorithm can therefore safely discard those lazy transitions leading to bad states (because they are controllable), provided that in the resulting automaton that specific request has been matched somewhere else. The following auxiliary definition will help in defining semi-controllability. It introduces *dangling* states, i.e. those unreachable or from which no final state can be reached.

Definition 8 *(Dangling state).* The state $\vec{q} \in Dangling(\mathcal{A})$ is *dangling* iff $\nexists\, w$ s.t. $\vec{q_0} \xrightarrow{w}{}^* \vec{q}$ or $\vec{q} \xrightarrow{w}{}^* \vec{q_f} \in F$.

The next definition classifies the transitions in an FMCA $\mathcal{A}$. Differently than what happens in standard SCT, all the transitions of FMCA are *observable*, because contracts declare the executions of a principal in terms of their requests and offers. Then we define the transitions that are controllable or not. We state the conditions that make a semi-controllable transition t controllable or not. Intuitively, t is controllable if in a given portion of $\mathcal{A}$ there exists a lazy match transition t', with source and target not dangling, and in both t and t' the *same* principal, in the *same* local state, does the *same* request. Otherwise, t is uncontrollable.

In what follows, we call $\mathcal{A}'$ sub-automaton of $\mathcal{A}$ (in symbols $\mathcal{A}' \subseteq \mathcal{A}$) whenever $\varphi_{\mathcal{A}'} = \varphi_{\mathcal{A}}$ and the other components of $\mathcal{A}'$ are included in the corresponding ones of $\mathcal{A}$.

Definition 9 *(Classifying transitions).* Let $t = (\vec{q}_1, \vec{a}_1, \vec{q}_1{}')$ be an (observable) transition in $\mathcal{A}$. 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 a request or a match on $a \in A^{\Box_u}$, then t is *uncontrollable* (in $\mathcal{A}$);



Fig. 7. A spurious composition $\mathcal{K}_{ill}$.

– If $\vec{a}_1$ is a request or a match on $a \in A^{\Box_\ell}$, then t is *semi-controllable* (in $\mathcal{A}$).
Moreover, given $\mathcal{A}' \subseteq \mathcal{A}$, if t is semi-controllable and $\exists t' = (\vec{q}_2, \vec{a}_2, \vec{q}_2{}') \in T^{\Box}_{\mathcal{A}'}$ s.t. $\vec{a}_2$ 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 $\mathcal{A}'$.

Example 3. Consider $\mathcal{A} = \mathsf{BusinessClient} \otimes \mathsf{Hotel} \otimes \mathsf{EconomyClient}$ from Section 2, its orchestration $\mathcal{K}_{\mathcal{A}_{\mathsf{P4858}}}$ in Fig. 5 and the trace

$$\vec{q}_{B0,H0,E0} \xrightarrow{(singleRoom, \overline{singleRoom}, \bullet)\Box_u} \vec{q}_{B6,H6,E0} \rightarrow^* \vec{q}_{B3,H3,E0} \xrightarrow{(\bullet, \overline{singleRoom}, singleRoom)\Box_\ell}$$

The semi-controllable transition $t = \vec{q}_{B0,H0,E0} \xrightarrow{(\bullet, \overline{singleRoom}, singleRoom)\Box_\ell}$ in $T_{\mathcal{A}}$ is controllable in $\mathcal{K}_{\mathcal{A}_{\mathsf{P4858}}}$ via $t' = \vec{q}_{B3,H3,E0} \xrightarrow{(\bullet, \overline{singleRoom}, singleRoom)\Box_\ell} \in T_{\mathcal{K}_{\mathcal{A}_{\mathsf{P4858}}}}$. Thus, t is safely removed in $\mathcal{K}_{\mathcal{A}_{\mathsf{P4858}}}$ because the corresponding request appears in another transition in a match (in this example t').

Example 4. Consider $\mathcal{A} = \mathsf{BusinessClient} \otimes \mathsf{Hotel}$ from Example 1 and its orchestration $\mathcal{K}_{\mathcal{A}_{\mathsf{P4858}}}$ displayed in Fig. 6. The semi-controllable transition $t = (\vec{q}_{B2,H2}, (invoice, \overline{invoice})\Box_\ell, \vec{q}_{B3,H7})$ in $T_{\mathcal{A}}$ is controllable in $\mathcal{K}_{\mathcal{A}_{\mathsf{P4858}}}$ via $t' = (\vec{q}_{B2,H9}, (invoice, \overline{invoice})\Box_\ell, \vec{q}_{B3,H3})$.

Conversely, consider the ill-formed orchestration $\mathcal{K}_{ill}$ in Fig. 7, obtained from $\mathcal{K}_{\mathcal{A}_{\mathsf{P4858}}}$ by removing states $\vec{q}_{B2,H9}$ and $\vec{q}_{B3,H7}$ and its incident transitions (including t'). Now no other matches for $invoice\Box_\ell$ are reachable. Hence, in this case, t is uncontrollable in $\mathcal{K}_{ill}$. Indeed, $\mathcal{K}_{ill}$ cannot be synthesised starting from $\mathcal{A}$ because t cannot be removed without violating the constraints in $\mathcal{A}$.

4. Controller synthesis for FMCA

We now extend the standard synthesis algorithm of SCT for computing the mpc to also deal with the newly introduced semi-controllable actions. Of course, with only urgent and permitted actions, the standard synthesis of SCT is immediately applicable. Moreover, we want to synthesise an orchestration of services that satisfies the feature constraint. To this aim, the synthesis algorithm computes the *mpc* of a *valid product* of the product line.

We first recall from [9] the properties of *(modal) agreement* and of *(modal) safety* of FMCA. Intuitively, a trace is in agreement if it is made of matches and offer actions only. An FMCA is safe when all traces of its language are in agreement, and it admits agreement when at least one of its traces is such.

Definition 10 *(Modal agreement, modal safety).* Let $\bigcirc \in \{\Diamond, \Box_u, \Box_\ell\}$. A trace accepted by an FMCA is in *agreement* if it belongs to the set

$$\mathfrak{A} = \{\, w \in (\Sigma^n \bigcirc)^* \mid \forall i \text{ s.t. } w_{(i)} = \vec{a}\bigcirc,\ \vec{a} \text{ is a match or an offer},\ n > 1 \,\}$$

An FMCA $\mathcal{A}$ is *safe* iff $\mathscr{L}(\mathcal{A}) \subseteq \mathfrak{A}$; otherwise it is *unsafe*. Finally, if $\mathscr{L}(\mathcal{A}) \cap \mathfrak{A} \neq \emptyset$, then $\mathcal{A}$ *admits agreement*.

Example 5. The FMCA in Fig. 6 admits agreement because the following trace belongs to its language and to the set $\mathfrak{A}$

$$(singleRoom, \overline{singleRoom})\Box_u (noFreeCancellation, \overline{noFreeCancellation})\Diamond$$
$$(privateBathroom, \overline{privateBathroom})\Diamond (\overline{card}, card)\Diamond (receipt, \overline{receipt})\Diamond$$

A few auxiliary definitions follow, which help to present our algorithm for synthesising an orchestration of an FMCA, viz. its maximal safe sub-portion.

Whilst generally products are total interpretations of a feature constraint, in a product we allow some atoms to have a "don't care" value by letting the interpretation function to be partial. We now define when a product is valid under a given, possibly partial, interpretation, and which are the basic actions that are mandatory and forbidden in the contract.

Definition 11 *(Valid products).* Let φ be a formula over $\mathsf{R} \cup \mathsf{O}$ and let $\mathcal{P} \ni p : \mathsf{R} \cup \mathsf{O} \Rightarrow \{true, false\}$ be a (partial) interpretation function.

Then $[\![\varphi]\!] = \{\, p \in \mathcal{P} \mid \varphi \models_p true \,\}$ is the set of all *valid products* of φ. Furthermore, let $\mathit{Mandatory}(p \in \mathcal{P}) = \{\, a \in \mathsf{R} \cup \mathsf{O} \mid p(a) = true \}$ and let $\mathit{Forbidden}(p \in \mathcal{P}) = \{\, a \in \mathsf{R} \cup \mathsf{O} \mid p(a) = false \,\}$.

Now, we formally define when a state is "bad", i.e. a transition outgoing from it cannot be blocked by the orchestrator, be it forbidden or a request that is not matched. This is because such transitions violate the constraint put by the predicate φ or they violate the agreement property. We also define when a transition of an FMCA is forced by its controller.

Definition 12 *(Uncontrollable disagreement).* Let $\mathcal{A}$ be an FMCA, $\mathcal{K} \subseteq \mathcal{A}$, and let $p \in [\![\varphi_{\mathcal{A}}]\!]$. A transition $t = \vec{q} \xrightarrow{\vec{a}} \in T_{\mathcal{A}}$ is *forced* in $\mathcal{A}$ by $\mathcal{K}$ iff (i) t is uncontrollable in $\mathcal{K}$ or (ii) there exist no $t' \in T_{\mathcal{K}}, t' \neq t$ with source $\vec{q} \notin F_{\mathcal{K}}$.

A state $\vec{q} \notin \mathit{Dangling}(\mathcal{K})$ is in *uncontrollable disagreement w.r.t. p by $\mathcal{K}$* iff there exists a trace $\vec{q}_{\mathcal{A}} \xrightarrow{w} {}^{*}\vec{q}_{1\mathcal{A}}$ made by forced transitions only, and either

1. an action occurring in w belongs to $\mathit{Forbidden}(p)$ or $w \notin \mathfrak{A}$; or
2. for all traces $\vec{q}_1 \xrightarrow{w'} {}^{*}\vec{q}_f \in F_{\mathcal{A}}$, condition (1) above holds for w'.

Example 6. State $\vec{q}_{B2,H2}$ of $\mathcal{A} = \mathsf{BusinessClient} \otimes \mathsf{Hotel}$ from Example 1 is in uncontrollable disagreement w.r.t. P4859 by $\mathcal{K}_{ill}$ (cf. Figs. 6 and 7) because the transition $(\vec{q}_{B2,H2}, (\mathit{invoice}, \overline{\mathit{invoice}})\Box_{\ell}, \vec{q}_{B3,H7}) \in T_{\mathcal{A}}$ is uncontrollable and forced in $\mathcal{K}_{ill}$ (cf. Example 4), and from state $\vec{q}_{B3,H7}$ the final state $\vec{q}_{B3,H3}$ can only be reached by executing the request action $(\bullet, \mathit{captcha}\Diamond)$ $(\notin \mathfrak{A})$.

