Controller Synthesis of Contract-based Service Product Lines

Extended Version

Davide Basile,
Maurice H. ter Beek,
Felicita Di Giandomenico,
Stefania Gnesi
ISTI-CNR, Pisa, Italy

Pierpaolo Degano,
Gian Luigi Ferrari
Dipartimento di Informatica
Università di Pisa, Italy

Axel Legay
IRISA/INRIA
Rennes, France

ABSTRACT

In Service-Oriented Computing, contracts offer a way to characterise the behavioural conformance of a composition of services, and guarantee that the results do not lead to spurious compositions. Through variability modelling, a product line of services is enabled to adapt to customer requirements and to changes in the context where they operate. We extend a previously introduced formal model of service contracts towards variability and product line modelling, in particular we include: (i) feature-based constraints and (ii) four classes of service requests to characterise different types of service agreement.

We then exploit Supervisory Control Theory to synthesise the most permissive controller of a composition of services that satisfies: (i) all feature constraints of the service product line, and (ii) the maximal number of service requests for which an agreement can be reached. Moreover, the controller of a service product line, whose number of products is potentially exponential in the number of features, can be synthesised from only a subset of its products. A prototypical tool supports the developed theory.

Reference format:
Davide Basile, Maurice H. ter Beek, Felicita Di Giandomenico, Stefania Gnesi, Pierpaolo Degano, Gian Luigi Ferrari, and Axel Legay. Controller Synthesis of Contract-based Service Product Lines: Extended Version. Technical Report 2017-TR-003, ISTI–CNR, 2017.

1 INTRODUCTION

Service-oriented computing (SOC) [21] is a paradigm for distributed computing based on the publication, discovery and orchestration of *services*, which are autonomous, platform-independent and reusable computational units. Services are usually programmed with little or no knowledge about clients and other services before being loosely coupled into networks of collaborating end-user applications.

Web applications reuse services in different configurations over time, e.g. due to the need to adapt to changes in the environment or to the resources of the devices on which they run. Therefore the idea to organise them into *dynamic service product lines* was first explored a decade ago in the SOAPL workshop series at three consecutive SPLC conferences (cf., e.g., [22, 27, 28]), leading to applications for Web stores, smart grids and services as used in scientific workflows and grid computing [1, 2, 11]. Recently, interest has been revived and the Web application JHipster, which has a micro-service architecture, has been lifted to a product line as well as e-Government public licensing services [17, 23].

On a different line, *service contracts* [4] have been introduced to formally describe the behaviour of services in terms of their obligations (i.e. *offers* of the service) and their requirements (i.e. *requests* by the service). Contracts characterise an *agreement* among services as an orchestration (i.e. a composition) of them based on the satisfaction of all requirements through obligations. Orchestrations can dynamically adapt to the discovery of new services, to service updates and to services that are no longer available.

In [6], *contract automata* were introduced as a formal model for service contracts. They represent either single services (called *principals*) or compositions of services based on orchestrated or choreographed coordination [7]. The goal of each principal is to reach an accepting (final) state by matching its requests with corresponding offers of other principals. Through service contracts it is then possible to characterise the behaviour of an ensemble of services. The notion of *agreement* then characterises safe executions of services (i.e. all requests matched by corresponding offers).

In [8], contract automata were equipped with variability mechanisms to distinguish *necessary* ($\Box$) from *permitted* ($\Diamond$) requests, mimicking uncontrollable and controllable actions, respectively, from Supervisory Control Theory [15]. Offers were only permitted as dictated by agreement. Contract agreement guaranteed the fulfilment of all necessary requests and negotiated the maximum number of permitted requests that could be fulfilled without spoiling the service composition. Contracts adapt to the overall agreement by renouncing to unsatisfiable, yet permitted requirements.

In this paper, we introduce *featured modal contract automata* (FMCA) for modelling contract-based dynamic service product lines. These FMCA extend the aforementioned modal service contract automata (MSCA) from [8] with the possibility to define:

(1) *Feature constraints* on service actions (requests and offers);
(2) *Urgent*, *greedy* and *lazy* necessary service requests.

These two additional variability dimensions required us to carefully revisit and extend all fundamental notions used in [8]. Features are identified as service actions, and each FMCA represents a behavioural product line of services equipped with feature constraints. A feature constraint can be any of the constraints used in feature models, including cross-tree constraints, defined as its corresponding propositional formula (cf. [9, 26]). Each product is identified as a truth assignment satisfying the feature constraints.

Urgent, greedy and lazy requests are, in decreasing order of relevance, necessary service requests with further restrictions on their satisfiability. Similarly to [8], permitted requests are optional and can thus be discarded for reaching an agreement.

The main contributions of this paper are as follows:

(1) We introduce a new formalism for contract-based dynamic service product lines;
(2) We define an algorithm for synthesising an orchestration of services in agreement, either for a single product or for the entire service product line. The result is the so-called *maximally permissive controller* (*mpc*) satisfying all feature constraints, all variants of necessary service requests and the maximal number of permitted requests.
(3) Based on a (partial) order of products of a product line, we show how to compute, starting from a subset of products:
    (a) the validity of all products of the service product line;
    (b) the *mpc* of the entire service product line;
    (c) the *mpc* of a product from that of its super-products.

Since the number of valid products of a product line is in general exponential in the number of features, it is important to note that the subset of products used in (3) is potentially smaller than the set of all products. The above mentioned (partial) order relates each product to its sub- and super-products, i.e. those products in which more or less, respectively, variability has been resolved. Products in which not all variability has been resolved are also known as subfamilies. Finally, the theory presented in this paper has been implemented in an open-source prototypical tool (cf. Fig. 2), available at https://github.com/davidebasile/FMCAT/, and its applicability is further demonstrated by a running example.

Outline. We introduce an example hotel reservation service product line in Sect. 2. We formally define feature constraints and (sub-/super-)products over service actions in Sect. 3, followed by FMCA and their compositions and refinements in Sect. 4. The controller synthesis algorithms for FMCA are presented in Sect. 5. In Sect. 6, we discuss related work and conclude the paper. An appendix contains all proofs and additional figures.

2 MOTIVATING EXAMPLE

To illustrate our approach and help intuition, we consider a simple franchise of hotel reservation systems. The system consists of Hotel and Client contracts. Some of them are depicted in Figures 1c–1e.

Feature model. A feature model defines all products of a product line and it has a corresponding propositional formula φ over (primitive) features, called *feature constraint* (cf. Sect. 3). We identify features as actions (requests and offers) performed by services. Moreover, each product is identified by its set of *required* (literals interpreted as *true* in φ) and *forbidden* features (literals interpreted as *false* in φ). All required features must be present in the service, while none of the forbidden features may be present.

For a lighter presentation, we consider only the Hotel product line's feature model depicted in Fig. 1a (Clients do not specify feature models), but our approach scales to larger numbers of features. The feature model allows two *alternative* payment methods: *cash* or *card*. Moreover, any product offering cash payment, *requires* the *invoice* feature to be present. Indeed, the Hotel franchise (i.e. the product line) wants to prevent any of its hotels (i.e. a product) to perform off-book payments. Thus the feature constraint corresponding to the feature model of Fig. 1a is:

$$\varphi = ((card \wedge \neg cash) \vee (cash \wedge \neg card)) \wedge (\neg cash \vee invoice)$$

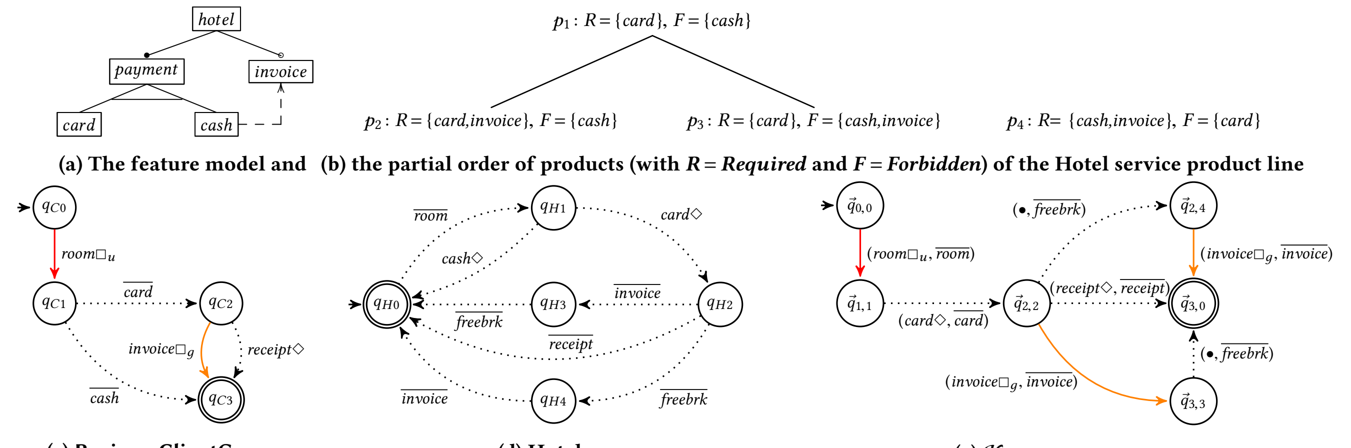
The three *valid products* are depicted in Fig. 1b, together with a valid super-product p_1 in which the presence of the *invoice* feature has not yet been resolved (unresolved features will be activated by the orchestration, if possible). These are the products satisfying φ (e.g. $card = true$ and $cash = false$ satisfies φ, denoted by $\varphi \models_{p_1} true$). Products can be ordered according to their required and forbidden actions. In Fig. 1b, p_2 and p_3 are sub-products of p_1, written $p_2 \preceq p_1$ and $p_3 \preceq p_1$. Indeed, the required and forbidden actions of p_1 are contained in those of p_2 and p_3. This ordering will be exploited for efficiently verifying all products.

Behavioural contracts. A service contract characterises service behaviour in terms of offer and request actions, drawn respectively as overlined and non-overlined labels, while permitted transitions are depicted as dotted (cf. Fig. 1). We extend contracts from [8] by also indicating "when" (i.e. in which states) necessary requests have to be matched. Therefore, we partition the set of necessary service requests into urgent ($\Box_u$), greedy ($\Box_g$) and lazy ($\Box_\ell$) requests. Urgent requests are the most restrictive and must be matched whenever they can be executed. Greedy requests must be matched as soon as possible, i.e. their execution can be delayed until the first match is available. Lazy requests are the less restrictive and only require to be matched somewhere.

In the hotel reservation service scenario, we assume two classes of Clients: business and economy. In Fig. 1c, the contract of a BusinessClient is depicted. It starts by requiring to book a room ($room\Box_u$). The Client request in this case is *urgent*, due to its business priority. Once the room is selected, the client can either perform an off-book cash-only payment ($\overline{cash}$), not requiring any receipt or invoice, or a credit card payment ($\overline{card}$). Assuming a client travelling for business, an invoice or receipt is needed for being reimbursed by the clients' organisation. The organisation must accept invoices, while receipts may be rejected. In case of cash payments, the client is (maliciously) using false invoices to ask larger reimbursement sums (e.g. a hotel product could be owned by an accomplice). In case of (honest) payment by credit card, the client will require an invoice ($invoice\Box_g$) or a receipt ($receipt\Diamond$) from the hotel. The invoice request is marked as (necessary) greedy. The contract of an EconomyClient is equal to that of a BusinessClient, except for the room request being marked as lazy ($room\Box_\ell$), i.e. with a lower priority. We will also consider a "lazier" version of both clients, where the invoice request is marked as lazy ($invoice\Box_\ell$), with ClientG and ClientL indicating the greedy and lazy version, respectively, of the invoice request.