Before giving the algorithm that synthesises an orchestration for a contract, we introduce the notion of mpc for an FMCA. Intuitively, given $\mathcal{A}$ and one of its valid products p, the mpc $\mathcal{K}$ of p is an FMCA that allows all traces of $\mathcal{A}$ in agreement with no states in uncontrollable disagreement w.r.t. p by $\mathcal{K}$, and blocks all the others. Moreover, all actions that are mandatory in p must occur in $\mathcal{K}$ whilst none of the forbidden ones. It turns out that $\mathcal{K}$ is unique up to language equivalence.

Definition 13 *(mpc of product).* Let $\mathcal{A}$, $\mathcal{K}$ be FMCA and let $p \in [\![\varphi_{\mathcal{A}}]\!]$. Then $\mathcal{K}$ is a *controller* of p iff the following hold

1. $\mathcal{K}$ is safe
2. $\mathit{Dangling}(\mathcal{K}) = \emptyset$
3. either $\mathscr{L}(\mathcal{K}) = \emptyset$ or $\forall a \in \mathit{Mandatory}(p)\ \exists w \in \mathscr{L}(\mathcal{K})$ s.t. a occurs in w
4. $\nexists w \in \mathscr{L}(\mathcal{K})$ s.t. w contains actions $\mathit{Forbidden}(p)$ or $\vec{q}_{0\mathcal{K}} \xrightarrow{w} {}^{*}\vec{q}_{\mathcal{K}}$, $\vec{q}_{0\mathcal{A}} \xrightarrow{w} {}^{*}\vec{q}_{\mathcal{A}}$ and $\vec{q}_{\mathcal{K}}$ is in uncontrollable disagreement w.r.t. p by $\mathcal{K}$.

A controller $\mathcal{K}$ of valid product p of $\mathcal{A}$ is the *most permissive (modal) controller (mpc)* iff for all controllers $\mathcal{K}'$ of p, $\mathscr{L}(\mathcal{K}') \subseteq \mathscr{L}(\mathcal{K})$ holds.

Example 7. All orchestrations discussed in Section 2 are the mpc of their corresponding service composition for either products P4858 or P4859.

The rest of this section presents an iterative algorithm that, given an FMCA $\mathcal{A}$, computes the mpc of one of its products p. Intuitively, the algorithm iteratively builds the set of bad states, i.e. those in uncontrollable disagreement, and it detects the bad transitions, i.e. those leading to such states. Recall that the bad states are those that cannot prevent a necessary request or a forbidden action to be eventually executed. Checking whether a transition is bad requires to inspect the whole automaton $\mathcal{A}$ to be able to decide whether a given transition is controllable or uncontrollable (cf. Definition 9). Apart from discarding the transitions forbidden by the product p, this is the main difference between our synthesis algorithm and the standard synthesis algorithm of [7], while—as expected—our algorithm still removes all requests that are not matched.

More precisely, we first let the initial mpc $\mathcal{K}_0$ be the whole $\mathcal{A}$, from which we remove its bad controllable transitions. The auxiliary set of bad states R_0 is also initialised with the source states of the bad uncontrollable transitions and with the dangling states of $\mathcal{K}_0$. At each iteration i, the algorithm prunes the controllable transitions with bad target and the uncontrollable transitions with bad source from $\mathcal{K}_i$. Moreover, R_i is updated by adding to R_{i-1} (i) the newly generated dangling states; (ii) the sources of uncontrollable transitions with bad target; and (iii) the sources of those transitions (of $\mathcal{A}$) not belonging to $\mathcal{K}_i$ that become uncontrollable (and bad) because of the pruning above, i.e. the sources of those transitions that turned from semi-controllable to uncontrollable because of pruning (cf. Definition 9). The algorithm terminates when no new updates are available, and the synthesised automaton, say $\mathcal{K}_n$, is the mpc of p. Of course, if the initial state is bad (in R_n) or some action mandatory in valid product p is unavailable in $\mathcal{K}_n$, then the mpc is empty.

Definition 14 *(Synthesis).* Let $\mathcal{A}$ be an FMCA and let $p \in [\![\varphi_{\mathcal{A}}]\!]$. The function $f : \mathit{FMCA} \times 2^{Q} \to \mathit{FMCA} \times 2^{Q}$ is iteratively defined as follows.

Let $\mathcal{K}_0 = \langle Q, \vec{q}_0, A^{\Diamond}, A^{\Box_u}, A^{\Box_\ell}, A^{o}, T \setminus \{ t \text{ controllable in } \mathcal{A} \mid t \text{ request } \vee\, a \in \mathit{Forbidden}(p) \}, \varphi_{\mathcal{A}}, F \rangle$, let $R_0 = \mathit{Dangling}(\mathcal{K}_0) \cup \{ \vec{q} \mid (\vec{q} \to) = t \in T^{\Box}_{\mathcal{A}} \text{ on a uncontrollable in } \mathcal{K}_0 \text{ and } (t \text{ request } \vee\, a \in \mathit{Forbidden}(p)) \}$.

Then $f(\mathcal{K}_{i-1}, R_{i-1}) = (\mathcal{K}_i, R_i)$, where

- $\mathcal{K}_i = \langle Q, \vec{q}_0, A^{\Diamond}, A^{\Box_u}, A^{\Box_\ell}, A^{o}, T_{\mathcal{K}_i}, \varphi_{\mathcal{A}}, F \rangle$, with

$$T_{\mathcal{K}_i} = T_{\mathcal{K}_{i-1}} \setminus (\{(\vec{q} \to \vec{q}\,') = t \in T_{\mathcal{K}_{i-1}} \mid t \text{ controllable in } \mathcal{K}_{i-1},\ \vec{q}\,' \in R_{i-1}\} \cup \{(\vec{q} \to) = t \in T_{\mathcal{K}_{i-1}} \mid t \text{ uncontrollable in } \mathcal{K}_{i-1},\ \vec{q} \in R_{i-1}\})$$

– $R_i = R_{i-1} \cup \{\vec{q} \mid (\vec{q} \to \vec{q}\,') \in T^{\Box}_{\mathcal{K}_i} \text{ uncontrollable in } \mathcal{K}_i,\ \vec{q} \notin R_{i-1},\ \vec{q}\,' \in R_{i-1}\} \cup \{\vec{q} \mid (\vec{q} \to) \in T^{\Box}_{\mathcal{A}} \text{ lazy uncontrollable in } \mathcal{K}_i\} \cup \mathit{Dangling}(\mathcal{K}_i)$

The following property is immediate.

Proposition 1. *Given two FMCA $\mathcal{A}, \mathcal{A}'$ and two sets of states R, R', let $(\mathcal{A}, R) \leq (\mathcal{A}', R')$ if $T_{\mathcal{A}} \supseteq T_{\mathcal{A}'}$ and $R \subseteq R'$.*
The function f of Definition 14 is monotone w.r.t. $\leq$ and, stipulating that $\mathcal{K}_0$ is as in Definition 14, its unique fixed point is:

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

The definition of the mpc for a valid product p is now straightforward.

Definition 15 *(Computing mpc).* Let $(\mathcal{K}_n, R_n) = \mathit{sup}(\{f^n(\mathcal{K}_0, R_0) \mid n \in \mathbb{N}\})$ as in Proposition 1 and let $T' = \{t = \vec{q} \to \in \mathcal{K}_n \mid t \text{ controllable in } \mathcal{K}_n,\ \vec{q} \in R_n\}$. Then the mpc $\mathcal{K}_{\mathcal{A}_p}$ for the valid product p of $\mathcal{A}$ is:

$$\mathcal{K}_{\mathcal{A}_p} = \begin{cases} \langle\rangle & \text{if } \vec{q}_0 \in R_n \text{ or } \exists a \in \mathit{Mandatory}(p) \text{ s.t. } \forall t \in T_{\mathcal{K}_n} : t \text{ is not a transition on } a \\ \langle Q \setminus R_n, \vec{q}_0, A^{\Diamond}, A^{\Box_u}, A^{\Box_\ell}, A^{o}, T_{\mathcal{K}_n} \setminus T', F \setminus R_n \rangle & \text{otherwise} \end{cases}$$

We now prove the main result of this section.

Theorem 1 *(mpc for product). Let $\mathcal{A}$ be an FMCA and let $p \in [\![\varphi_{\mathcal{A}}]\!]$ be a valid product. The FMCA $\mathcal{K}_{\mathcal{A}_p}$ computed through Definition 15 is the mpc for the valid product p of $\mathcal{A}$.*

Once obtained the mpc, one can construct the controlled system through a standard synchronous composition of $\mathcal{A}$ and $\mathcal{K}_{\mathcal{A}_p}$ (and not through the operators in Definition 5). As a matter of fact, it is unnecessary to specify the controlled system, because in our framework the mpc *is* the orchestration itself, and the needed interactions between the orchestrator and the principals are left implicit (for a longer discussion, cf. [22]).

5. Refining FMCA

Based on the notion of controllability of Definition 9, we now define a refinement relation between FMCA in the classical sense such that a refined automaton (i.e. with less states and/or transitions) still preserves certain properties of interest of the original automaton. We will then use this notion to efficiently compute the orchestration of a given product line.

As for the standard modal refinement [23], we stipulate that an FMCA $\mathcal{A}_r$ refines an FMCA $\mathcal{A}$ when $T_{\mathcal{A}_r} \subseteq T_{\mathcal{A}}$. More precisely, all the uncontrollable transitions of $\mathcal{A}$ are also present in $\mathcal{A}_r$, and a subset of the controllable ones of $\mathcal{A}$ also belong to $\mathcal{A}_r$. In addition, we require that a semi-controllable transition that is controllable in $\mathcal{A}$ must be present in $\mathcal{A}_r$, if it turns out to be uncontrollable there (this means that all the transitions making it controllable in $\mathcal{A}$ are not present in $\mathcal{A}_r$, as dictated by Definition 9).

Definition 16 *(Refinement of FMCA).* Let $\mathcal{A}$ and $\mathcal{A}_r$ be two FMCA. Then $\mathcal{A}_r$ is a *pre-refinement* of $\mathcal{A}$ iff

– $\mathcal{A}_r \subseteq \mathcal{A}$
– $t \in T_{\mathcal{A}}$ is uncontrollable implies $t \in T_{\mathcal{A}_r}$
– $t \in T_{\mathcal{A}}$ is semi-controllable and $\nexists t' \in T_{\mathcal{A}_r}$ s.t. t is controllable via t' (in $\mathcal{A}_r$) implies $t \in T_{\mathcal{A}_r}$

Finally, $\mathcal{A}_r$ is a *refinement* of $\mathcal{A}$, in symbols $\mathcal{A}_r \sqsubseteq \mathcal{A}$, if it is obtained from a pre-refinement of $\mathcal{A}$ by removing all the dangling nodes and the transitions they share.