In Fig. 1d, the Hotel contract is depicted. Recall its feature constraint φ. This service starts by offering a room ($\overline{room}$). The hotel accepts payments by either credit card ($card\Diamond$) or cash ($cash\Diamond$). The mutual exclusion (xor) between these features (known to be not expressible solely by the automaton [10]) is specified in φ. It ensures that exactly one type of payment is available in each Hotel product. In case of cash payments, the service returns to its initial (and final) state q_{H0}. Otherwise, the service proceeds by emitting either a receipt ($\overline{receipt}$) or an invoice ($\overline{invoice}$). In the latter case, a special free breakfast offer ($\overline{freebrk}$) is delivered as a gift either before or after the invoice has been emitted. After these possible interactions, the hotel service returns to its initial state q_{H0}.

We also consider two different contracts for the Hotel service product line, called HotelGreedyBad and HotelLazyBad. All these product lines share the same feature model, but have a slightly


hotel
payment
invoice
card
cash
$p_1: R = \{card\}, F = \{cash\}$
$p_2: R = \{card,invoice\}, F = \{cash\}$
$p_3: R = \{card\}, F = \{cash,invoice\}$
$p_4: R = \{cash,invoice\}, F = \{card\}$
q_{C0} q_{C1} q_{C2} q_{C3}
$room\Box_u$
$\overline{card}$
$invoice\Box_g$
$receipt\Diamond$
$\overline{cash}$
q_{H0} q_{H1} q_{H2} q_{H3} q_{H4}
$\overline{room}$
$card\Diamond$
$cash\Diamond$
$\overline{invoice}$
$\overline{freebrk}$
$\overline{receipt}$
$\vec{q}_{0,0}$ $\vec{q}_{1,1}$ $\vec{q}_{2,2}$ $\vec{q}_{2,4}$ $\vec{q}_{3,0}$ $\vec{q}_{3,3}$
$(room\Box_u, \overline{room})$
$(card\Diamond, \overline{card})$
$(\bullet, \overline{freebrk})$
$(receipt\Diamond, \overline{receipt})$
$(invoice\Box_g, \overline{invoice})$

(a) The feature model and (b) the partial order of products (with $R = Required$ and $F = Forbidden$) of the Hotel service product line

(c) BusinessClientG **(d) Hotel** **(e) $\mathcal{K}_{\text{BusinessClientG} \otimes \text{Hotel}_{p_1}}$**

Figure 1: The product line of Hotel reservation service

different behaviour than Hotel. The HotelGreedyBad activates a captcha in case the client selects the offer $\overline{invoice}$ instead of the free breakfast offer, to avoid possible denial-of-service attacks. More precisely, the Hotel transition $(q_{H3}, \overline{freebrk}, q_{H0})$ in Fig. 1d is replaced with the permitted request $(q_{H3}, captcha\Diamond, q_{H0})$. Finally, HotelLazyBad is equal to HotelGreedyBad, except for state q_{H4} and its incident transitions that are removed: this service does not offer any free breakfast and always performs a captcha check.

Contract compositions. The composition BusinessClientG⊗Hotel in agreement (i.e. orchestration) of product p_1 is depicted in Fig. 1e. In this case, the feature constraint (and hence the valid products) of the orchestration is exactly φ (recall that the client has no feature constraint). This orchestration is identical to the one of p_2 because the required invoice feature (required in p_2 and not in p_1) is available in both orchestrations. Conversely, the orchestration BusinessClientG ⊗ Hotel of products p_3 and p_4 is *empty*: no agreement exists. For product p_3, it forbids the necessary (greedy) invoice request, executable in both states $\vec{q}_{2,2}$ and $\vec{q}_{2,4}$.[1] While p_1, p_2 and p_3 are products featuring payments made by credit card, product p_4 corresponds to the Hotel product requiring payments by cash (and hence forbidding payments by credit card). In this case, the franchise Hotel product line is protected from possible off-book payments by also requiring (via φ) the offer $\overline{invoice}$. The orchestration is indeed empty: the malicious behaviour is blocked.

Note here that requirements of products are stricter conditions than necessary requests. Indeed, the requirements of a product are defined on top of its behavioural contract, while necessary requests are defined inside the contract. Unreachable necessary requests do not spoil the contract agreement. Conversely, an unreachable action required by a product violates the contract agreement (e.g. *invoice* in p_4 is unreachable because *card* is forbidden).

Finally, the orchestration of the entire service product line is simply the orchestration of one of the products p_1 or p_2.

We next explain how the different classes of necessary requests affect the orchestration of service contracts. The considered orchestrations are always referring to the entire product line.

[1]In Fig. 1e, the subscripts of $\vec{q}$ identify the client's local state and the hotel's local state, in the order in which they are composed. Likewise for other compositions.

We first consider a composition of the Hotel service with both business and economy clients and show how urgent requests can be used to enforce priorities in service requests. If EconomyClient is served *before* BusinessClient, i.e. (EconomyClient ⊗ Hotel) ⊗ BusinessClient, then $t_1 = (\vec{q}_{0,0,0}, (room\Box_\ell, \overline{room}, \bullet), \vec{q}_{1,1,0})$, i.e. a match between lazy request and offer, is activated in the composition instead of transition $t_2 = (\vec{q}_{0,0,0}, (\bullet, \overline{room}, room\Box_u), \vec{q}_{0,1,1})$, i.e. an urgent match. As a result, the corresponding orchestration will be empty. Intuitively, the business class should be served before the economy one (i.e. t_2 instead of t_1). Indeed, an orchestration in agreement is admitted by EconomyClient ⊗ (Hotel ⊗ BusinessClient), with business client served before the economy one (cf. Fig. 2).

We now consider the orchestration of the service composition BusinessClientG⊗HotelGreedyBad, which is empty. This is caused by the request $(\vec{q}_{3,3}, (\bullet, captcha\Diamond), \vec{q}_{3,0})$ not matched (recall that each request must be matched by a corresponding offer). In this case, the orchestration cannot prevent the execution of the greedy transition $(\vec{q}_{2,2}, (invoice\Box_g, \overline{invoice}), \vec{q}_{3,3})$. Indeed, the greedy invoice request of BusinessClientG requires to be matched as soon as possible: it cannot be "delayed" to the subsequent state $\vec{q}_{2,4}$ (cf. Fig. 1e). The resulting orchestration is thus empty.

This is not the case for the orchestration of the composition BusinessClientL ⊗ HotelGreedyBad. Now the necessary lazy transition $(\vec{q}_{2,2}, (invoice\Box_\ell, \overline{invoice}), \vec{q}_{3,3})$, and consequently the permitted request transition $(\vec{q}_{3,3}, (\bullet, captcha\Diamond), \vec{q}_{3,0})$, are removed in the orchestration. This is possible since the necessary lazy request *invoice* of BusinessClientL is delayed to be matched in state $\vec{q}_{2,4}$.

Finally, consider the orchestration of the service composition BusinessClientL ⊗ HotelLazyBad. In this composition, all transitions incident in states $\vec{q}_{2,4}$ of Fig. 1e are absent (recall that HotelLazyBad does not offer free breakfast). The lazy match transition $(\vec{q}_{2,2}, (invoice\Box_\ell, \overline{invoice}), \vec{q}_{3,3})$ thus cannot be removed in the orchestration, as it is the only available match and the lazy invoice request is *necessary*. Also in this case the orchestration is empty.

In the next section, we present a novel formal model for contract-based product lines capable of expressing all aspects discussed in this section, as well as a novel synthesis algorithm for computing the orchestration of a single product or the entire product line.

3 FEATURE CONSTRAINTS AND PRODUCTS

A feature model is a rooted and/or tree in which nodes are features and additional relations between nodes model further constraints (typically mandatory, optional or alternative, but also requires and xor) [14, 30]. It is well known that a feature model is equivalent to a propositional formula over features. Thus, checking the validity of a product with respect to the feature model reduces to a Boolean satisfiability problem, efficiently computable with BDD or SAT solvers [9, 19, 26]. Following [9, 30], we distinguish compound features (intermediate, decomposable nodes) and primitive features (influencing final products). The latter are represented by the leaves of a feature model and the propositional formula representing a feature model uses only them as literals (cf. Sect. 2).

In our framework, we distinguish basic actions belonging to the sets of *requests* $\mathbb{R} = \{a, b, c, \ldots\}$ and *offers* $\mathbb{O} = \{\overline{a}, \overline{b}, \overline{c}, \ldots\}$ where $\mathbb{R} \cap \mathbb{O} = \emptyset$. Primitive features are identified as basic actions. A *feature constraint* is a propositional logic formuale φ over $\mathbb{R} \cup \mathbb{O}$.

A service product line is then characterised by a conjunction of feature constraints with literals in $\mathbb{R} \cup \mathbb{O}$, such that each assignment p satisfying φ (written $\varphi \models_p$ *true*) is a *valid product*.

Definition 3.1 (Valid products). Let φ be a conjunction of feature constraints with literals in $\mathbb{R} \cup \mathbb{O}$ and let $\mathcal{P} : \mathbb{R} \cup \mathbb{O} \Rightarrow \{true, false\}$ be an interpretation function. Then $[\![\varphi]\!] = \{ p \mid \varphi \models_p true \text{ and } p \in \mathcal{P} \}$ is the set of all valid products of φ. Moreover, given $p \in [\![\varphi]\!]$, the sets of required and forbidden actions in p are $Required(p) = \{ a \mid p(a) = true \}$ and $Forbidden(p) = \{ a \mid p(a) = false \}$, respectively.

All valid products $[\![\varphi]\!]$ of a family can be ordered by component-wise set inclusion as (a subset of) elements of a lattice such that the bottom element $\bot$ requires and forbids all actions, whereas the top element $\top$ has neither required nor forbidden actions. Note that not all elements of such a lattice correspond to valid products.

Definition 3.2 (Sub-products). Let $(R, F) \subseteq (R', F') \in ((\mathbb{R} \cup \mathbb{O}) \times (\mathbb{R} \cup \mathbb{O}), \subseteq)$ be a lattice iff $R \subseteq R'$ and $F \subseteq F'$. The partial order of products of a family $[\![\varphi]\!]$ is $([\![\varphi]\!], \leq)$, where $p \leq p'$ (p is a sub-product of p' or, alternatively, p' is a super-product of p) iff

$$(Required(p'), Forbidden(p')) \subseteq (Required(p), Forbidden(p))$$

Example 3.3. The feature model depicted in Figure 1a is represented by the propositional formula φ in Sect. 2. The partial order of products is depicted in Figure 1b, where $\leq$ grows top-down. We have $[\![\varphi]\!] = \{p_1, p_2, p_3, p_4\}$ and $\leq = \{(p_2, p_1), (p_3, p_1)\}$.

In the sequel, we will exploit the partial order of (valid) products to synthesise the most permissive controller of a service product line.

4 FEATURED MODAL CONTRACT AUTOMATA

We now formally define feature modal contract automata (FMCA), which extend modal service contract automata (MSCA) [8].

We borrow some useful notation from [6, 7]. The alphabet of *basic actions* is defined as $\Sigma = \mathbb{R} \cup \mathbb{O} \cup \{\bullet\}$ where $\bullet \notin \mathbb{R} \cup \mathbb{O}$ is a distinguished element representing the *idle* move. We define the involution $co(\bullet) : \Sigma \mapsto \Sigma$ s.t. $co(\mathbb{R}) = \mathbb{O}$, $co(\mathbb{O}) = \mathbb{R}$ and $co(\bullet) = \bullet$.