Example 8. Consider $\mathcal{K}_{\mathcal{A}_{\mathrm{P4859}}}$ in Fig. 6 and the automaton $\mathcal{K}'$ obtained by adding to $\mathcal{K}_{\mathcal{A}_{\mathrm{P4859}}}$ transition $t = (\vec{q}_{B2,H2}, (\mathit{invoice}, \overline{\mathit{invoice}})\Box_\ell, \vec{q}_{B3,H7})$, state $\vec{q}_{B3,H7}$ and transition $(\vec{q}_{B3,H7}, (\overline{\mathit{freebrk}}\Diamond, \bullet), \vec{q}_{B3,H3})$.

Then the refinement $\mathcal{K}_{\mathcal{A}_{\mathrm{P4859}}} \sqsubseteq \mathcal{K}'$ holds because $t \in T_{\mathcal{K}'}$ is semi-controllable in $\mathcal{K}_{\mathcal{A}_{\mathrm{P4859}}}$ and controllable via $(\vec{q}_{B2,H9}, (\mathit{invoice}, \overline{\mathit{invoice}})\Box_\ell, \vec{q}_{B3,H3})$.

Finally, consider again $\mathcal{K}_{ill}$ from Example 4 in Fig. 7; in this case it holds that $\mathcal{K}_{ill} \not\sqsubseteq \mathcal{K}'$ because $t \in T_{\mathcal{K}'}$, $t \notin \mathcal{K}_{ill}$ but t is uncontrollable in $\mathcal{K}_{ill}$.

The next theorem states that the mpc of valid product p of $\mathcal{A}$ produces the largest refinement of the principals in $\mathcal{A}$ guaranteeing that there exists an agreement among the parties. Intuitively, if a permitted action does not spoil the overall agreement, then it will be available in the composition of services.

Theorem 2 *(Largest refinement). Let $\mathcal{A} = \bigotimes_{i\in I} \mathcal{A}_i$ be a composition of principals $\mathcal{A}_i$, let p be a valid product of $\mathcal{A}$, let $\mathcal{K}_{\mathcal{A}_p} \neq \langle\rangle$ be its mpc computed through Definition 14 and let $\forall i \in I : \Pi_i(\mathcal{K}_{\mathcal{A}_p}) = \mathcal{A}_{r_i}$ be its principals. Then:*

$$\forall i \in I : \mathcal{A}_{r_i} \sqsubseteq \mathcal{A}_i \tag{1}$$

$$\mathcal{K}' \neq \langle\rangle \text{ controller of } p \text{ of } \mathcal{A} \text{ implies } \forall i \in I : \Pi_i(\mathcal{K}') \sqsubseteq \mathcal{A}_{r_i} \tag{2}$$

Example 9. Let $\mathcal{K}$ be the orchestration of BusinessClient $\otimes$ Hotel in Fig. 6. Then $(\prod^1(\mathcal{K}) = \mathsf{Client}_p) \sqsubseteq$ BusinessClient.

6. Feature constraints and products

In Section 4, we presented the algorithm for synthesising an orchestration of service contracts for a *specific* product of a product line. The number of valid products of a product line is in general exponential in the number of features [17]; here we construct the orchestration of the *entire* product line only using a small selected subset of valid products.

All valid products $[\![\varphi]\!]$ of a product line can be partially ordered by (component-wise) set inclusion, providing us with the basis for computing the orchestration of a product line.

Definition 17 *(Partially ordering $[\![\varphi]\!]$).* Let $[\![\varphi]\!]$ be the set of valid products. Then $([\![\varphi]\!], \preceq)$ is a partially ordered set, where $p \preceq p'$ (in other words p is a *sub-product* of p' or p' is a *super-product* of p) iff component-wise

$$(\mathit{Mandatory}(p'), \mathit{Forbidden}(p')) \subseteq (\mathit{Mandatory}(p), \mathit{Forbidden}(p))$$

Moreover, the *maximal products* are the maximal elements of $\preceq$.

Example 10. The feature constraint φ in Section 2 has 4860 valid products (recall that we also consider partially interpreted products). Three exemplary products are the maximal product P4858 (cf. Section 2) with mandatory features {card, sharedBathroom} and forbidden feature {cash}, product P4829 with mandatory features {card, sharedBathroom} and forbidden features {cash, freeCancellation} and, lastly, product P4832 with mandatory features {card, sharedBathroom, privateBathroom} and forbidden feature {cash}. Thus follows P4829 $\preceq$ P4858 and P4832 $\preceq$ P4858.

6.1. FMCA respecting valid products

The validity of a product p is defined in logical terms, but it is convenient to see how it is reflected in the behaviour of an FMCA $\mathcal{A}$. Intuitively, all the mandatory actions in p correspond to executable transitions in $\mathcal{A}$ and no actions forbidden in p have executable counterparts in $\mathcal{A}$.

Definition 18 *(Respecting validity).* An FMCA $\mathcal{A}$ *respects* $p \in [\![\varphi_{\mathcal{A}}]\!]$ iff

1. $\forall a \in \mathit{Mandatory}(p)\ \exists (\vec{q}, \vec{a}, \vec{q}\,') \in T_{\mathcal{A}}$ s.t. $\vec{a}$ is an action on a and $\vec{q}, \vec{q}\,' \notin \mathit{Dangling}(\mathcal{A})$, and
2. $\forall b \in \mathit{Forbidden}(p)\ \nexists (\vec{q}, \vec{b}, \vec{q}\,') \in T_{\mathcal{A}}$ s.t. $\vec{b}$ is an action on b and $\vec{q}, \vec{q}\,' \notin \mathit{Dangling}(\mathcal{A})$.

We now prove that the partial order on valid products is such that if $\mathcal{A}$ respects one of them, then it respects all its super-products.

Theorem 3 *(Respecting validity is preserved by $\preceq$). Let $\mathcal{A}$ be an FMCA and let $p, p' \in [\![\varphi_{\mathcal{A}}]\!]$. Then:*

$$(\mathcal{A} \text{ respects } p \text{ and } p \preceq p') \text{ implies } \mathcal{A} \text{ respects } p'$$

Example 11. While P4858 and P4859 from Section 2 are maximal products featuring payments made by credit card, the maximal products P4854 and P4857 correspond to the Hotel product requiring payments by cash (and hence forbidding payments by credit card). Both products P4858 and P4859 are respected by BusinessClient $\otimes$ Hotel $\otimes$ EconomyClient (cf. Fig. 5), whereas P4854 and P4857 are not. Consider again product P4829 from Example 10. Since P4829 $\preceq$ P4858 and BusinessClient $\otimes$ Hotel $\otimes$ EconomyClient respects P4829, it also respects P4858. Moreover, every sub-product of P4854 or P4857 is not respected by all the given orchestrations (clients never pay cash).

The theorem above suggests an efficient procedure for singling out valid products respected by $\mathcal{A}$: visit the partially ordered set $([\![\varphi]\!], \preceq)$ in top-down fashion, starting from the maximal ones, and discard the subsets of products rooted in p if $\mathcal{A}$ does not respect p.

The following lemma relates (i) the existence of an mpc for a valid product p of $\mathcal{A}$ to (ii) the notion of respecting validity. While (i) implies (ii), the two notions imply each other whenever the set of actions mandatory in p is non-empty. Finally, Lemma 1(5) complements Theorem 3 and says that if a valid product p has a non-empty mpc, then there exists a p' such that $p' \preceq p$ and p' has a non-empty mpc.

Lemma 1. *Let $\mathcal{K}_{\mathcal{A}_p}$ be the mpc of a product p of $\mathcal{A}$. Then:*

$$\mathscr{L}(\mathcal{K}_{\mathcal{A}_p}) \neq \emptyset \text{ implies } \mathcal{K}_{\mathcal{A}_p} \text{ respects } p \tag{3}$$

$$(\mathcal{K}_{\mathcal{A}_p} \text{ respects } p \text{ and } \mathit{Mandatory}(p) \neq \emptyset) \text{ implies } \mathscr{L}(\mathcal{K}_{\mathcal{A}_p}) \neq \emptyset \tag{4}$$

$$(\mathscr{L}(\mathcal{K}_{\mathcal{A}_p}) \neq \emptyset \text{ and } I = \{p' \neq p \mid p' \preceq p\} \neq \emptyset) \text{ implies } \exists p' \in I : \mathscr{L}(\mathcal{K}_{\mathcal{A}_{p'}}) \neq \emptyset \tag{5}$$

Example 12. Consider BusinessClient ⊗ Hotel ⊗ EconomyClient from Section 2. The orchestrations of products P4858 and P4859 are non-empty, and by Lemma 1(3) it follows that both products are respected in their orchestration. Conversely, since both products have a non-empty mandatory set of actions and are respected by their corresponding mpc, by Lemma 1(4) it follows that their orchestrations are non-empty. Finally, consider products P4832 and P4829, which are both sub-products of P4858 (cf. Example 10). By Lemma 1(5) it follows that there exists a sub-product of P4858 with non-empty orchestration. Actually, both products P4832 and P4829 are such.

Note that in general the converse of Lemma 1(3) does not hold, because respecting validity ignores agreement, which is enforced by the mpc. A trivial counterexample is an $\mathcal{A}$ not admitting agreement and with products with no mandatory features.

The following lemma shows that the mpc $\mathcal{K}_{\mathcal{A}_p}$ for a valid product p of $\mathcal{A}$ is a refinement of the mpc $\mathcal{K}_{\mathcal{A}_{p'}}$ of a super-product p'. In other words, the partial order on valid products induces a refinement of controllers.

Lemma 2. *Let $\mathcal{A}$ be an FMCA and let $p, p' \in [\![\varphi_{\mathcal{A}}]\!]$. Then:*

$$p \preceq p' \text{ and } \mathscr{L}(\mathcal{K}_{\mathcal{A}_p}) \neq \emptyset \text{ implies } \mathcal{K}_{\mathcal{A}_p} \sqsubseteq \mathcal{K}_{\mathcal{A}_{p'}}$$

Example 13. Consider again $\mathcal{A}$ = BusinessClient ⊗ Hotel ⊗ EconomyClient from Section 2, the maximal product P4858, and its sub-product P4844 with mandatory features {card, sharedBathroom} and forbidden features {cash, sharedRoom}. Since the orchestration $\mathcal{K}_{\mathcal{A}_{\mathsf{P4844}}}$ is non-empty, by Lemma 2 it is a refinement of the orchestration $\mathcal{K}_{\mathcal{A}_{\mathsf{P4858}}}$ of $\mathcal{A}$ for product 4858 (depicted in Fig. 5). Indeed, $\mathcal{K}_{\mathcal{A}_{\mathsf{P4844}}}$ is obtained from $\mathcal{K}_{\mathcal{A}_{\mathsf{P4858}}}$ by removing state $\vec{q}_{B3,H8,E8}$ and its incident transitions.

An important consequence of Lemma 2 is that we can compute the orchestration $\mathcal{O}_{\mathcal{A}}$ of a product line $\mathcal{A}$ without generating the mpc for each of its valid products. Indeed, $\mathcal{O}_{\mathcal{A}}$ is the union of some controllers of certain valid products defined below, where we introduce the relation $\asymp$ that is clearly an equivalence relation. Note that in the following we refer to the set of valid products of $\mathcal{A}$ with non-empty controllers, partially ordered in the usual way.

Definition 19 *(Canonical products).* Let $\mathcal{A}$ be an FMCA, let $p, p' \in [\![\varphi_{\mathcal{A}}]\!]$, let $ME(\mathcal{A})$ be the set of maximal elements of $(\{p \mid \mathscr{L}(\mathcal{K}_{\mathcal{A}_p}) \neq \emptyset\}, \preceq)$, let

$$p \asymp p' \text{ iff } \mathit{Forbidden}(p) = \mathit{Forbidden}(p')$$

and let the *canonical products* $p_c \in CP(\mathcal{A})$ be the representatives of the equivalence classes of $ME(\mathcal{A})/\asymp$.

Intuitively, a canonical product represents all the maximal elements in $\preceq$ that have the same set of forbidden actions. Note that the information about mandatory actions is ignored by the equivalence relation $\asymp$ because we are only considering non-empty controllers.

Example 14. For $\mathcal{A}$ = BusinessClient ⊗ Hotel ⊗ EconomyClient from Section 2, as before, we obtain $ME(\mathcal{A})$ = {P4858, P4859}. Furthermore, the product P4844 of Example 13 has the same set of forbidden features as maximal product P4857, and P4844 $\asymp$ P4857. Finally, P4858 and P4859 are the canonical elements of their (singleton) equivalence class in $ME(\mathcal{A})/\asymp$.

The orchestration of the product line is now defined as the union of the orchestrations of the canonical products, where union is the standard operation on automata.

Definition 20 *(Orchestration of product line).* The orchestration of an FMCA $\mathcal{A}$ is defined as:

$$\mathcal{O}_{\mathcal{A}} = \bigcup_{p \in CP(\mathcal{A})} \mathcal{K}_{\mathcal{A}_p}$$

The canonical products fully characterise the mpc of each valid product in $\mathcal{A}$ as refinement of the orchestration of the product line, as guaranteed by the following theorem.

Theorem 4 *(Refinement of product line). Given the product line orchestration $\mathcal{O}_{\mathcal{A}}$ from Definition 20, it holds that:*

$$\forall\, p \in [\![\varphi_{\mathcal{A}}]\!] : \mathscr{L}(\mathcal{K}_{\mathcal{A}_p}) \neq \emptyset \;\; \textit{implies} \;\; \mathcal{K}_{\mathcal{A}_p} \sqsubseteq \mathcal{O}_{\mathcal{A}}$$

Example 15. Consider again $\mathcal{A} = \mathsf{BusinessClient} \otimes \mathsf{Hotel} \otimes \mathsf{EconomyClient}$ from Section 2. Its orchestration of the product line is $\mathcal{O}_{\mathcal{A}} = \mathcal{K}_{\mathcal{A}_{\mathsf{P4858}}} \cup \mathcal{K}_{\mathcal{A}_{\mathsf{P4859}}}$. Moreover $\mathscr{L}(\mathcal{K}_{\mathcal{A}_{\mathsf{P4844}}}) \neq \emptyset$ (cf. Example 13) and, by Theorem 4, $\mathcal{K}_{\mathcal{A}_{\mathsf{P4844}}} \sqsubseteq \mathcal{O}_{\mathcal{A}}$.

We now sketch an algorithm that incrementally computes the synthesis of the mpc for a valid product p of $\mathcal{A}$ (starting from the canonical products). First compute the mpc of the immediate super-products of p, then intersect all the resulting automata (with the standard intersection operation on automata) and, finally, apply to it the procedure defined in Definition 15 to obtain $\mathcal{K}_{\mathcal{A}_p}$. This algorithm is based on Theorem 4 and its correctness is guaranteed by the following theorem.

Theorem 5 *(Efficient mpc synthesis). Let $p \in \mathit{Depth}(\mathcal{A}, n)$, where*

$$\mathit{Depth}(\mathcal{A}, n) = \{\, p \mid p \in [\![\varphi_{\mathcal{A}}]\!],\ |\mathit{Mandatory}(p)| + |\mathit{Forbidden}(p)| = n \,\}$$

Then:

$$\mathcal{K}_{\mathcal{A}_p} \neq \langle\rangle \textit{ and } \mathit{Depth}(\mathcal{A}, n\text{-}1) \neq \emptyset \;\; \textit{implies} \;\; \mathcal{K}_{\mathcal{A}_p} \sqsubseteq \bigcap_{p \preceq p' \in \mathit{Depth}(\mathcal{A}, n\text{-}1)} \mathcal{K}_{\mathcal{A}_{p'}}$$

Example 16. Let $\mathcal{A}$ be as in Example 13, and recall that it has a non-empty orchestration of product P4844. We obtain $\mathit{Depth}(\mathcal{A}, 3) = \{\mathsf{P4858}, \mathsf{P4859}\}$, $\mathsf{P4844} \in \mathit{Depth}(\mathcal{A}, 4)$, $\mathsf{P4844} \preceq \mathsf{P4858}$, and $\mathsf{P4844} \preceq \mathsf{P4859}$. Theorem 5 guarantees that $\mathcal{K}_{\mathcal{A}_{\mathsf{P4844}}} \sqsubseteq \mathcal{K}_{\mathcal{A}_{\mathsf{P4858}}} \cap \mathcal{K}_{\mathcal{A}_{\mathsf{P4859}}}$ and thus $\mathcal{K}_{\mathcal{A}_{\mathsf{P4844}}} = \mathcal{K}_{\mathcal{A}_{\mathsf{P4858}}} \cap \mathcal{K}_{\mathcal{A}_{\mathsf{P4859}}}$.

The algorithm sketched above is more efficient than the standard ones (cf., e.g., [20]) that compute the controllers for *all* the valid products of a given product line, without taking advantage of the fact that they share some parts, as expressed by the relation $\sqsubseteq$.

7. Related work

Many formalisms exist for modelling and analysing service contracts. In this section, we first discuss some main differences between the most representative ones and FMCA, after which we discuss the basic differences with automata-based formalisms from Component-Based Software Engineering.

In [24–26], *behavioural contracts* of web services are described by CCS-like process algebrae, which model service features through input and output actions that synchronise. They have different, generally weaker notions of contract compliance than ours, e.g. only involving two parties. Choreographies were studied in [26] by seeing them as compound services, similar to our composed services, except that service competition was not considered. *Sessions* and *session types* [27–31] were introduced to reason over the behaviour of orchestrations and choreographies in terms of service interactions. Differently than in our proposal, none of the above papers considers different levels of criticality of service interactions (cf. [32] for a survey).

As anticipated, FMCA builds upon contract automata [6] that were used to study several issues arising in a composition of service contracts. In particular, the problem of circular dependencies among contracts was investigated by defining *weak agreement*. Roughly, this property considers acceptable traces where requests are recorded as debits that in the future are satisfied by the corresponding offers. It was studied for so-called competitive and collaborative contracts, with the results that generally safety is preserved in collaborative contracts and not in competitive ones. Weak agreement is suitably checked by algorithms for network flow optimisation. Contract automata were also related to two intuitionistic logics introduced for modelling circular dependencies among contracts. Orchestration and choreography of contract automata was investigated in [22], by identifying the conditions for dismissing the central orchestrator for both synchronous and asynchronous choreographies, thus avoiding the overhead due to the interactions between services and the orchestrator. Controller synthesis of contracts was recently extended to a real-time setting [33,34], where it is rendered as a winning strategy for timed contract games.

The definition of FMCA also borrows from two automata-based formalisms, namely Modal Transition Systems (MTSs) [35, 36] and Featured Transition Systems (FTSs) [37]. Our distinction into permitted and necessary transitions, borrowed from [9], was inspired by MTSs, while the explicit incorporation of feature constraints comes from FTSs. Compared to FMCA, neither of these two formalisms can explicitly handle *dynamic* composition and orchestration, by means of which new services that join composite services can intercept already matched actions. Such a compositionality is a basic characteristic that FMCA inherited from contract automata.

The accidentally homonym contract automata of [38] model generic *legal contracts* between two parties, expressed in natural language. Their states are tagged with deontic modalities in the form of obligations and permissions. These modalities are similar to our necessary and permitted requests, but they have no degree of criticality. FMCA target a different

domain, viz. multi-party service contracts. Moreover, [38] studies techniques for solving contract violation, while the focus of our compositional approach is on the synthesis of an orchestration of services.

Within the area of Component-Based Software Engineering, there are many formalisms for describing and composing components, similar to behavioural contracts. We briefly survey some of them. I/O automata [39] are input-enabled, i.e. in any state they are ready to receive any possible input from the environment, and composition is restricted to automata that do not share external actions. Therefore, they cannot model contracts that *compete* on offering or requesting the same service, a key feature of FMCA. Interface automata [40] are not input-enabled, rather they broadcast offers to every request. Their compatibility between interfaces requires that all offers are matched, dual to our agreement. Neither interface nor I/O automata have actions with different levels of criticality (other than concrete real-time constraints in their timed version). Modal I/O automata [41] combine the characteristics of interface and I/O automata with may and must modalities of MTSs. Some actions can thus be declared more critical than others, but still they differ from FMCA in the aspects mentioned above.

Supervisory Control Theory (SCT) was first applied to behavioural product line models in [20], where the `CIF 3` toolset was used to synthesise all the valid products starting from components and requirements rendered as automata. However, all the actions are controllable, unlike our approach; the controller of the product line requires computing those of all the valid products, and finally orchestration is not considered. Another work along this line is [21], where the standard synthesis algorithm was adapted to obtain a specific controller for each consistent product of the product line. Differently than in our proposal, the behaviour is specified by modal sequence diagrams and their actions are only controllable or uncontrollable. More recently, the interplay between SCT and product lines modelled as Priced Featured Automata was studied [42], where three-valued logic and partial-order reduction were used to greatly reduce the number of controllers required.

8. Conclusions and future work

We have proposed FMCA, an automata-based formalism for service contracts borrowing some aspects of Software Product Lines. According to this approach, services come in different configurations, or products. Two distinguished orthogonal ingredients of FMCA deal with the arising variability, which permit (i) different levels of criticality for service requests, defining how certain requests must be matched in a contract, and (ii) feature constraints over the product line, defining the valid products. With these ingredients, one can model service contracts with more adaptive service orchestrations and more fine-grained Service Level Agreement.