Let $\vec{v} = (e_1, \ldots, e_n)$ be a vector of *rank* $n \geq 1$, denoted by r_v, 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 matches, i.e. there exists exactly i, j such that $\vec{a}_{(i)}$ is an offer and $\vec{a}_{(j)}$ is the complementary request or vice versa; 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 $\bullet$. Formally:

Definition 4.1 (Actions). Given a vector $\vec{a} \in \Sigma^n$, if

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

Actions $\vec{a}$ and $\vec{b}$ are *complementary*, denoted by $\vec{a} \bowtie \vec{b}$, if and only if the following holds: (i) $\exists \alpha \in \mathbb{R} \cup \mathbb{O}$ s.t. $\vec{a}$ is either a request or an offer on α; (ii) $\vec{a}$ is an offer on α implies that $\vec{b}$ is a request on $co(\alpha)$; (iii) $\vec{a}$ is a request on α implies that $\vec{b}$ is an offer 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 contract-based service product line through (i) *permitted* and *necessary* transitions (inherited from MSCA); and (ii) a conjunction of feature constraints φ identifying all valid products. We recall that, similarly to MSCA, all offers are permitted. Permitted offers and requests are optional and can be discarded.

The set of necessary requests of an FMCA is further partitioned into *urgent*, *greedy* and *lazy*. These sets contain *necessary* requests that must be matched to reach an agreement among contracts. Compared to MSCA, we thus offer modellers another layer of variability based on the possibility to specify "when" requests must be matched in a composition (cf. Sect. 2). In Sect. 4.2, we will show how these requests give rise to an increasing degree of controllability.

Definition 4.2 (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* $\mathcal{A}$, FMCA for short, of rank $n \geq 1$ is a tuple $\langle Q, \vec{q_0}, A^{\Diamond}, A^{\Box_u}, A^{\Box_g}, A^{\Box_\ell}, A^o, T, \varphi, F\rangle$, where

- $Q = Q_1 \times \cdots \times Q_n \subseteq \mathfrak{Q}^n$
- $\vec{q_0} \in Q$ is the initial state
- $A^{\Diamond}, A^{\Box_u}, A^{\Box_g}, A^{\Box_\ell} \subseteq \mathbb{R}$ are (pairwise disjoint) sets of permitted, urgent, greedy and lazy requests, resp., and we denote by $A^r = A^{\Diamond} \cup A^{\Box_u} \cup A^{\Box_g} \cup A^{\Box_\ell}$ the set of requests
- $A^o \subseteq \mathbb{O}$ 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 $T^{\Box}$ with $T = T^{\Diamond} \cup T^{\Box}$ such that, given $t = (\vec{q}, \vec{a}, \vec{q'}) \in T$, the following holds:
  - $\vec{a}$ 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 on $a \in A^{\Diamond}$, an offer on $\overline{a} \in A^o$ or a match on $a \in A^{\Diamond} \cup A^o$
  - $t \in T^{\Box}$ iff $\vec{a}$ is either a request $a \in A^{\Box_u} \cup A^{\Box_g} \cup A^{\Box_\ell}$ or a match on $a \in A^{\Box_u} \cup A^{\Box_g} \cup A^{\Box_\ell} \cup A^o$
- φ is a conjunction of feature constraints
- $F \subseteq Q$ is the set of final states

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

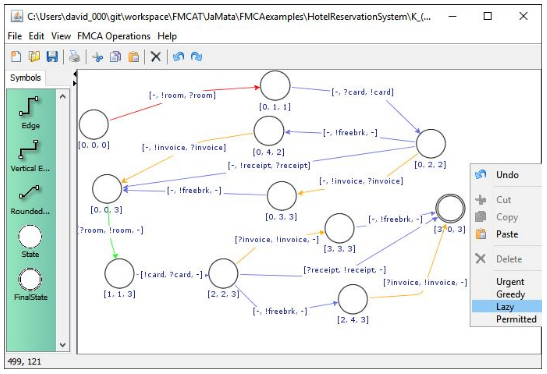


Figure 2: FMCAT with $\mathcal{K}_{\text{EconomyClientG}\otimes(\text{Hotel}\otimes\text{BusinessClientG})}$

For brevity, unless stated differently, we assume a fixed FMCA $\mathcal{A} = \langle Q_{\mathcal{A}}, \vec{q_0}_{\mathcal{A}}, A^{\Diamond}_{\mathcal{A}}, A^{\Box_u}_{\mathcal{A}}, A^{\Box_g}_{\mathcal{A}}, A^{\Box_\ell}_{\mathcal{A}}, A^{o}_{\mathcal{A}}, T_{\mathcal{A}}, \varphi_{\mathcal{A}}, F_{\mathcal{A}} \rangle$ of rank n. Subscript $\mathcal{A}$ may be omitted when no confusion can arise. Moreover, if not stated otherwise, each operation $f(A^r)$ (e.g. union) is intended to be performed homomorphically on $f(A^{\Diamond})$, $f(A^{\Box_u})$, $f(A^{\Box_g})$, $f(A^{\Box_\ell})$. Finally, abusing notation we may write $T^{\Diamond \cup \Box}$ as shorthand for $T^{\Diamond} \cup T^{\Box}$ and likewise for other transition sets, and we may denote a transition t as a request, offer or a match if its label is such. Note that only requests and matches can be marked necessary: a service contract can always withdraw offers, as they are not necessary for reaching an agreement (cf. Sect. 5), i.e. they are optional. The example in Sect. 2 explains the intuition behind this design choice. 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 no client requires free breakfast to match the offer, although they all are willing to pay for a room. An FMCA recognises a trace language over actions and their modalities.

Definition 4.3. Let $\mathcal{A}$ be an FMCA and $\bigcirc \in \{\Diamond, \Box_u, \Box_g, \Box_\ell\}$. 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, 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 (e.g. $(a\Box_\ell, \overline{a}) \equiv (a, \overline{a})\Box_\ell$).

4.1 Composing FMCA

The FMCA operators of composition are crucial for specifying dynamic service product lines, in particular for generating (at binding time) an ensemble of services. By adding new services to an existing composition, it is possible to dynamically update the service product line and to synthesise, if possible, a composition satisfying all requirements defined by the service contracts (cf. Sect. 5).

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

Definition 4.4 (Composable). A set $Set = \{\mathcal{A}_i \mid i \in 1 \ldots n\}$ of FMCA is *composable* iff $(\bigwedge_{\mathcal{A}_i \in Set}\ \varphi_{\mathcal{A}_i}) \not\models \mathit{false}$.

Example 4.5. All the FMCA in Sect. 2 are trivially composable: indeed all clients have feature constraint $\varphi_{Client} = \mathit{true}$ and it holds that $\mathit{true} \wedge \varphi \not\models \mathit{false}$.

We now introduce our first (non-associative) operation of composition. The operands of the composition $\otimes$ are either principals or composite services. Intuitively, the product composition interleaves the actions of all operands, with the only restriction that if two operands are ready to execute two complementary actions ($\vec{a}_i \bowtie \vec{a}_j$) then only their match will be allowed and their interleaving prevented. Below we use $\bigcirc$ as a placeholder for both necessary ($\Box$) and permitted ($\Diamond$) transitions. More in detail, the transitions of the composite service are generated as follows. Case (1) in Definition 4.6 generates match transitions starting from two operands' transitions having complementary actions ($\vec{a}_i \bowtie \vec{a}_j$). If, e.g., $(\vec{q}_j, \vec{a}_j, \vec{q}\,'_j) \in T^{\Box}$, then the resulting match transition will be marked as necessary (i.e. $(\vec{q}, \vec{c}, \vec{q}\,') \in T^{\Box}$). If both operands' complementary actions are permitted, then their resulting match transition t will be marked as permitted. All other principals not involved in t will remain idle.

Case (2) in Definition 4.6 generates all interleaved transitions only if no complementary actions can be executed from the composed source state (i.e. $\vec{q}$). In this case, an operand executes its transition $t = (\vec{q}_i, \vec{a}_i, \vec{q}\,'_i)$ and all other operands remain idle. The composed transition will be marked as necessary (permitted) only if t is necessary (permitted, respectively). Note that condition $\vec{a}_i \bowtie \vec{a}_j$ excludes pre-existing match transitions of the operands from generating 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 4.6 (Composition). Let $\mathcal{A}_i$ be composable FMCA of rank r_i, $i \in 1, \ldots, n$, and let $\bigcirc \in \{\Diamond, \Box\}$. 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 $\vec{q_0} = \vec{q_0}_1 \cdots \vec{q_0}_n$
- $A^r = \bigcup_{i \in 1 \cdots n} A^r_i$, $A^o = \bigcup_{i \in 1 \cdots n} A^o_i$,
- $T^{\bigcirc} \subseteq Q \times A \times Q$ 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 there are $1 \le i < j \le n$ 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 there is $1 \le i \le n$ s.t. $(\vec{q}_i, \vec{a}_i, \vec{q}\,'_i) \in T^{\bigcirc}_i$ and
  $$\begin{cases} \vec{c} = \bullet^u \vec{a}_i \bullet^v, \text{with } u = r_1 + \cdots + r_{i-1}, \\ v = r_{i+1} + \cdots + r_n, |\vec{c}| = m, \\ \vec{q}\,' = \vec{q}_1 \cdots \vec{q}_{i-1}\, \vec{q}\,'_i\, \vec{q}_{i+1} \cdots \vec{q}_n \text{ and } \forall j \neq i, 1 \le j \le n \\ \text{s.t. } (\vec{q}_j, \vec{a}_j, \vec{q}\,'_j) \in T^{\bigcirc \cup \Diamond}_j,\ \vec{a}_i \bowtie \vec{a}_j \text{ does not hold} \end{cases}$$
- $\varphi = \bigwedge_{i \in 1 \ldots n} \varphi_i$
- $F = \{\, \vec{q}_1 \cdots \vec{q}_n \mid \vec{q}_1 \cdots \vec{q}_n \in Q,\ \vec{q}_i \in F_i,\ i \in 1 \ldots n \,\}$

Example 4.7. In Figs. 1c–1d, two principals discussed in Sect. 2 are depicted. A sub-portion of their composition BusinessClientG $\otimes$ Hotel is shown in Fig. 1e. The outgoing transition $\vec{q}_{0,0} \xrightarrow{(room\Box_u,\, \overline{room})}$ is an example of an urgent match between the urgent request $room\Box_u$ of the first principal and the permitted offer $\overline{room}$ of the second. Moreover, in this composition the transitions $\vec{q}_{0,0} \xrightarrow{(room\Box_g,\, \bullet)}$ or $\vec{q}_{0,0} \xrightarrow{(\bullet,\, \overline{room})}$ are not allowed because of $(room\Box_g, \bullet) \bowtie (\bullet, \overline{room})$.

The projection operator $\prod^i(\mathcal{A})$ retrieves the principal with index i involved in $\mathcal{A}$ and identifies its original transitions and feature constraint. The associative composition operator $\boxtimes$ is defined 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. These two operators are minor adaptations of those from [8] (cf. Appendix 7 for their definitions). In particular, $\boxtimes$ 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 service product lines can be obtained, as explained below. For the sequel, we assume every FMCA $\mathcal{A}$ of rank $r_{\mathcal{A}} > 1$ to be composed by FMCA with the composition operators described here.