We have defined automata composition and their orchestration, i.e. a way to guarantee that both types of variability constraints are satisfied, besides fulfilling client's requests. The orchestration has the form of the most permissive controller of Supervisory Control Theory, and it has been obtained by extending the classical synthesis algorithm. Our novel notion of semi-controllability turned out to be crucial in handling different service requests.

We have defined a partial order on the valid products of a product line, through which its orchestration can be efficiently computed. Indeed, one only constructs the orchestration of the few maximal products in the partial order. Technically, this required to introduce a refinement relation on automata and to consider partially interpreted products. Typically, the maximal products of a product line are much less than its valid products that are exponentially many. Our proposal thus improves over the methods available in the literature, e.g. that in [20]. Also, one can only inspect these maximal products to find possible inconsistencies in the contract specifications.

We have implemented an open-source prototypical tool to support specification of a product line through the associated FMCA, and to compute the orchestrations of its valid products.

Future work includes a study of circular dependencies among services, by extending the results of [6]. It would also be interesting to investigate a choreographed coordination approach for FMCA, as was recently undertaken for a related contract automata formalism with different notions of semi-controllability [43]. Moreover, we plan to establish a correspondence between FMCA and Featured Transition Systems [37], in order to transfer some techniques of [44] for proving correctness properties.

Further work is needed to apply our theory to provide a formal framework for modelling and synthesising dynamic service product lines [45–53], i.e. services in which different configurations are reused to adapt to environments that change over time (including so-called late variability at runtime).

Finally, another research direction is enhancing service requests and offers with quantitative information. Reaching an agreement would then amount to finding the optimal trade-off among FMCA, each with a positive pay-off. This might lead to a formalisation of Quality of Service, allowing us to assess non-functional properties of services, like reliability or performance.

Appendix A. Proofs

We provide here all the proofs and some auxiliary results.

Proposition 1. *Given two FMCA $\mathcal{A}, \mathcal{A}'$ and two sets of states R, R', let $(\mathcal{A}, R) \leq (\mathcal{A}', R')$ if $T_{\mathcal{A}} \supseteq T_{\mathcal{A}'}$ and $R \subseteq R'$.*

The function f of Definition 14 is monotone w.r.t. $\leq$ and, stipulating that $\mathcal{K}_0$ is as in Definition 14, its unique fixed point is:

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

Proof. By Definition 14, at each iteration the set of states R_i can only increase while the set of transitions T_i can only decrease, thus f is monotone on the partial order $(\{(\mathcal{A}, R) \mid \mathcal{A} \in FMCA, R \subseteq \mathfrak{Q}\}, \leq)$. Moreover, this partial order is complete since the sets of transitions $T_{\mathcal{A}}$ and states R are finite, hence by the Knaster-Tarski theorem the least fixed point of f exists, is unique and can be computed in a finite number of iterations. □

To prove that the automaton computed through Definition 15 is indeed the mpc for the valid product p of $\mathcal{A}$, we use the following auxiliary lemma.

Lemma 3. *Let $\mathcal{A}$ be an FMCA, let $p \in [\![\varphi_{\mathcal{A}}]\!]$ be a valid product, let $\mathcal{K}_n$ be the FMCA (of p) and let R_n be the set of states computed through Proposition 1. Then:*

$$\forall \vec{q} \in R_n : \vec{q} \text{ is unreachable in } \mathcal{K}_n \text{ or } \vec{q}_0 \in R_n \tag{A.1}$$

$$\text{let } U_{p\mathcal{A}\mathcal{K}_n} = \{\vec{q} \mid \vec{q} \text{ in uncontrollable disagreement w.r.t. } p \text{ by } \mathcal{K}_n\} \text{then } U_{p\mathcal{A}\mathcal{K}_n} \cup Dangling(\mathcal{K}_n) = R_n \tag{A.2}$$

$$(\exists\, w \text{ not containing basic actions } a \in Forbidden(p) \text{ s.t.}$$
$$S_w = \{\vec{q}_{\mathcal{K}_n} \mid (\vec{q}_{0_{\mathcal{K}_n}} \xrightarrow{w_1}{}^* \vec{q}_{\mathcal{K}_n} \xrightarrow{w_2}{}^* \vec{q}_{f_{\mathcal{K}_n}}) \wedge (w = w_1 w_2 \in \mathscr{L}(\mathcal{A}) \cap \mathfrak{A})\} \neq \emptyset$$
$$\text{and } S_w \cap U_{p\mathcal{A}\mathcal{K}_n} = \emptyset) \text{ implies } S_w \cap R_n = \emptyset \tag{A.3}$$

Proof. We first prove (A.1). By contradiction, assume $\vec{q}_0 \notin R_n$ and there exists a sequence of states (traversed by a trace) $\rho = \vec{q}_0 \cdots \vec{q}'\vec{q}'' \cdots \vec{q}'''$ s.t. $\vec{q}'', \ldots, \vec{q}''' \in R_n$ and $\vec{q}' \notin R_n$. Let i be an iteration of the algorithm in Definition 14 s.t. $\vec{q}'' \in R_{i-1}$, $\vec{q}' \notin R_{i-1}$, and let $t = \vec{q}' \to \vec{q}''$ be the transition traversed in ρ. If t is controllable, then by Definition 14 it is removed in $\mathcal{K}_n$, a contradiction. Otherwise, if t is uncontrollable, then $\vec{q}'$ is added to R_i by Definition 14, and $\vec{q}' \in R_n$, a contradiction.

Next we prove (A.2). We start by proving $U_{p\mathcal{A}\mathcal{K}_n} \cup Dangling(\mathcal{K}_n) \subseteq R_n$. By contradiction, assume $\exists \vec{q} \in U_{p\mathcal{A}\mathcal{K}_n} \cup Dangling(\mathcal{K}_n)$ s.t. $\vec{q} \notin R_n$. By Definition 14, $Dangling(\mathcal{K}_n) \subseteq R_n$. Hence, $\vec{q} \notin Dangling(\mathcal{K}_n)$ and $\vec{q} \in U_{p\mathcal{A}\mathcal{K}_n}$. Thus, by Definition 12, there exists a trace w s.t. $\vec{q} \xrightarrow{w}{}^* \vec{q}_1$ is only executing forced transitions, and either (i) $w \notin \mathfrak{A}$ or w contains a basic action $a \in Forbidden(p)$ or (ii) $\forall \vec{q}_1 \xrightarrow{w'}{}^* \vec{q}_f \in F_{\mathcal{A}}$ and w' satisfies condition (i).

We first assume case (i) holds and let $w_1\vec{a}$ be a prefix of w such that $\vec{q} \xrightarrow{w_1}{}^* \vec{q}' \xrightarrow{\vec{a}}$ and either $\vec{a}$ is a request or it is forbidden in p. We first prove that $\vec{q}' \in R_0$. If $\vec{q}' \xrightarrow{\vec{a}}$ is controllable, then by Definition 12 it is the only outgoing transition from $\vec{q}'$, which is removed in $\mathcal{K}_0$ by Definition 14, and hence $\vec{q}' \in Dangling(\mathcal{K}_0) \subseteq R_0$. Similarly, if $\vec{q}' \xrightarrow{\vec{a}}$ is uncontrollable in $\mathcal{K}_0$, then $\vec{q}' \in R_0$ by Definition 14. Thus we have proved that $\vec{q}' \in R_0$ is reachable by only executing forced transitions from $\vec{q}$ in $\mathcal{A}$.

We now proceed by induction on the length of the trace $\vec{q} \xrightarrow{w_1}{}^* \vec{q}'$. For the base case, we have a transition $t = \vec{q} \to \vec{q}'$. Similarly to the previous reasoning, if t is controllable in $\mathcal{K}_1$ then by Definition 14, it is removed in $\mathcal{K}_1$ and, by Definition 12, t is the only outgoing transition from state $\vec{q}$ (i.e. it is forced) and hence $\vec{q} \in Dangling(\mathcal{K}_1)$ and $\vec{q} \in R_1 \subseteq R_n$, a contradiction. Otherwise, if t is uncontrollable in $\mathcal{K}_1$, then by Definition 14, $\vec{q} \in R_1 \subseteq R_n$, a contradiction. For the inductive step, we have $\vec{q} \to \vec{q}'' \to^* \vec{q}'$ s.t. $\vec{q}'' \in R_{i-1}$ and $\vec{q} \notin R_{i-1}$. By applying the same reasoning as for the base case we can conclude that $\vec{q} \in R_i \subseteq R_n$, a contradiction.

For case (ii), a final state $\vec{q}_f$ cannot be reached from $\vec{q}_1$ without executing either a request or a forbidden action, hence by hypothesis $\vec{q}_1$ cannot avoid to reach a final state without traversing a state $\vec{q}' \in R_0$ (otherwise a trace without requests and forbidden actions would exist). By Definition 14, there will be an iteration i s.t. $\vec{q}_1 \in R_i$, and $\vec{q} \in R_n$ by proceeding as for case (i).

We now prove $R_n \subseteq U_{p\mathcal{A}\mathcal{K}_n} \cup Dangling(\mathcal{K}_n)$. The proof is by induction on R_i. The base case is R_0. From Definition 14, it follows that $Dangling(\mathcal{K}_0) \subseteq R_0$. The last case is $\{\vec{q} \mid (\vec{q} \to) = t \in T^{\square}_{\mathcal{A}}$ on a uncontrollable in $\mathcal{K}_0$, $(t$ request $\vee\, a \in Forbidden(p))\}$, i.e. by Definition 12, in $U_{p\mathcal{A}\mathcal{K}_n}$. Note that if a transition t is uncontrollable in $\mathcal{K}_i$ for some i, then for all j, $i \leq j \leq n$, t is uncontrollable in $\mathcal{K}_j$.

For the inductive step, by the inductive hypothesis we know $R_{i-1} \subseteq U_{p\mathcal{A}\mathcal{K}_n} \cup Dangling(\mathcal{K}_{i-1})$ and we prove $R_i \subseteq U_{p\mathcal{A}\mathcal{K}_n} \cup Dangling(\mathcal{K}_i)$. We proceed again by the cases of Definition 14. The first case is $S = \{\vec{q} \mid (\vec{q} \to \vec{q}') \in T^{\square}_{\mathcal{K}_i}$ uncontrollable in $\mathcal{K}_i$, $\vec{q} \notin R_{i-1}$, $\vec{q}' \in R_{i-1}\} \cup \{\vec{q} \mid \vec{q} \to \in T^{\square}_{\mathcal{A}}$ lazy uncontrollable in $\mathcal{K}_i\}$. By Definition 12, the transitions used in S are forced in $\mathcal{A}$ because they are uncontrollable in $\mathcal{K}_i$ (hence in $\mathcal{K}_n$) and, by the inductive hypothesis, their target state is in $U_{p\mathcal{A}\mathcal{K}_n} \cup Dangling(\mathcal{K}_{i-1})$. It follows that $S \subseteq U_{p\mathcal{A}\mathcal{K}_n} \cup Dangling(\mathcal{K}_i)$. The last case is $Dangling(\mathcal{K}_i)$ that holds trivially.

Finally, we prove (A.3). By hypothesis, $S_w \cap Dangling(\mathcal{K}_n) = \emptyset$ and the thesis follows by (A.2). □

Theorem 1 *(mpc for product). Let $\mathcal{A}$ be an FMCA and let $p \in [\![\varphi_{\mathcal{A}}]\!]$ be a valid product. The FMCA $\mathcal{K}_{\mathcal{A}_p}$ computed through Definition 15 is the mpc for the valid product p of $\mathcal{A}$.*

Proof. We will prove that the algorithm always terminates, that $\mathcal{K}_{\mathcal{A}_p}$ is a controller of product p of $\mathcal{A}$ and, in particular, that it is the mpc of product p of $\mathcal{A}$. Termination of the algorithm is ensured by Proposition 1.

Next we prove that $\mathcal{K}_{\mathcal{A}_p}$ is a controller of p of $\mathcal{A}$, i.e.: $p \in [\![\varphi_{\mathcal{A}}]\!]$ (trivial); (1) $\mathcal{K}$ is safe, (2) $Dangling(\mathcal{K}) = \emptyset$ (trivial), (3) $\mathscr{L}(\mathcal{K}) = \emptyset$ or $\forall a \in Mandatory(p)\ \exists w \in \mathscr{L}(\mathcal{K})$ s.t. basic action a occurs in w, and (4) $\nexists w \in \mathscr{L}(\mathcal{K})$ s.t. w contains actions $a \in Forbidden(p)$ or $\vec{q}_{0_{\mathcal{K}}} \xrightarrow{w}{}^* \vec{q}_{\mathcal{K}}$, $\vec{q_{0_{\mathcal{A}}}} \xrightarrow{w}{}^* \vec{q_{\mathcal{A}}}$ and $\vec{q}_{\mathcal{K}}$ is in uncontrollable disagreement w.r.t. p by $\mathcal{K}$.

We first prove (1). Since $\mathcal{K}_{\mathcal{A}_p}$ is derived from $\mathcal{A}$ by pruning transitions, trivially $\mathscr{L}(\mathcal{K}_{\mathcal{A}_p}) \subseteq \mathscr{L}(\mathcal{A})$. To prove $\mathscr{L}(\mathcal{K}_{\mathcal{A}_p}) \subseteq \mathfrak{A}$, we have to show that no trace w' recognised by $\mathscr{L}(\mathcal{K}_{\mathcal{A}_p})$ contains a request $\vec{a}$. Note that the algorithm only prunes and never adds transitions and since in $\mathcal{K}_0$ all controllable requests are pruned, $\vec{a}$ cannot be a controllable request. By contradiction, assume $\vec{a}$ is an uncontrollable request, executed by a transition $\vec{q} \xrightarrow{\vec{a}\square}$, and w' is recognised by $\mathcal{K}_{\mathcal{A}_p}$. Then $\vec{q} \in R_0$ and thus $\vec{q} \in R_n$ by Definition 14. By Lemma 3(A.1), we have $\vec{q}_0 \in R_n$ and we reach the contradiction $\mathcal{K}_{\mathcal{A}_p} = \langle\rangle$.

We now prove (3). From the fact that none of the states of $\mathcal{K}_{\mathcal{A}_p}$ is dangling, (3) trivially holds by Definition 14.

We conclude by proving (4). Assume $\vec{q_0}_{\mathcal{K}} \xrightarrow{w'}{}^* \vec{q}'_{\mathcal{K}} \xrightarrow{\vec{a}}$, $w'\vec{a}$ prefix of w, with $\vec{a}$ on action $a \in Forbidden(p)$ holds. By Definition 14, $\vec{q}'_{\mathcal{K}} \xrightarrow{\vec{a}}$ is not controllable (otherwise it would have been removed) and hence $\vec{q}'_{\mathcal{K}} \in R_0 \subseteq R_n$. Then $\vec{q}'_{\mathcal{K}}$ is not a state of $\mathcal{K}_{\mathcal{A}_p}$, a contradiction. Assume that $(w, \vec{q_0}_{\mathcal{K}}) \xrightarrow{w}{}^* (\varepsilon, \vec{q}_{\mathcal{K}})$ and $(w, \vec{q_{0_{\mathcal{A}}}}) \xrightarrow{w}{}^* (\varepsilon, \vec{q_{\mathcal{A}}})$, with $\vec{q}$ in uncontrollable disagreement holds. Since $\mathcal{K}_{\mathcal{A}_p}$ is derived from $\mathcal{A}$, we have $\vec{q}_{0_{\mathcal{K}}} = \vec{q}_{0_{\mathcal{A}}}$ and $\vec{q}_{\mathcal{K}} = \vec{q}_{\mathcal{A}}$. By Lemma 3(A.2), $\vec{q}_{\mathcal{A}} \in R_n$ and since it is reachable from $\vec{q}_{0_{\mathcal{A}}}$, by Lemma 3(A.1), it must be the case that $\vec{q}_{0_{\mathcal{A}}} \in R_n$ and we reach the contradiction $\mathcal{K}_{\mathcal{A}_p} = \langle\rangle$.

Finally, it remains to prove that $\mathcal{K}_{\mathcal{A}_p}$ is the mpc. By contradiction, assume $\mathcal{K}'$ to be a controller of product p of $\mathcal{A}$ such that $\mathscr{L}(\mathcal{K}_{\mathcal{A}_p}) \subset \mathscr{L}(\mathcal{K}')$. Hence there must be a trace $w_1 \in \mathscr{L}(\mathcal{K}')$ s.t. $w_1 \notin \mathscr{L}(\mathcal{K}_{\mathcal{A}_p})$. Let S_{w_1} be the set of states traversed by $\mathcal{K}'$ to recognise w_1. Since $\mathcal{K}'$ is a controller, $S_{w_1} \cap U_{p\mathcal{A}\mathcal{K}'} = \emptyset$. By Lemma 3(A.3), $S_{w_1} \cap R_n = \emptyset$. Then, by Definition 14, all states in S_{w_1} are in $\mathcal{K}_{\mathcal{A}_p}$. Moreover, all transitions used for recognising w_1 are not requests nor forbidden because $\mathcal{K}'$ is a controller, and since $S_{w_1} \cap R_n = \emptyset$ no state in S_{w_1} is in any of $R_0, \ldots, R_n$. By Definition 14, these are all possible cases for which a transition is removed. Hence, all transitions used for recognising w are also in $\mathcal{K}_{\mathcal{A}_p}$, and it follows that $w_1 \in \mathscr{L}(\mathcal{K}_{\mathcal{A}_p})$, a contradiction. □

The proof of the next theorem makes use of the following auxiliary lemma.

Lemma 4. *Let $\mathcal{A}$ be an FMCA, let $p \in [\![\varphi_{\mathcal{A}}]\!]$ and let $\mathcal{K}_{\mathcal{A}_p} \neq \langle\rangle$ be its mpc computed through Definition 15. Then $\mathcal{K}_{\mathcal{A}_p} \sqsubseteq \mathcal{A}$.*

Proof. From Theorem 1 and Definition 14, we have $\mathcal{K}_{\mathcal{A}_p} \subseteq \mathcal{A}$ (component-wise inclusion with exception of φ) and, moreover, $Q_{\mathcal{K}} \cap Dangling(\mathcal{K}_{\mathcal{A}_p}) = \emptyset$.

It remains to prove that (1) all uncontrollable transitions of $\mathcal{A}$ and (2) all semi-controllable transitions t of $\mathcal{A}$ that are uncontrollable in $\mathcal{K}_{\mathcal{A}_p}$ (i.e. $\nexists t \in T_{\mathcal{K}_{\mathcal{A}_p}}$ s.t. t is controllable via t' in $\mathcal{K}_{\mathcal{A}_p}$), in both cases (1–2) with source $\vec{q} \in Q_{\mathcal{K}}$, are available in $\mathcal{K}_{\mathcal{A}_p}$. For (1), by contradiction, let $\vec{q} \in Q_{\mathcal{K}}$ and let $t = (\vec{q}, \vec{a}, \vec{q}\,')$ be s.t. t is uncontrollable in $\mathcal{A}$ and $t \notin T^{\square}_{\mathcal{K}}$. By Definition 14, $t \notin T^{\square}_{\mathcal{K}}$ only if $\vec{q} \in R_n$ and, by Lemma 3(A.2), $\vec{q} \in U_{p\mathcal{A}\mathcal{K}_n}$. If $\vec{q} \in U_{p\mathcal{A}\mathcal{K}_n}$, then by Lemma 3(A.1) $\mathcal{K}_{\mathcal{A}_p} = \langle\rangle$, a contradiction.

For (2), by contradiction, let $\vec{q} \in Q_{\mathcal{K}}$ and let $t = (\vec{q}, \vec{a}, \vec{q}\,')$ be s.t. t is controllable in $\mathcal{A}$, uncontrollable in $\mathcal{K}_{\mathcal{A}_p}$ and $t \notin T^{\square}_{\mathcal{K}}$. Hence t must be uncontrollable lazy in $\mathcal{K}_{\mathcal{A}_p}$, and $\vec{q} \in R_n$ by Definition 14. Finally, by Lemma 3(A.2), $\vec{q} \in U_{p\mathcal{A}\mathcal{K}_n}$ and, by Lemma 3(A.1), $\mathcal{K}_{\mathcal{A}_p} = \langle\rangle$, a contradiction. □

Theorem 2 *(Largest refinement). Let $\mathcal{A} = \bigotimes_{i \in I} \mathcal{A}_i$ be a composition of principals $\mathcal{A}_i$, let p be a valid product of $\mathcal{A}$, let $\mathcal{K}_{\mathcal{A}_p} \neq \langle\rangle$ be its mpc computed through Definition 14 and let $\forall i \in I : \Pi_i(\mathcal{K}_{\mathcal{A}_p}) = \mathcal{A}_{r_i}$ be its principals. Then:*

$$\forall i \in I : \mathcal{A}_{r_i} \sqsubseteq \mathcal{A}_i \tag{1}$$

$$\mathcal{K}' \neq \langle\rangle \text{ controller of } p \text{ of } \mathcal{A} \text{ implies } \forall i \in I : \Pi_i(\mathcal{K}') \sqsubseteq \mathcal{A}_{r_i} \tag{2}$$

Proof. Lemma 4 and Definition 6 suffice to prove (1).