Example 4.8. Recall from Sect. 2 the compositions (EconomyClient $\otimes$ Hotel) $\boxtimes$ BusinessClient and EconomyClient $\otimes$ (Hotel $\otimes$ BusinessClient). In both, the transition $\vec{q}_{0,0,0} \xrightarrow{(\bullet,\, \overline{room},\, room\Box_u)}$ is allowed, while it is not in the composition (EconomyClient $\otimes$ Hotel) $\otimes$ BusinessClient, which has an empty orchestration (cf. Sect. 2).

4.2 Refining FMCA

In this section, we define a refinement relation among FMCA, based on the notion of controllability from Supervisory Control Theory [15]. The refinement will be used in Sect. 5 to relate the valid products of an FMCA (cf. Sect. 4.3) and to synthesise the service product family. We start by defining dangling states, i.e. those unreachable or from which no final state can be reached.

Definition 4.9 (Dangling state). Let $\mathcal{A}$ be an FMCA. Then $\vec{q} \in Q$ is *dangling* ($\vec{q} \in Dangling(\mathcal{A})$) iff $\nexists w$ s.t. $\vec{q}_0 \xrightarrow{w}{}^{*} \vec{q}$ or $\vec{q} \xrightarrow{w}{}^{*} \vec{q}_f \in F$.

We now characterise when a transition of an FMCA is *controllable* or *uncontrollable*. The correspondence between permitted/ necessary and controllable/uncontrollable was mainly exploited in [8], where all necessary requests were *greedy*. Here we add an extra layer of information about "when" a necessary request can be matched, which is mainly due to how we build the composition of FMCA (interleavings in Definition 4.6). All permitted actions (offers and requests) are fully controllable. As briefly discussed previously, *urgent*, *greedy* and *lazy* requests have an increasing degree of controllability. An urgent request is fully uncontrollable: it must be matched in every possible state in which it can be executed. A greedy request can be disabled by the controller as long as the first match is available. Finally, a lazy request only requires to be matched: its matches are controllable by the orchestrator, provided at least one match is available. Below, we characterise the controllability of greedy and lazy requests (Definition 4.10) and the controllability of lazy matches (Definition 4.12). We remark that, for permitted and urgent requests, we do not need to characterise their controllability, because in the first case they are always controllable (permitted) and in the second case always uncontrollable (urgent).

A (greedy or lazy) request transition t is a *controllable greedy/lazy request* in FMCA $\mathcal{A}$ if there exists a (greedy or lazy) match transition t' in $\mathcal{A}$ and in both t and t' the same principal, in the same local state, does the same request, and additionally the target state of t' is not dangling. Formally:

Definition 4.10 (Controllable Greedy/Lazy Request). Let $\mathcal{A}$ be an FMCA and let $t = (\vec{q}_1, \vec{a}_1, \vec{q}_1')$ be a request on $a \in A^{\Box_g}$ (resp. $a \in A^{\Box_\ell}$). Then t is a *controllable greedy/lazy request (cglr)* transition in $\mathcal{A}$ iff $\exists(\vec{q}_2, \vec{a}_2, \vec{q}_2') \in T^{\Box}$ s.t. $\vec{a}_2$ is a match, $\vec{q}_2' \notin Dangling(\mathcal{A})$, $\vec{q}_{1(i)} = \vec{q}_{2(i)}$ and $\vec{a}_{1(i)} = \vec{a}_{2(i)} \in A^{\Box_g}$ (resp. $a \in A^{\Box_\ell}$); otherwise t is an *uncontrollable greedy/lazy request (uglr)* transition in $\mathcal{A}$.

Example 4.11. Consider $\mathcal{A}$=EconomyClient $\otimes$ (Hotel $\otimes$ BusinessClient) from Sect. 2, and $\vec{q}_{0,0,0} \xrightarrow{(\bullet,\, \overline{room},\, room\Box_u)} \vec{q}_{0,1,1} \xrightarrow{(\bullet,\, card\Diamond,\, \overline{card})} \vec{q}_{0,2,2} \xrightarrow{(\bullet,\, \overline{receipt},\, receipt\Diamond)} \vec{q}_{0,0,3} \xrightarrow{(room\Box_\ell,\, \overline{room},\, \bullet)}$. Because of this trace, the lazy request transition $\vec{q}_{0,0,0} \xrightarrow{(room\Box_\ell,\, \bullet,\, \bullet)}$ is a cglr transition in $\mathcal{A}$. This cglr request transition can safely be removed in the orchestration: the corresponding request appears in another transition as a match (in this example $\vec{q}_{0,0,3} \xrightarrow{(room\Box_\ell,\, \overline{room},\, \bullet)}$).

Consider a match transition t and a request transition t_r. Intuitively, t_r is said to be extracted from t iff the only difference between t and t_r is that the principal executing the offer in t is idle in t_r, i.e. $t = (\vec{q}, \vec{a}, \vec{q}')$ with $\vec{a}_{(j)} = a \in A^o$, and $t_r = (\vec{q}_r, \vec{a}_r, \vec{q}_r')$ s.t. $\vec{q}'_{r(j)} = \vec{q}_{(j)}$ and $\vec{a}_{r(j)} = \bullet$ and equals t anywhere else. A lazy match transition t is an *uncontrollable lazy match* in $\mathcal{A}$ if the lazy request transition t_r extracted from t is an uglr transition in $\mathcal{A}$. Formally:

Definition 4.12 (Uncontrollable lazy match). Let $\mathcal{A}$ be an FMCA, $t = (\vec{q}, \vec{a}, \vec{q}') \notin T_{\mathcal{A}}$ with $\vec{q} \in Q_{\mathcal{A}} \setminus Dangling(\mathcal{A})$ be a lazy match transition and t_r be the request transition extracted from t. Then t is an *uncontrollable lazy match (ulm)* transition in $\mathcal{A}$ iff t_r is an uglr transition in $\mathcal{A}$; otherwise t is a *controllable lazy match (clm)* in $\mathcal{A}$.

Example 4.13. Consider the composition $\mathcal{A}$ =BusinessClientL $\otimes$ HotelGreedyBad of Sect. 2 and its orchestration $\mathcal{K}_{\mathcal{A}}$. The necessary lazy transition $t = (\vec{q}_{2,2}, (invoice\Box_\ell, \overline{invoice}), \vec{q}_{3,3})$ is removed in $\mathcal{K}_{\mathcal{A}}$. Moreover, $\vec{q}_{2,2}$ is a state of $\mathcal{K}_{\mathcal{A}}$ that is not dangling. The request $t_r = (\vec{q}_{2,2}, (invoice\Box_\ell, \bullet), \vec{q}_{3,2})$ extracted from t is a cglr in $\mathcal{K}_{\mathcal{A}}$, because the request $invoice\Box_\ell$ is matched in the transition $t' = (\vec{q}_{2,4}, (invoice\Box_\ell, \overline{invoice}), \vec{q}_{3,0})$. Conversely, consider the ill-formed orchestration $\mathcal{K}_{\text{ill}}$, obtained from the FMCA in Fig. 1e by only considering states $\vec{q}_{0,0}$, $\vec{q}_{1,1}$, $\vec{q}_{2,2}$, $\vec{q}_{3,0}$ and their incident transitions. Now t' is not a transition of $\mathcal{K}_{\text{ill}}$; and no other matches for $invoice\Box_\ell$ are reachable. Hence, in this case, t is an ulm in $\mathcal{K}_{\text{ill}}$.

The notion of *uncontrollable* transition is defined next. Intuitively, an uncontrollable transition cannot be blocked by the orchestrator without affecting the agreement among contracts.

Definition 4.14 (Uncontrollable transition). Let $\mathcal{A}$ be an FMCA and $t = (\vec{q}, \vec{a}, \vec{q}')$ a transition on a. Then t is *uncontrollable (unc.)* in $\mathcal{A}$ iff either $a \in A^{\Box_u}$ or $\vec{a}$ is a match on $a \in A^{\Box_g}$ or t is uglr in $\mathcal{A}$ or t is an ulm in $\mathcal{A}$; otherwise t is *controllable (con.)*.

We are now ready to introduce the refinement relation between FMCA. Intuitively, an FMCA $\mathcal{A}_r$ refines an FMCA $\mathcal{A}$ when all the uncontrollable transitions of $\mathcal{A}$ are maintained in $\mathcal{A}_r$, as well as controllable transitions of $\mathcal{A}$ that are uncontrollable in $\mathcal{A}_r$. Only a subset of the controllable transitions of $\mathcal{A}$ are available in $\mathcal{A}_r$.

Definition 4.15 (Refinement). An FMCA $\mathcal{A}_r$ is a *refinement* of an FMCA $\mathcal{A}$, denoted by $\mathcal{A}_r \preceq \mathcal{A}$, iff $\exists$ a refinement relation $\mathcal{R} \subseteq Q \times Q_r$ s.t. $A_r^r \subseteq A^r$, $A_r^o \subseteq A^o$, $(\vec{q}_0, \vec{q}_{0r}) \in \mathcal{R}$ and $\forall\, (\vec{q}, \vec{q}_r) \in \mathcal{R}$, abbreviating $t = (\vec{q}, \vec{a}, \vec{q}')$ and $t_r = (\vec{q}_r, \vec{a}, \vec{q}_r')$, the following holds:

- $t \in T_{\mathcal{A}}$ unc. in $\mathcal{A}$ iff $\exists \vec{q}_r' \in Q_r$ s.t. $t_r \in T_{\mathcal{A}_r} \wedge (\vec{q}', \vec{q}_r') \in \mathcal{R}$
- $t_r \in T_{\mathcal{A}_r}$ con. in $\mathcal{A}_r$ implies $\exists \vec{q}' \in Q$ s.t. $t \in T_{\mathcal{A}} \wedge (\vec{q}', \vec{q}_r') \in \mathcal{R}$
- $t \in T_{\mathcal{A}}$ con. in $\mathcal{A}$, unc. in $\mathcal{A}_r$ iff $\exists \vec{q}_r' \in Q_r$ s.t. $t_r \in T_{\mathcal{A}_r} \wedge (\vec{q}', \vec{q}_r') \in \mathcal{R}$
- $\vec{q} \notin Dangling(\mathcal{A}) \wedge \vec{q}_r \notin Dangling(\mathcal{A}_r)$

Example 4.16. Consider the compositions $\mathcal{A} = \mathsf{BusinessClientL} \otimes \mathsf{Hotel}$ and $\mathcal{A}_1 = \mathsf{BusinessClientL} \otimes \mathsf{HotelBadGreedy}$ of Sect. 2. Both orchestrations $\mathcal{K}_{\mathcal{A}}$ and $\mathcal{K}_{\mathcal{A}_1}$ are non-empty. Indeed, $\mathcal{K}_{\mathcal{A}}$ is as shown in Fig. 1e, except for all greedy actions that are turned to lazy; while $\mathcal{K}_{\mathcal{A}_1}$ is equal to $\mathcal{K}_{\mathcal{A}}$, except for the state $\vec{q}_{3,3}$ (and the incident transitions) that have been removed. Then $\mathcal{K}_{\mathcal{A}_1} \preceq \mathcal{K}_{\mathcal{A}}$ holds because $t = (\vec{q}_{2,2}, (invoice\Box_\ell, \overline{invoice}), \vec{q}_{3,3})$ is controllable in $\mathcal{K}_{\mathcal{A}}$. Finally, consider $\mathcal{A}_2 = \mathsf{BusinessClientL} \otimes \mathsf{HotelBadLazy}$ with empty orchestration and $\mathcal{K}_{\text{ill}}$ from Example 4.13. We have $\mathcal{K}_{\text{ill}} \not\preceq \mathcal{A}_2$, because $t \in T_{\mathcal{A}_2}, t \notin \mathcal{K}_{\text{ill}}$ and t is controllable in $\mathcal{A}_2$ and uncontrollable in $\mathcal{K}_{\text{ill}}$.