To prove (2), assume by contradiction that for some i we have $\prod_i(\mathcal{K}') = \mathcal{A}_{r'_i} \not\sqsubseteq \mathcal{A}_{r_i}$. Then by (1) and Lemma 4, there must exist $t \in T^{\diamond}_{\mathcal{K}'} \setminus T^{\diamond}_{\mathcal{K}_{\mathcal{A}_p}}$. By Theorem 1, $\mathscr{L}(\mathcal{K}') \subseteq \mathscr{L}(\mathcal{K}_{\mathcal{A}_p})$, a contradiction. □

Theorem 3 *(Respecting validity is preserved by $\preceq$). Let $\mathcal{A}$ be an FMCA and let $p, p' \in [\![\varphi_{\mathcal{A}}]\!]$. Then:*

$$(\mathcal{A} \text{ respects } p \text{ and } p \preceq p') \text{ implies } \mathcal{A} \text{ respects } p'$$

Proof. Intuitively, p imposes more restrictions on respecting validity than p'. By contradiction, assume that p is respected by $\mathcal{A}$ and that p' is not respected by $\mathcal{A}$. By Definition 18, either one of the following cases must hold:

(1) $\exists a \in Mandatory(p')$ s.t. $\forall (\vec{q}, \vec{a}, \vec{q}\,') \in T_{\mathcal{A}} : \vec{a}$ is an action on $b \neq a$ or $\vec{q}, \vec{q}\,' \in Dangling(\mathcal{A})$. In this case, p is not respected by $\mathcal{A}$ because, by Definition 17, $Mandatory(p') \subseteq Mandatory(p)$, a contradiction.
(2) $\exists b \in Forbidden(p')$ s.t. $(\vec{q}, \vec{b}, \vec{q}\,') \in T_{\mathcal{A}}$, $\vec{b}$ is an action on b and $\vec{q}, \vec{q}\,' \notin Dangling(\mathcal{A})$. In this case, p is not respected by $\mathcal{A}$ because by Definition 17, $Forbidden(p') \subseteq Forbidden(p)$, a contradiction. □

Lemma 1. *Let $\mathcal{K}_{\mathcal{A}_p}$ be the mpc of a product p of $\mathcal{A}$. Then:*

$$\mathscr{L}(\mathcal{K}_{\mathcal{A}_p}) \neq \emptyset \text{ implies } \mathcal{K}_{\mathcal{A}_p} \text{ respects } p \tag{3}$$

$$(\mathcal{K}_{\mathcal{A}_p} \text{ respects } p \text{ and } \mathit{Mandatory}(p) \neq \emptyset) \text{ implies } \mathscr{L}(\mathcal{K}_{\mathcal{A}_p}) \neq \emptyset \tag{4}$$

$$(\mathscr{L}(\mathcal{K}_{\mathcal{A}_p}) \neq \emptyset \text{ and } I = \{p' \neq p \mid p' \preceq p\} \neq \emptyset) \text{ implies } \exists p' \in I : \mathscr{L}(\mathcal{K}_{\mathcal{A}_{p'}}) \neq \emptyset \tag{5}$$

Proof. For (3) assume by contradiction that p is not respected by $\mathcal{K}_{\mathcal{A}_p}$. Then, by Definition 18, either one of the following two cases must hold:

(1) $\exists a \in \mathit{Mandatory}(p)$ s.t. $\forall (\vec{q}, \vec{a}, \vec{q}\,') \in T_{\mathcal{K}_{\mathcal{A}_p}} : \vec{a}$ is an action on $b \neq a$ or $\vec{q}, \vec{q}\,' \in \mathit{Dangling}(\mathcal{K}_{\mathcal{A}_p})$. In this case, $\nexists\, w \in \mathscr{L}(\mathcal{K}_{\mathcal{A}_p})$ s.t. w contains a basic action a and thus, by Definition 13, $\mathcal{K}_{\mathcal{A}_p}$ is not an mpc, a contradiction.

(2) $\exists b \in \mathit{Forbidden}(p)$ s.t. $(\vec{q}, \vec{b}, \vec{q}\,') \in T_{\mathcal{K}_{\mathcal{A}_p}}$, $\vec{b}$ is an action on b and $\vec{q}, \vec{q}\,' \notin \mathit{Dangling}(\mathcal{K}_{\mathcal{A}_p})$. In this case, $\exists\, w_1 \vec{b} w_2 \in \mathscr{L}(\mathcal{K}_{\mathcal{A}_p})$ for some w_1, w_2 and, by Definition 13, $\mathcal{K}_{\mathcal{A}_p}$ is not an mpc, a contradiction.

For (4), by hypothesis $\exists a \in \mathit{Mandatory}(p)$ such that $(\vec{q}, \vec{a}, \vec{q}\,') \in T_{\mathcal{K}_{\mathcal{A}_p}}$ on a and $\vec{q}, \vec{q}\,' \notin \mathit{Dangling}(\mathcal{K}_{\mathcal{A}_p})$, and $\mathscr{L}(\mathcal{K}_{\mathcal{A}_p}) \neq \emptyset$ by Definition 8.

For (5), it suffices to note that a sub-product p' can be obtained by adding an action $a \notin \mathit{Mandatory}(p) \cup \mathit{Forbidden}(p)$ to either (i) $\mathit{Mandatory}\,(p')$ or (ii) $\mathit{Forbidden}(p')$ (the existence of such action a is guaranteed by hypothesis). Action a is either present or not in $\mathcal{K}_{\mathcal{A}_p}$. If action a is present, then the sub-product p' obtained through case (i) (i.e. $a \in \mathit{Mandatory}(p')$) is s.t. $\mathscr{L}(\mathcal{K}_{\mathcal{A}_{p'}}) \neq \emptyset$ by hypothesis (we are requiring an action that is present). If action a is not present, then the sub-product obtained through case (ii) (i.e. $a \in \mathit{Forbidden}(p')$) is s.t. $\mathscr{L}(\mathcal{K}_{\mathcal{A}_{p'}}) \neq \emptyset$ by hypothesis (we are forbidding an action that is not present). □

Lemma 2. *Let $\mathcal{A}$ be an FMCA and let $p, p' \in [\![\varphi_{\mathcal{A}}]\!]$. Then:*

$$p \preceq p' \text{ and } \mathscr{L}(\mathcal{K}_{\mathcal{A}_p}) \neq \emptyset \text{ implies } \mathcal{K}_{\mathcal{A}_p} \sqsubseteq \mathcal{K}_{\mathcal{A}_{p'}}$$

Theorem 4 *(Refinement of product line). Given the product line orchestration $\mathcal{O}_{\mathcal{A}}$ from Definition 20, it holds that:*

$$\forall\, p \in [\![\varphi_{\mathcal{A}}]\!] : \mathscr{L}(\mathcal{K}_{\mathcal{A}_p}) \neq \emptyset \text{ implies } \mathcal{K}_{\mathcal{A}_p} \sqsubseteq \mathcal{O}_{\mathcal{A}}$$

Proof. The statement follows immediately from Definition 20, Lemma 2, Lemma 1 and the hypothesis. □

Theorem 5 *(Efficient mpc synthesis). Let $p \in \mathit{Depth}(\mathcal{A}, n)$, where*

$$\mathit{Depth}(\mathcal{A}, n) = \{\, p \mid p \in [\![\varphi_{\mathcal{A}}]\!],\ |\mathit{Mandatory}(p)| + |\mathit{Forbidden}(p)| = n \,\}$$

Then:

$$\mathcal{K}_{\mathcal{A}_p} \neq \langle\rangle \text{ and } \mathit{Depth}(\mathcal{A}, n\text{-}1) \neq \emptyset \text{ implies } \mathcal{K}_{\mathcal{A}_p} \sqsubseteq \bigcap_{p \preceq p' \in \mathit{Depth}(\mathcal{A}, n\text{-}1)} \mathcal{K}_{\mathcal{A}_{p'}}$$

Proof. Straightforward from Lemma 2. □

References

[1] D. Georgakopoulos, M.P. Papazoglou (Eds.), Service-Oriented Computing, MIT Press, Cambridge, MA, USA, 2008.

[2] Q. Yi, X. Liu, A. Bouguettaya, B. Medjahed, Deploying and managing Web services: issues, solutions, and directions, VLDB J. 17 (3) (2008) 537–572, https://doi.org/10.1007/s00778-006-0020-3.

[3] 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, M. Papazoglou, A service computing manifesto: the next 10 years, Commun. ACM 60 (4) (2017) 64–72, https://doi.org/10.1145/2983528.

[4] M.H. ter Beek, A. Bucchiarone, S. Gnesi, Web service composition approaches: from industrial standards to formal methods, in: Proceedings of the 2nd International Conference on Internet and Web Applications and Services, ICIW'07, IEEE, 2007.

[5] M. Bartoletti, T. Cimoli, R. Zunino, Compliance in behavioural contracts: a brief survey, in: C. Bodei, G.L. Ferrari, C. Priami (Eds.), Programming Languages with Applications to Biology and Security, in: LNCS, vol. 9465, Springer, 2015, pp. 103–121.

[6] D. Basile, P. Degano, G.L. Ferrari, Automata for specifying and orchestrating service contracts, Log. Methods Comput. Sci. 12 (4) (2016), https://doi.org/10.2168/LMCS-12(4:6)2016.

[7] P.J. Ramadge, W.M. Wonham, Supervisory control of a class of discrete event processes, SIAM J. Control Optim. 25 (1) (1987) 206–230, https://doi.org/10.1137/0325013.

[8] C.G. Cassandras, S. Lafortune, Introduction to Discrete Event Systems, Springer, New York, NY, USA, 2006.

[9] D. Basile, F. Di Giandomenico, S. Gnesi, P. Degano, G.L. Ferrari, Specifying variability in service contracts, in: Proceedings of the 11th International Workshop on Variability Modelling of Software-Intensive Systems, VaMoS'17, ACM, 2017, pp. 20–27.
[10] G.H. von Wright, An Essay in Modal Logic, Studies in Logic and the Foundations of Mathematics, North-Holland, Amsterdam, The Netherlands, 1951.
[11] G.H. von Wright, Deontic logic, Mind 60 (237) (1951) 1–15, https://doi.org/10.1093/mind/LX.237.1.
[12] K. Pohl, G. Böckle, F.J. van der Linden, Software Product Line Engineering: Foundations, Principles, and Techniques, Springer, Berlin, Germany, 2005.
[13] S. Apel, D.S. Batory, C. Kästner, G. Saake, Feature-Oriented Software Product Lines: Concepts and Implementation, Springer, Berlin, Germany, 2013.
[14] M. Mannion, Using first-order logic for product line model validation, in: G.J. Chastek (Ed.), Proceedings of the 2nd International Software Product Lines Conference, SPLC'02, in: LNCS, vol. 2379, Springer, 2002, pp. 176–187.
[15] D.S. Batory, Feature models, grammars, and propositional formulas, in: J.H. Obbink, K. Pohl (Eds.), Proceedings of the 9th International Software Product Lines Conference, SPLC'05, in: LNCS, vol. 3714, Springer, 2005, pp. 7–20.
[16] K. Czarnecki, A. Wąsowski, Feature diagrams and logics: there and back again, in: Proceedings of the 11th International Conference on Software Product Lines, SPLC'07, IEEE, 2007, pp. 23–34.
[17] D. Benavides, S. Segura, A. Ruiz-Cortés, Automated analysis of feature models 20 years later: a literature review, Inf. Syst. 35 (6) (2010) 615–636, https://doi.org/10.1016/j.is.2010.01.001.
[18] D. Basile, P. Degano, G.L. Ferrari, E. Tuosto, Playing with our CAT and communication-centric applications, in: E. Albert, I. Lanese (Eds.), Proceedings of the 36th International Conference on Formal Techniques for Distributed Objects, Components, and Systems, FORTE'16, in: LNCS, vol. 9688, Springer, 2016, pp. 62–73.
[19] J. Meinicke, T. Thüm, R. Schröter, F. Benduhn, T. Leich, G. Saake, Mastering Software Variability with FeatureIDE, Springer, Cham, Switzerland, 2017.
[20] M.H. ter Beek, M. Reniers, E. de Vink, Supervisory controller synthesis for product lines using CIF 3, in: T. Margaria, B. Steffen (Eds.), Proceedings of the 7th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation, ISoLA'16, in: LNCS, vol. 9952, Springer, 2016, pp. 856–873.
[21] M. Cordy, J.-M. Davril, J. Greenyer, E. Gressi, P. Heymans, All-at-once-synthesis of controllers from scenario-based product line specifications, in: Proceedings of the 19th International Conference on Software Product Line, SPLC'15, ACM, 2015, pp. 26–35.
[22] D. Basile, P. Degano, G.L. Ferrari, E. Tuosto, Relating two automata-based models of orchestration and choreography, J. Log. Algebraic Methods Program. 85 (3) (2016) 425–446, https://doi.org/10.1016/j.jlamp.2015.09.011.
[23] M.H. ter Beek, A. Fantechi, S. Gnesi, F. Mazzanti, Modelling and analysing variability in product families: model checking of modal transition systems with variability constraints, J. Log. Algebraic Methods Program. 85 (2) (2016) 287–315, https://doi.org/10.1016/j.jlamp.2015.11.006.
[24] G. Castagna, N. Gesbert, L. Padovani, A theory of contracts for web services, ACM Trans. Program. Lang. Syst. 31 (5) (2009) 19:1–19:61, https://doi.org/10.1145/1538917.1538920.
[25] L. Acciai, M. Boreale, G. Zavattaro, Behavioural contracts with request-response operations, Sci. Comput. Program. 78 (2) (2013) 248–267, https://doi.org/10.1016/j.scico.2011.10.007.
[26] C. Laneve, L. Padovani, An algebraic theory for Web service contracts, Form. Asp. Comput. 27 (4) (2015) 613–640, https://doi.org/10.1007/s00165-015-0334-2.
[27] R. Bruni, I. Lanese, H. Melgratti, E. Tuosto, Multiparty sessions in SOC, in: D. Lea, G. Zavattaro (Eds.), Proceedings of the 10th International Conference on Coordination Models and Languages, COORDINATION'08, in: LNCS, vol. 5052, Springer, 2008, pp. 67–82.
[28] K. Honda, N. Yoshida, M. Carbone, Multiparty asynchronous session types, in: Proceedings of the 35th Symposium on Principles of Programming Languages, POPL'08, ACM, 2008, pp. 273–284.
[29] M. Dezani-Ciancaglini, U. de'Liguoro, Sessions and session types: an overview, in: C. Laneve, J. Su (Eds.), Proceedings of the 6th International Conference on Web Services and Formal Methods, WS-FM'09, in: LNCS, vol. 6194, Springer, 2010, pp. 1–28.
[30] G. Castagna, M. Dezani-Ciancaglini, L. Padovani, On global types and multi-party sessions, Log. Methods Comput. Sci. 8 (1:24) (2012) 1–45, https://doi.org/10.2168/LMCS-8(1:24)2012.
[31] J. Michaux, E. Najm, A. Fantechi, Session types for safe Web service orchestration, J. Log. Algebraic Program. 82 (8) (2013) 282–310, https://doi.org/10.1016/j.jlap.2013.05.004.
[32] H. Hüttel, I. Lanese, V. Vasconcelos, L. Caires, M. Carbone, P. Deniélou, D. Mostrous, L. Padovani, A. Ravara, E. Tuosto, H.T. Vieira, G. Zavattaro, Foundations of session types and behavioural contracts, ACM Comput. Surv. 49 (1) (2016) 3:1–3:36, https://doi.org/10.1145/2873052.
[33] D. Basile, M.H. ter Beek, A. Legay, L. Traonouez, Orchestration synthesis for real-time service contracts, in: M.F. Atig, S. Bensalem, S. Bliudze, B. Monsuez (Eds.), Proceedings of the 12th International Conference on Verification and Evaluation of Computer and Communication Systems, VECoS'18, in: LNCS, vol. 11181, Springer, 2018, pp. 31–47.
[34] D. Basile, M.H. ter Beek, A. Legay, Timed service contract automata, Innov. Syst. Softw. Eng., https://doi.org/10.1007/s11334-019-00353-3.
[35] A. Antonik, M. Huth, K.G. Larsen, U. Nyman, A. Wąsowski, 20 years of modal and mixed specifications, Bull. Eur. Assoc. Theor. Comput. Sci. 95 (2008) 94–129.
[36] J. Křetínský, 30 years of modal transition systems: survey of extensions and analysis, in: L. Aceto, G. Bacci, G. Bacci, A. Ingólfsdóttir, A. Legay, R. Mardare (Eds.), Models, Algorithms, Logics and Tools, in: LNCS, vol. 10460, Springer, 2017, pp. 36–74.
[37] A. Classen, M. Cordy, P.-Y. Schobbens, P. Heymans, A. Legay, 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) (2013) 1069–1089, https://doi.org/10.1109/TSE.2012.86.
[38] S. Azzopardi, G. Pace, F. Schapachnik, G. Schneider, Contract automata, Artif. Intell. Law 24 (3) (2016) 203–243, https://doi.org/10.1007/s10506-016-9185-2.
[39] N. Lynch, M. Tuttle, An introduction to input/output automata, CWI Quart. 2 (3) (1989) 219–246.
[40] L. de Alfaro, T. Henzinger, Interface automata, in: Proceedings of the 8th European Software Engineering Conference and 9th International Symposium on Foundations of Software Engineering, ESEC/FSE'01, ACM, 2001, pp. 109–120.
[41] K.G. Larsen, U. Nyman, A. Wąsowski, Modal I/O automata for interface and product line theories, in: R. De Nicola (Ed.), Proceedings of the 16th European Symposium on Programming, ESOP'07, in: LNCS, vol. 4421, Springer, 2007, pp. 64–79.
[42] D. Basile, Applying supervisory control synthesis to priced featured automata and energy problems, Int. J. Softw. Tools Technol. Transf. 21 (6) (2019) 607–612, https://doi.org/10.1007/s10009-019-00533-3.
[43] D. Basile, M.H. ter Beek, R. Pugliese, Bridging the gap between supervisory control and coordination of services: synthesis of orchestrations and choreographies, in: H.R. Nielson, E. Tuosto (Eds.), Proceedings of the 21st International Conference on Coordination Models and Languages, COORDINATION'19, in: LNCS, vol. 11533, Springer, 2019, pp. 129–147.
[44] M. Cordy, A. Classen, G. Perrouin, P.-Y. Schobbens, P. Heymans, A. Legay, Simulation-based abstractions for software product-line model checking, in: Proceedings of the 34th International Conference on Software Engineering, ICSE'12, IEEE, 2012, pp. 672–682.
[45] M. Acher, P. Collet, P. Lahire, J. Montagnat, Imaging services on the grid as a product line: requirements and architecture, in: R. Krut, S. Cohen (Eds.), Proceedings of the 2nd Workshop on Service-Oriented Architectures and Software Product Lines: Putting Both Together, SOAPL'08, University of Limerick, Ireland, 2008, pp. 137–142.
[46] S. Hallsteinsen, S. Jiang, R. Sanders, Dynamic service product lines in service oriented computing, in: Proceedings of the 13th International Software Product Line Conference, SPLC'09, vol. 2, ACM, 2009, pp. 28–34.

[47] P. Istoan, G. Nain, G. Perrouin, J.-M. Jézéquel, Dynamic software product lines for service-based systems, in: Proceedings of the 9th International Conference on Computer and Information Technology, CIT'09, vol. 2, IEEE, 2009, pp. 193–198.
[48] J. Lee, G. Kotonya, Combining service-orientation with product line engineering, IEEE Softw. 27 (2010) 35–41, https://doi.org/10.1109/MS.2010.30.
[49] J. Lee, D. Muthig, M. Naab, A feature-oriented approach for developing reusable product line assets of service-based systems, J. Syst. Softw. 83 (7) (2010) 1123–1136, https://doi.org/10.1016/j.jss.2010.01.048.
[50] M.H. ter Beek, S. Gnesi, M.N. Njima, Product lines for service oriented applications - PL for SOA, in: L. Kovács, R. Pugliese, F. Tiezzi (Eds.), Proceedings of the 7th International Workshop on Automated Specification and Verification of Web Systems, WWV'11, in: EPTCS, vol. 61, 2011, pp. 34–48.
[51] H. Gomaa, K. Hashimoto, Dynamic software adaptation for service-oriented product lines, in: Proceedings of the 15th International Software Product Line Conference, SPLC'11, vol. 2, ACM, 2011, pp. 35:1–35:8.
[52] L. Baresi, S. Guinea, L. Pasquale, Service-oriented dynamic software product lines, Computer 45 (10) (2012) 42–48, https://doi.org/10.1109/MC.2012.289.
[53] J. Lee, G. Kotonya, D. Robinson, Engineering service-based dynamic software product lines, Computer 45 (10) (2012) 49–55, https://doi.org/10.1109/MC.2012.284.