4.3 Valid Products of FMCA

Intuitively, a *valid product* p of an FMCA $\mathcal{A}$ is such that all its required actions are available while its forbidden actions are not. More precisely, p is a (valid) interpretation of the feature constraints of $\mathcal{A}$ (i.e. $p \in [\![\varphi_{\mathcal{A}}]\!]$) such that for all *true* literals a (i.e. $a \in Required(p)$) a reachable transition t on a is executable in $\mathcal{A}$, whereas for all *false* literals b (i.e. $b \in Forbidden(p)$) no reachable transition t on b can be executed in $\mathcal{A}$. Formally:

Definition 4.17 (Valid product). Let $\mathcal{A}$ be an FMCA, then $p \in [\![\varphi_{\mathcal{A}}]\!]$ is valid in $\mathcal{A}$ iff (i) $\forall a \in Required(p)\ \exists(\vec{q}, \vec{a}, \vec{q}\,') \in T_{\mathcal{A}}$ s.t. $\vec{a}$ is an action on a and $\vec{q}\,' \notin Dangling(\mathcal{A})$, and (ii) $\forall b \in Forbidden(p)$ $\nexists(\vec{q}, \vec{b}, \vec{q}\,') \in T_{\mathcal{A}}$ s.t. $\vec{b}$ is an action on b and $\vec{q}\,' \notin Dangling(\mathcal{A})$.

Given a service product line $\mathcal{A}$, one of the benefits of adopting a partial order of products is the possibility to determine all valid products in $\mathcal{A}$ by only exploring a subset of them, as proved in the following theorem. In particular, if a sub-product p is valid then all its super-products p' are also valid products or, equivalently, if a product p' is not valid then neither is any of its sub-products p.

Theorem 4.18 (Quick products validation). *Let $\mathcal{A}$ be a FMCA and $p, p' \in [\![\varphi_{\mathcal{A}}]\!]$ be two products s.t. $p \preceq p'$. Then the following holds:*

$$p \text{ is valid in } \mathcal{A} \text{ implies } p' \text{ is valid in } \mathcal{A} \qquad (1)$$

Example 4.19. Products p_1 and p_2 of Sect. 2 are valid in all orchestrations, while p_3 and p_4 are not. Since $p_2 \preceq p_1$, validity of p_2 implies validity of p_1. Moreover, every potential sub-product of p_3 or p_4 (obtainable by extending the feature model in Fig. 1a) will be not valid in all the given orchestrations.

All valid products in $\mathcal{A}$ can be determined by visiting the lattice of Definition 3.1 in a top-down breadth-first search and by pruning those sub-trees rooted in a product not valid in $\mathcal{A}$. In the next section, the notion of composition of services in agreement and a technique for synthesising it are detailed.

5 CONTROLLER SYNTHESIS FOR FMCA

We first define the property of *(modal) agreement* on FMCA languages, and a technique for synthesising an orchestration of services in agreement. Intuitively, a trace is in agreement if it is a concatenation of matches, offer actions and their modalities. We recall two definitions from [8].

Definition 5.1 (Modal agreement). 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 is safe when all traces of its language are in agreement, and it admits agreement when at least one of its traces is. Formally:

Definition 5.2 (Modal safety). An FMCA $\mathcal{A}$ is *safe* if $\mathscr{L}(\mathcal{A}) \subseteq \mathfrak{A}$; otherwise it is *unsafe*. If $\mathscr{L}(\mathcal{A}) \cap \mathfrak{A} \neq \emptyset$ then $\mathcal{A}$ *admits agreement.*

Example 5.3. Consider the FMCA in Fig. 1e. Its language contains $w = \{(room, \overline{room})\Box_u(\overline{card}, card)\Diamond(receipt, \overline{receipt})\Diamond\}$. This FMCA admits agreement because $w \in \mathfrak{A}$.

We now define an algorithm for synthesising an orchestration of FMCA, viz. the maximal sub-portion of an FMCA $\mathcal{A}$ that is safe. The orchestration will be the *most permissive controller* (*mpc* for short) in the style of Supervisory Control for Discrete Event Systems [15, 29]. A discrete event system is a finite state automaton, where *marked* (i.e. final) states represent the successful termination of a task, while *forbidden* states should never be traversed in "good" computations. The purpose of Supervisory Control Theory is to synthesise a controller that enforces good computations. To do so, it distinguishes between *controllable* events (those that the controller can disable) and *uncontrollable* events (those that are always enabled), besides partitioning events into *observable* and *unobservable* (obviously uncontrollable). If all events are observable, then an *mpc* exists which never blocks a good computation [15].

The purpose of contracts is to declare all executions of a principal in terms of requests and offers. Therefore, we assume that all actions of a (composed) contract are observable. The composition of contracts computed through Definition 4.6 corresponds to the uncontrolled system (i.e. plant) in [15, 29]. Note that the composition of the *mpc* and the plant (i.e. controlled system) is not generated through the operators in Definition 4.6 ($\otimes$, $\boxtimes$). As usual, the controlled system can be obtained by a standard synchronous composition of the *mpc* with the plant, which blocks all transitions that are in the plant but not in the *mpc*. Here we do not specify the controlled system. Indeed, the interactions between the orchestrator and the principals, that are used for realising the orchestration computed through the *mpc*, are implicit in our framework [7].

Clearly, the behaviour that we want to enforce upon a given FMCA are exactly the traces in agreement; thus we assume both (i) request transitions and (ii) forbidden transitions to lead to a forbidden state. To fulfil the modalities imposed by FMCA, we force a composition to be in agreement only if there exists a match for each necessary (urgent, greedy or lazy) request. Moreover, we want to synthesise an orchestration of services that satisfies the feature constraints defined in Sect. 3. To this aim, the synthesis algorithm computes the *mpc* of a *product* of the family identified by the featured constraints. Even though the number of products of a family is in general exponential in the number of features [14], we will show how it is possible to synthesise the *mpc* for the entire product family from only a subset of valid products (cf. Theorem 5.18).

Before defining and computing the *mpc*, we define a state to be in uncontrollable disagreement if the controller cannot avoid a "bad" transition (i.e. request or forbidden action) from being executed.

Definition 5.4 (Uncontrollable disagreement). Let $\mathcal{A}, \mathcal{K}$ be two FMCA 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 unc. in $\mathcal{K}$; or (ii) no others $t' \in T_{\mathcal{K}}$ have source $\vec{q} \notin F_{\mathcal{K}}$. A state $\vec{q} \notin Dangling(\mathcal{A})$ is in *uncontrollable disagreement (unc.dis.)* in p of $\mathcal{A}$ by $\mathcal{K}$ iff $\vec{q} \xrightarrow{w}{}^{*} \vec{q}_1$ by only executing forced transitions and either (1) $w \notin \mathfrak{A}$ or containing a basic action $a \in \mathit{Forbidden}(p)$ or (2) $\nexists w' \in \mathfrak{A}$ not containing basic actions $a \in \mathit{Forbidden}(p)$ s.t. $\vec{q}_1 \xrightarrow{w'}{}^{*} \vec{q}_f \in F_{\mathcal{A}}$.

Example 5.5. State $\vec{q}_{2,2}$ of $\mathcal{A}$ = BusinessClientL ⊗ HotelBadLazy in Sect. 2 is in unc. dis. in (p_1/p_2) of $\mathcal{A}$ by $\mathcal{K}_{ill}$ because the transition $(\vec{q}_{2,2}, (\mathit{invoice}\square_\ell, \overline{\mathit{invoice}}), \vec{q}_{3,3})$ is ulm in $\mathcal{K}_{ill}$ (cf. Example 4.13), and from state $\vec{q}_{3,3}$ the final state $\vec{q}_{3,0}$ can only be reached by executing the request action $(\bullet, \mathit{captcha}\Diamond)$.

A controller $\mathcal{K}$ of (valid) product p of $\mathcal{A}$ allows (1) all traces in agreement where (2) no states in unc.dis. in p of $\mathcal{A}$ by $\mathcal{K}$ are traversed, and blocks those traces not satisfying (1) or (2). Moreover, all actions required by p must be executable by $\mathcal{K}$. Hence a controller $\mathcal{K}$ of an FMCA $\mathcal{A}$ is again an FMCA. The *mpc* of product p of $\mathcal{A}$ is the largest FMCA that is a controller of p, and it is unique up to language equivalence.

Definition 5.6 (Mpc of product). Let $\mathcal{A}, \mathcal{K}$ be FMCA and $p \in [\![\varphi_{\mathcal{A}}]\!]$. Then $\mathcal{K}$ is a *(modal) controller* of product p of $\mathcal{A}$ iff (1) $\mathcal{K}$ is safe, (2) $\mathit{Dangling}(\mathcal{K}) = \emptyset$, (3) $\mathscr{L}(\mathcal{K}) = \emptyset$ or $\forall a \in \mathit{Required}(p)\ \exists w \in \mathscr{L}(\mathcal{K})$ s.t. w contains basic action a, and (4) $\nexists w \in \mathscr{L}(\mathcal{K})$ s.t. w contains actions $a \in \mathit{Forbidden}(p)$ or $\vec{q}_{0\mathcal{K}} \xrightarrow{w}{}^{*} \vec{q}_{\mathcal{K}}$, $\vec{q}_0 \xrightarrow{w}{}^{*} \vec{q}$ and $\vec{q}$ is in unc. dis. in p of $\mathcal{A}$ by $\mathcal{K}$. A controller $\mathcal{K}$ of product p of $\mathcal{A}$ is the *most permissive (modal) controller (mpc)* iff $\forall$ controllers $\mathcal{K}'$ of p, $\mathscr{L}(\mathcal{K}') \subseteq \mathscr{L}(\mathcal{K})$ holds.

Example 5.7. All orchestrations discussed in Sect. 2 are the *mpc* of their corresponding service composition for products p_1 and p_2.

The following lemma relates (i) the existence of an *mpc* of product p of $\mathcal{A}$ to (ii) the notion of validity of p in $\mathcal{A}$ (cf. Definition 4.17). In general, (i) is stronger than (ii), but the two notions are equivalent if and only if the set of actions required by p is non-empty. Finally, Lemma 5.8(4) complements Theorem 4.18 by exploiting the information related to the existence of an *mpc*. While we know from Theorem 4.18 that all sub-products of a product p are not valid in $\mathcal{A}$ if p is not valid in $\mathcal{A}$, the following lemma ensures the existence of a sub-product of p valid in $\mathcal{A}$ (with non-empty *mpc*) provided that p admits a non-empty *mpc*.

Lemma 5.8. *Let $\mathcal{K}_{\mathcal{A}_p}$ be the mpc of product p of $\mathcal{A}$. Then we have:*

$$\mathscr{L}(\mathcal{K}_{\mathcal{A}_p}) \neq \emptyset \text{ implies } p \text{ valid in } \mathcal{K}_{\mathcal{A}_p} \tag{2}$$

$$p \text{ valid in } \mathcal{K}_{\mathcal{A}_p} \text{ and } \mathit{Required}(p) \neq \emptyset \text{ implies } \mathscr{L}(\mathcal{K}_{\mathcal{A}_p}) \neq \emptyset \tag{3}$$

$$\mathscr{L}(\mathcal{K}_{\mathcal{A}_p}) \neq \emptyset \wedge \exists p_1 : p_1 \leq p \text{ implies } \exists p_2 : p_2 \leq p \wedge \mathscr{L}(\mathcal{K}_{\mathcal{A}_{p_2}}) \neq \emptyset \tag{4}$$

Example 5.9. In Sect. 2, for all compositions discussed, their *mpc* of products $p1$ and $p2$ are non-empty (cf. Lemma 5.8(2)). Moreover, both products have a non-empty required set of actions (cf. Lemma 5.8(3)) and are indeed valid products of their corresponding *mpc* (cf. Example 4.19). Finally, $p2 \leq p1$ holds (cf. Lemma 5.8(4)).

Note that in general the converse of Lemma 5.8(2) does not hold, because product validity ignores agreement, which is required by the *mpc* (e.g. it suffices to consider $\mathcal{A}$ not admitting agreement and with products not requiring anything).

We now outline the iterative algorithm for computing the *mpc* of product p of an FMCA $\mathcal{A}$, after which we formally present the algorithm in Definition 5.10. With respect to the standard synthesis in [29], we exploit non-local information related to other transitions for deciding whether a given transition is controllable or uncontrollable (cf. Definition 4.10 and Definition 4.12). At each step i, the algorithm updates incrementally a set of states R_i and revises an FMCA $\mathcal{K}_i$; it terminates when no more updates are possible. Intuitively, the property of agreement requires that all requests are matched. Hence, we want to remove all possible (non-matched) requests. We also want to remove all actions that are forbidden by the product. The *mpc* must prevent these "bad" transitions (requests and actions forbidden by the product) from being executed. This is straightforward for bad controllable transitions, while we can only try to make the bad uncontrollable transitions unreachable. To this aim, the sets R_i contain the "bad" states: those that cannot prevent a necessary request or a forbidden action to be eventually executed (i.e. states in uncontrollable disagreement). Note that by pruning transitions, a cglr transition may become uncontrollable (i.e. the match transition required by Definition 4.10 is removed).

At the starting point, the bad controllable transitions are removed in $\mathcal{K}_0$. For the bad uncontrollable transitions, their source states are added to the set of bad states R_0. Moreover, the dangling states of $\mathcal{K}_0$ are added to R_0. At each iteration i, the algorithm prunes in a backwards fashion from $\mathcal{K}_i$ the controllable transitions with bad target and the uncontrollable transitions with bad source. 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) sources of transitions previously pruned that could have become uncontrollable (and bad) by the successive pruning operations. The transitions in (iii) are either cglr transitions (cf. Definition 4.10) or clm transitions (cf. Definition 4.12) that have become uglr and ulm, respectively (all matches required by the corresponding definitions have been completely pruned from $\mathcal{K}_i$). The algorithm terminates when no new updates are available. Upon termination, if the initial state is bad (in R_n) or some action required by product p is unavailable in $\mathcal{K}_n$, then the *mpc* is empty. Otherwise, the synthesised automaton $\mathcal{K}_n$ is the *mpc* of p.

Since the set R_i is finite and can only increase in each step, the termination of the algorithm is guaranteed.

Definition 5.10 (Synthesis). Let $\mathcal{A}$ be an FMCA, $p \in [\![\varphi_{\mathcal{A}}]\!]$ and $f : \mathit{FMCA} \times 2^Q \to \mathit{FMCA} \times 2^Q$ be a monotone function on the cpo $P = (2^Q, \subseteq)$. Moreover, let $\mathcal{K}_0 = \langle Q, \vec{q}_0, A^{\Diamond}, A^{\square_u}, A^{\square_g}, A^{\square_\ell}, A^o, T \setminus \{\, t \text{ con. in } \mathcal{A} \mid t \text{ request } \vee a \in \mathit{Forbidden}(p) \,\}, \varphi_{\mathcal{A}}, F \rangle$, let $R_0 = \mathit{Dangling}(\mathcal{K}_0) \cup \{\, \vec{q} \mid (\vec{q} \rightarrow) = t \in T^{\square}_{\mathcal{A}} \text{ on } a \text{ unc. in } \mathcal{K}_0,\ (t \text{ request } \vee a \in \mathit{Forbidden}(p)) \,\}$. Let $f(\mathcal{K}_{i-1}, R_{i-1}) = (\mathcal{K}_i, R_i)$ s.t.:

- $\mathcal{K}_i = \langle\, Q, \vec{q}_0, A^{\Diamond}, A^{\square_u}, A^{\square_g}, A^{\square_\ell}, A^o, T_{\mathcal{K}_{i-1}} \setminus$ $(\{\, (\vec{q} \rightarrow \vec{q}\,') = t \in T_{\mathcal{K}_{i-1}} \mid t \text{ con. in } \mathcal{K}_{i-1} \wedge \vec{q}\,' \in R_{i-1} \,\} \cup$ $\{\, (\vec{q} \rightarrow) = t \in T_{\mathcal{K}_{i-1}} \mid t \text{ unc. in } \mathcal{K}_{i-1} \wedge \vec{q} \in R_{i-1} \,\}), \varphi_{\mathcal{A}}, F \,\rangle$
- $R_i = R_{i-1} \cup \{\, \vec{q} \mid (\vec{q} \rightarrow \vec{q}\,') \in T^{\square}_{\mathcal{K}_i} \text{ unc. in } \mathcal{K}_i,\ \vec{q} \notin R_{i-1} \wedge \vec{q}\,' \in R_{i-1} \,\} \cup \{\, \vec{q} \mid (\vec{q} \rightarrow) \in T^{\square}_{\mathcal{A}} \text{ uglr or ulm in } \mathcal{K}_i \,\} \cup \mathit{Dangling}(\mathcal{K}_i)$

and let $(\mathcal{K}_n, R_n) = \mathit{sup}(\{\, f^n(\mathcal{K}_0, R_0) \mid n \in \mathbb{N} \,\})$ be the least fixed point of f. Then the *mpc* $\mathcal{K}_{\mathcal{A}_p}$ of product p of $\mathcal{A}$ is computed as:

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

The following theorem proves that the automaton computed through Definition 5.10 is indeed the *mpc* of product p of $\mathcal{A}$.

Theorem 5.11 (mpc of a 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 5.10 is the mpc of product p of $\mathcal{A}$.*

The next theorem is based on the notion of modal refinement in Definition 4.15 and it states that the *mpc* of product p of $\mathcal{A}$ produces the largest refinement of the principals in $\mathcal{A}$ such that an agreement among the parties is possible. Intuitively, if a permitted action does not spoil the overall agreement then it will be available in the composition of services.

Theorem 5.12 (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 5.10 and let $\Pi_i(\mathcal{K}_{\mathcal{A}_p}) = \mathcal{A}_{r_i}$ be its projections on ith principals. Then the following holds:*

$$\forall i \in I : \mathcal{A}_{r_i} \preceq \mathcal{A}_i \tag{5}$$

$$\forall \mathcal{K}' \neq \langle\rangle \textit{ controller of } p \textit{ of } \mathcal{A} \textit{ and } \forall i : \Pi_i(\mathcal{K}') = \mathcal{A}_{r'_i} \preceq \mathcal{A}_{r_i} \tag{6}$$

Example 5.13. Let $\mathcal{K}$ be the FMCA of Fig. 1e. Then $(\prod^1(\mathcal{K}) = \text{Client}_p) \preceq$ BusinessClientG. In particular, Client_p is as in Fig. 1c but without offer $\overline{cash}$, as it is forbidden by p_1. None of the other permitted transitions is removed by the *mpc*, e.g. $\vec{q}_{2,2} \xrightarrow{(receipt\Diamond,\ \overline{receipt})}$ and $\vec{q}_{2,2} \xrightarrow{(\bullet,\ freebrk)}$, because they do not spoil the overall agreement.

The following lemma relates the partial order of products with the one induced by refinement: the *mpc* $\mathcal{K}_{\mathcal{A}_p}$ of product p of $\mathcal{A}$ is a refinement of the *mpc* $\mathcal{K}_{\mathcal{A}_{p'}}$ of a super-product p' of $\mathcal{A}$.

Lemma 5.14 (Controller refinement). *Let $\mathcal{A}$ be an FMCA and let $p, p' \in [\![\varphi_{\mathcal{A}}]\!]$. Then the following holds:*

$$p \preceq p' \textit{ and } \mathscr{L}(\mathcal{K}_{\mathcal{A}_p}) \neq \emptyset \quad \textit{implies} \quad \mathcal{K}_{\mathcal{A}_p} \preceq \mathcal{K}_{\mathcal{A}_{p'}} \tag{7}$$

Example 5.15. In Sect. 2, for all compositions, the *mpc* for products p_1 and p_2 were indeed equal, and the refinement relation holds trivially. Products p_3 and p_4 have an empty *mpc.*

An important consequence is that all the *mpc* of valid products in $\mathcal{A}$ are ordered according to the refinement relation. This order can be exploited for computing the *mpc* of all the products of a service product line without generating the *mpc* for each product.

The next result states that the *mpc* of an FMCA $\mathcal{A}$ can be defined by the union of a subset of controllers of *canonical* valid products of $\mathcal{A}$: those at "higher" level in the lattice of Definition 3.1, quotiented by their forbidden actions. The selected products are those sufficient and necessary for characterising the *mpc* of each valid product in $\mathcal{A}$ as refinement of the *mpc* of the product family. FMCA union and intersection are standard automata operations.

Definition 5.16 (Canonical products). Let $\mathcal{A}$ be an FMCA. Then

$$TVP(\mathcal{A}) = \{\, p \mid \mathscr{L}(\mathcal{K}_{\mathcal{A}_p}) \neq \emptyset,\ \nexists p' \text{ s.t. } \mathscr{L}(\mathcal{K}_{\mathcal{A}'_p}) \neq \emptyset \text{ and } p \preceq p' \,\}$$

is the set of *top valid products* of $\mathcal{A}$. Moreover, let

$$p \equiv p' \text{ iff } p, p' \in [\![\varphi_{\mathcal{A}}]\!] \text{ s.t. } Forbidden(p) = Forbidden(p')$$

be an equivalence relation on products of $\mathcal{A}$ s.t. $TVP(\mathcal{A})/\equiv$ is the quotient set of $TVP(\mathcal{A})$ by $\equiv$. The set of *canonical products* of $\mathcal{A}$ is

$$CP(\mathcal{A}) = \{\, p_c \mid p_c \text{ is the canonical element of } [p] \in TVP(\mathcal{A})/\equiv \,\}$$

Example 5.17. For all service compositions $\mathcal{A}$ described in Sect. 2, $TVP(\mathcal{A}) = \{p_1\}$. Moreover, $p_1 \equiv p_2$ belong to the same equivalence class, while p_3 and p_4 do not. Finally, p_1 is the canonical element of its (singleton) equivalence class.

Theorem 5.18 (mpc of a product family). *Let $\mathcal{A}$ be an FMCA. Then the mpc of a product family $\mathcal{A}$ is $\mathcal{K}_{\mathcal{A}} = \bigcup_{p \in CP(\mathcal{A})} \mathcal{K}_{\mathcal{A}_p}$ and:*

$$\forall p \in [\![\varphi_{\mathcal{A}}]\!] : \mathscr{L}(\mathcal{K}_{\mathcal{A}_p}) \neq \emptyset \textit{ implies } \mathcal{K}_{\mathcal{A}_p} \preceq \mathcal{K}_{\mathcal{A}} \tag{8}$$

$$\forall p' \in CP(\mathcal{A})\ \exists p'' \in [\![\varphi_{\mathcal{A}}]\!] : \mathscr{L}(\mathcal{K}_{\mathcal{A}_{p''}}) \neq \emptyset \textit{ and } \mathcal{K}_{\mathcal{A}_{p''}} \npreceq \bigcup_{p \in CP(\mathcal{A}),\, p \neq p'} \mathcal{K}_{\mathcal{A}_p} \tag{9}$$

Note that the information about required actions can be discarded because we are only considering non-empty controllers.

Example 5.19. Each non-empty *mpc* $\mathcal{K}$ of a product family $\mathcal{A}$ described in Sect. 2 is exactly the *mpc* of its canonical product p_1. If, by extending the feature model, we were to have two other products p_5 and p_6 with non-empty *mpc*, such that $p_5 \equiv p_6$, $p_6 \preceq p_5$, $p_6 \preceq p_1$ and $p_5 \not\equiv p_1$, then p_5 would be an additional canonical product with *mpc* $\mathcal{K}' \npreceq \mathcal{K}$. The *mpc* of the product family would be the union of the *mpc* of p_1 and p_5 (i.e. $\mathcal{K}' \cup \mathcal{K}$).

We now prove that it is possible to build the *mpc* of product p of $\mathcal{A}$ starting from an FMCA smaller than $\mathcal{A}$, viz. from the intersection of the *mpc* of the immediate super-products of p. This suggests a way to efficiently compute the synthesis.

Theorem 5.20 (Quick mpc synthesis). *Let $\mathcal{A}$ be an FMCA, let $\ell \in \mathbb{N}$, let $Depth(\mathcal{A}, \ell) = \{\, p \mid p \in [\![\varphi_{\mathcal{A}}]\!]$ and $|Required(p)| + |Forbidden(p)| = \ell \,\}$ and let $p \in Depth(\mathcal{A}, n)$. Then it holds that:*

$$\textit{if } \mathcal{K}_{\mathcal{A},p} \neq \langle\rangle \textit{ then } \mathcal{K}_{\mathcal{A},p} \preceq \bigcap_{p' \in Depth(\mathcal{A}, n-1) \neq \emptyset,\ p \preceq p'} \mathcal{K}_{\mathcal{A},p'} \tag{10}$$

Example 5.21. Continuing Example 5.19, one could compute the *mpc* of p_6 starting from the intersection of the *mpc* of p_1 and p_5.

6 RELATED WORK AND CONCLUSION

In this paper, we presented FMCA as a novel formal model for expressing contract-based dynamic service product lines.

The definition of FMCA builds on two of the better known automata-based models proposed for modelling and analysing variability in product lines, which are typically based on superimposing multiple product automata in a single, enriched family automaton. From Modal Transition Systems (MTS) [3], FMCA inherit the distinction into permitted (may) and necessary (must) transitions, whereas the explicit incorporation of feature constraints stems from Featured Transition Systems (FTS) [16]. MTS were first recognised as a suitable behavioural model for describing product lines in [20], which provided an algorithm to check the conformance of product behaviour against that of the product family. Subsequent extensions involving notions from interface automata and I/O automata were defined in [24] and [25], respectively. Another line of research led to MTS with an associated set of variability constraints expressed over actions and a dedicated variability model checker that allows one to verify a property for a family and conclude that the result also holds for all its products [10, 12].

Compared to FMCA, none of these models can explicitly handle *dynamic* product lines, a characteristic FMCA inherits from [6, 7]. Furthermore, we tackled the problem of synthesising the *mpc* of a family of service contracts fulfilling all feature constraints, all necessary service requests and the maximal number of permitted service requests. Building on an earlier model [8], permitted and necessary transitions are interpreted as controllable and uncontrollable transitions in Supervisory Control Theory [15]. In this paper, necessary transitions are further distinguished according to their degree of controllability. The synthesis algorithm is thus enriched to consider necessary transitions whose controllability can be altered based on non-local information. Supervisory Control Theory was previously applied to Software Product Line Engineering in [13], where the `CIF 3` toolset was used to synthesise all valid products of a product line composed of behavioural components and requirements modelled as automata.

Based on the synthesis of the *mpc* in [8], our approach to synthesise a family of services does not consider all actions to be controllable, as in [13], but considers increasing levels of uncontrollability (from urgent to lazy requests). The information related to the specific requirements of each product (required and forbidden features) is also integrated into the synthesis algorithm. Moreover, whilst the number of products is in general exponential in the number of features, the organisation of the family's products (and their *mpc*) into a partial order makes our approach more scalable. As a result, the obtained *mpc* of the family of services can be synthesised from only a subset of its products, whereas other approaches require to synthesise the *mpc* of each single product. Finally, the presented theory has been implemented in a prototypical tool, which was used to compute all examples given throughout the paper.

It remains to compare our approach with other synthesis algorithms and to quantify how well it scales. To this aim, we would like to model and analyse a real world service-based application, as was done in [5] for a system without variability.

We conjecture the existence of a correspondence between FMCA and FTSs, which might allow to apply some of the simulation results from [18] concerning the preservation of important correctness properties. In particular, we would like to exploit the results from [18] to introduce a notion of uncontrollable disagreement in the refinement of FMCA, such that the refinements will coincide with the image of the synthesis function of Definition 5.10 (i.e. all and only refinements such that at least one product has a non-empty *mpc*). In order to adapt such simulation results directly to a partial order of FMCA, it remains to extend the results from [18] towards the refinement of FMCA, which is planned as future work.

Another direction for future work is to enhance service requests and offers with quantities. In Sect. 2, e.g., Clients could express the actual amount of money they are willing to pay. Reaching an agreement would then amount to finding the optimal trade-off among principals such that each one has a positive pay-off function. This might lead to a formalisation of Quality of Service parameters of Service Level Agreements in our model, allowing us to assess non-functional parameters like reliability or energy consumption in a composition of service contracts.

REFERENCES

[1] M. Acher, P. Collet, A. Gaignard, P. Lahire, J. Montagnat, and R.B. France. 2012. Composing multiple variability artifacts to assemble coherent workflows. *Softw. Qual. J.* 20, 3 (2012), 689–734.

[2] M. Acher, P. Collet, P. Lahire, and J. Montagnat. 2008. Imaging Services on the Grid as a Product Line: Requirements and Architecture. In *Proceedings SOAPL.*

[3] A. Antonik, M. Huth, K.G. Larsen, U. Nyman, and A. Wąsowski. 2008. 20 Years of Modal and Mixed Specifications. *B. EATCS* 95 (2008), 94–129.

[4] M. Bartoletti, T. Cimoli, and R. Zunino. 2015. Compliance in Behavioural Contracts: A Brief Survey. In *Programming Languages with Applications to Biology and Security (LNCS)*, Vol. 9465. Springer, 103–121.

[5] D. Basile, S. Chiaradonna, F. Di Giandomenico, and S. Gnesi. 2016. A stochastic model-based approach to analyse reliable energy-saving rail road switch heating systems. *J. Rail Transp. Plann. Man.* 6, 2 (2016), 163–181.

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

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

[8] D. Basile, F. Di Giandomenico, S. Gnesi, P. Degano, and G.L. Ferrari. 2017. Specifying Variability in Service Contracts. In *Proceedings VaMoS.* ACM, 20–27.

[9] D.S. Batory. 2005. Feature Models, Grammars, and Propositional Formulas. In *Proceedings SPLC (LNCS)*, Vol. 3714. Springer, 7–20.

[10] M.H. ter Beek, A. Fantechi, S. Gnesi, and F. Mazzanti. 2016. Modelling and analysing variability in product families: Model checking of modal transition systems with variability constraints. *J. Log. Algebr. Meth. Program.* 85 (2016), 287–315.

[11] M.H. ter Beek, S. Gnesi, and M.N. Njima. 2011. Product Lines for Service Oriented Applications - PL for SOA. In *Proceedings WWV (EPTCS)*, Vol. 61. 34–48.

[12] M.H. ter Beek and F. Mazzanti. 2014. VMC: Recent Advances and Challenges Ahead. In *Proceedings SPLC*, Vol. 2. ACM, 70–77.

[13] M.H. ter Beek, M.A. Reniers, and E.P. de Vink. 2016. Supervisory Controller Synthesis for Product Lines Using CIF 3. In *Proceedings ISoLA (LNCS)*, Vol. 9952. Springer, 856–873.

[14] D. Benavides, S. Segura, and A. Ruiz-Cortés. 2010. Automated Analysis of Feature Models 20 Years Later: a Literature Review. *Inf. Syst.* 35, 6 (2010), 615–636.

[15] C.G. Cassandras and S. Lafortune. 2006. *Introduction to Discrete Event Systems.* Springer, New York.

[16] A. Classen, M. Cordy, P.-Y. Schobbens, P. Heymans, A. Legay, and J.-F. Raskin. 2013. 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.

[17] G. Cledou and L.S. Barbosa. 2017. Modeling Families of Public Licensing Services: A Case Study. In *Proceedings FormaliSE.* ACM.

[18] M. Cordy, A. Classen, G. Perrouin, P.-Y. Schobbens, P. Heymans, and A. Legay. 2012. Simulation-based abstractions for software product-line model checking. In *ICSE.* IEEE, 672–682.

[19] K. Czarnecki and A. Wąsowski. 2007. Feature Diagrams and Logics: There and Back Again. In *Proceedings SPLC.* IEEE, 23–34.

[20] D. Fischbein, S. Uchitel, and V.A. Braberman. 2006. A foundation for behavioural conformance in software product line architectures. In *Proceedings ROSATEA.* ACM, 39–48.

[21] D. Georgakopoulos and M.P. Papazoglou (Eds.). 2008. *Service-oriented Computing.* MIT Press, Cambridge, MA, USA.

[22] S. Gunther and T. Berger. 2008. Service-Oriented Product Lines: A Development Process and Feature Management Model for Web Services. In *Proceedings SOAPL.*

[23] A. Halin, A. Nuttinck, M. Acher, X. Devroey, G. Perrouin, and P. Heymans. 2017. Yo Variability! JHipster: A Playground for Web-apps Analyses. In *Proceedings VaMoS.* ACM, 44–51.

[24] K. Larsen, U. Nyman, and A.Wąsowski.2007.Modal I/OAutomata for Interface and Product Line Theories. In *Proceedings ESOP (LNCS)*, Vol. 4421. Springer, 64–79.

[25] K. Lauenroth, K. Pohl, and S. Töhning. 2009. Model Checking of Domain Artifacts in Product Line Engineering. In *Proceedings ASE.* IEEE, 269–280.

[26] M. Mannion. 2002. Using First-Order Logic for Product Line Model Validation. In *Proceedings SPLC (LNCS)*, Vol. 2379. Springer, 176–187.

[27] F.M. Medeiros, E.S. de Almeida, and S.R. de Lemos Meira. 2009. Towards an Approach for Service-Oriented Product Line Architectures. In *Proceedings SOAPL.*

[28] M. Raatikainen, V. Myllärniemi, and T. Männistö. 2007. Comparison of Service and Software Product Family Modeling. In *Proceedings SOAPL.*

[29] P.J. Ramadge and W.M. Wonham. 1987. Supervisory control of a class of discrete event processes. *SIAM J. Control Optim.* 25, 1 (1987), 206–230.

[30] P.-Y. Schobbens, P. Heymans, J.-C. Trigaux, and Y. Bontemps. 2006. Feature Diagrams: A Survey and a Formal Semantics. In *Proceedings RE.* IEEE, 136–145.

7 DEFINITIONS

We define projection and composition mentioned in Sect. 4 in detail.

Definition 7.1 (Projection). Let $\mathcal{A}$ be an FMCA. Then the *projection* $\prod^i(\mathcal{A}) = \langle \prod^i(Q), \vec{q_0}_{(i)}, \prod^i(A^{\diamond}), \prod^i(A^{\Box_u}), \prod^i(A^{\Box_g}), \prod^i(A^{\Box_\ell}), \prod^i(A^o), \prod^i(T), \prod^i(\varphi), \prod^i(F)\rangle$ on the ith principal, $i \in 1 \dots n$, is s.t.

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

The (associative) composition operation $\boxtimes$ first extracts from its operands the principals they are composed of through projection, and then reassembles them through composition $\otimes$.

Definition 7.2 (A-composition). Let $\mathcal{A}_1, \mathcal{A}_2$ be two composable FMCA of rank n and m, respectively, and let $I = \{\, \prod^i(\mathcal{A}_1) \mid 0 < i \leq n \,\} \cup \{\, \prod^j(\mathcal{A}_2) \mid 0 < j \leq m \,\}$. Then the *a-product composition* of $\mathcal{A}_1$ and $\mathcal{A}_2$ is $\mathcal{A}_1 \boxtimes \mathcal{A}_2 = \bigotimes_{\mathcal{A}_i \in I} \mathcal{A}_i$.

8 PROOFS

We provide here the proofs that have been omitted from the paper.

THEOREM 4.18 (QUICK PRODUCTS VALIDATION). *Let $\mathcal{A}$ be a FMCA and $p, p' \in [\![\varphi_{\mathcal{A}}]\!]$ be two products s.t. $p \leq p'$. Then the following holds:*

$$p \textit{ is valid in } \mathcal{A} \textit{ implies } p' \textit{ is valid in } \mathcal{A} \tag{1}$$

PROOF. Intuitively, p imposes more restrictions on validity than p'. By contradiction, assume that p is valid in $\mathcal{A}$ and that p' is not valid in $\mathcal{A}$. By Definition 4.17, it must be the case that either

(1) $\exists a \in Required(p')$ s.t. $\forall (\vec{q}, \vec{a}, \vec{q'}) \in T_{\mathcal{A}}$: $\vec{a}$ is an action on $b \neq a$ or $\vec{q'} \in Dangling(\mathcal{A})$. In this case, p is not valid in $\mathcal{A}$ because by Definition 3.1, $Required(p') \subseteq Required(p)$.

(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 $\vec{q'} \notin Dangling(\mathcal{A})$. In this case, p is not valid in $\mathcal{A}$ as by Definition 3.1, $Forbidden(p') \subseteq Forbidden(p)$, a contradiction. □

LEMMA 8.1. *Let $\mathcal{K}_{\mathcal{A}_p}$ be the mpc of product p of $\mathcal{A}$. Then we have:*

$$\mathscr{L}(\mathcal{K}_{\mathcal{A}_p}) \neq \emptyset \textit{ implies } p \textit{ valid in } \mathcal{K}_{\mathcal{A}_p} \tag{2}$$

$$p \textit{ valid in } \mathcal{K}_{\mathcal{A}_p} \textit{ and } Required(p) \neq \emptyset \textit{ implies } \mathscr{L}(\mathcal{K}_{\mathcal{A}_p}) \neq \emptyset \tag{3}$$

$$\mathscr{L}(\mathcal{K}_{\mathcal{A}_p}) \neq \emptyset \wedge \exists p_1 : p_1 \leq p \textit{ implies } \exists p_2 : p_2 \leq p \wedge \mathscr{L}(\mathcal{K}_{\mathcal{A}_{p_2}}) \neq \emptyset \tag{4}$$

PROOF. For Equation (2), by contradiction assume that p is not valid in $\mathcal{K}_{\mathcal{A}_p}$. By Definition 4.17, it must be the case that either

(1) $\exists a \in Required(p)$ 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 $\vec{q'} \in Dangling(\mathcal{K}_{\mathcal{A}_p})$. In this case, $\nexists w \in \mathscr{L}(\mathcal{K}_{\mathcal{A}_p})$ s.t. w contains basic action a and thus, by Definition 5.6, $\mathcal{K}_{\mathcal{A}_p}$ is not an *mpc*, a contradiction.

(2) $\exists b \in Forbidden(p)$ s.t. $(\vec{q}, \vec{b}, \vec{q'}) \in T_{\mathcal{K}_{\mathcal{A}_p}}$, $\vec{b}$ is an action on b and $\vec{q'} \notin Dangling(\mathcal{K}_{\mathcal{A}_p})$. 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 5.4 some state in $\mathcal{K}_{\mathcal{A}_p}$ is in uncontrollable disagreement, a contradiction.

For Equation (3), by hypothesis $\exists a \in Required(p)$ s.t. $(\vec{q}, \vec{a}, \vec{q'}) \in T_{\mathcal{K}_{\mathcal{A}_p}}$ on a and $\vec{q'} \notin Dangling(\mathcal{K}_{\mathcal{A}_p})$, and $\mathscr{L}(\mathcal{K}_{\mathcal{A}_p}) \neq \emptyset$ by Definition 4.9.

For Equation (4), it suffices to note that a sub-product p_2 can be obtained by adding an action $a \notin Required(p) \cup Forbidden(p)$ to either (i) $Required(p_2)$ or (ii) $Forbidden(p_2)$ (the existence of such action a is guaranteed by hypothesis). The action a is either present or not present in $\mathcal{K}_{\mathcal{A}_p}$. If action a is present, the sub-product obtained through case (i) ($a \in Required(p_2)$) is such that $\mathscr{L}(\mathcal{K}_{\mathcal{A}_{p_2}}) \neq \emptyset$ by hypothesis (we are requiring an action that is present). If action a is not present, the sub-product obtained through case (ii) ($a \in Forbidden(p_2)$) is such that $\mathscr{L}(\mathcal{K}_{\mathcal{A}_{p_2}}) \neq \emptyset$ by hypothesis (we are forbidding an action that is not present). □

Next we state some auxiliary results used to prove Theorem 5.11.

LEMMA 8.2. *Let $\mathcal{A}$ be an FMCA, $p \in [\![\varphi_{\mathcal{A}}]\!]$ be a valid product, $\mathcal{K}_n$ be the FMCA (of p) computed through Definition 5.10 and R_n be the set of states computed through Definition 5.10. Then*

$$\forall \vec{q} \in R_n : \vec{q} \textit{ is unreachable in } \mathcal{K}_n \textit{ or } \vec{q_0} \in R_n \tag{11}$$

$$\textit{let } \ U_{p\mathcal{A}\mathcal{K}_n} = \{\vec{q} \mid \vec{q} \textit{ is in unc.dis. in } p \textit{ of } \mathcal{A} \textit{ by } \mathcal{K}_n\} \textit{ then}$$
$$U_{p\mathcal{A}\mathcal{K}_n} \cup Dangling(\mathcal{A}) = R_n \tag{12}$$

$$(\exists w \textit{ not containing basic actions } a \in Forbidden(p) \textit{ s.t.}$$
$$S_w = \{\, \vec{q} \mid (\vec{q_0} \xrightarrow{w_1}{}^{*} \vec{q} \xrightarrow{w_2}{}^{*} \vec{q_f}) \wedge (w = w_1 w_2 \in \mathscr{L}(\mathcal{A}) \cap \mathfrak{A}) \,\} \neq \emptyset$$
$$\textit{and } S_w \cap U_{p\mathcal{A}\mathcal{K}_n} = \emptyset) \textit{ implies } S_w \cap R_n = \emptyset \tag{13}$$

PROOF. We prove Equation (11). By contradiction, assume $\vec{q_0} \notin R_n$ and exists a path $p = \vec{q_0} \cdots \vec{q}\vec{q'} \cdots \vec{q''}$ such that $\vec{q'}, \dots, \vec{q''} \in R_n$ and $\vec{q} \notin R_n$. Let i be an iteration of the algorithm in Definition 5.10 such that $\vec{q'} \in R_{i-1}, \vec{q} \notin R_{i-1}$, and let $t = \vec{q} \rightarrow \vec{q'}$ be the transition traversed in p. If t is controllable, then by Definition 5.10 it is removed in $\mathcal{K}_n$, a contradiction. Otherwise, if t is uncontrollable, then $\vec{q}$ is added to R_i by Definition 5.10 and $\vec{q} \in R_n$, a contradiction.

We now prove Equation (12).

We start with $U_{p\mathcal{A}\mathcal{K}_n} \cup Dangling(\mathcal{A}) \subseteq R_n$. By contradiction, assume $\exists\, \vec{q} \in U_{p\mathcal{A}\mathcal{K}_n} \cup Dangling(\mathcal{A})$ s.t. $\vec{q} \notin R_n$. By Definition 5.10 $Dangling(\mathcal{A}) \subseteq R_0 \subseteq R_n$. By Definition 5.4, there exists a trace w s.t. $\vec{q} \xrightarrow{w}{}^{*} \vec{q_1}$ by only executing forced transitions, and either (1) $w \notin \mathfrak{A}$ or containing a basic action $a \in Forbidden(p)$ or (2) $\nexists w' \in \mathfrak{A}$ not containing basic actions $a \in Forbidden(p)$ s.t. $\vec{q_1} \xrightarrow{w'}{}^{*} \vec{q_f} \in F_{\mathcal{A}}$.

For case (1) we first prove that there exists a state $\vec{q'} \in R_0$ reachable by only executing forced transitions from $\vec{q}$ in $\mathcal{A}$ and $\vec{q} \xrightarrow{w1}{}^{*} \vec{q'} \xrightarrow{\vec{a}}$ such that either $\vec{a}$ is a request or it is forbidden in p. If $\vec{q'} \xrightarrow{\vec{a}}$ is controllable, then by Definition 5.4 it is the only outgoing transition from $\vec{q'}$, which is removed in $\mathcal{K}_0$ by Definition 5.10 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 5.10. 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 $\vec{q} \rightarrow^{*} \vec{q'}$. For the base case we have a transition $t = \vec{q} \rightarrow \vec{q'}$. Similarly to the previous reasoning, if t is controllable in $\mathcal{K}_1$ then by Definition 5.10 it is removed in $\mathcal{K}_1$, and by Definition 5.4, 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 5.10, $\vec{q} \in R_1 \subseteq R_n$, a contradiction. For the inductive step we have $\vec{q} \rightarrow \vec{q}'' \rightarrow^* \vec{q}'$ 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 (2) it is not possible to reach a final state $\vec{q}_f$ 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 in $\vec{q}' \in R_0$ (it is not difficult to see that otherwise a trace without requests and forbidden actions would exists). By Definition 5.10, there will be an iteration i s.t. $\vec{q}_1 \in R_i$, and we can prove $\vec{q} \in R_n$ by proceeding as for case (1).

We now show $R_n \subseteq U_{p\mathcal{A}\mathcal{K}_n} \cup \mathit{Dangling}(\mathcal{A})$. The proof is by induction on R_i. The base case is R_0. From Definition 5.10 follows that $\mathit{Dangling}(\mathcal{A}) \subseteq R_0$. We have the case $\vec{q} \in \mathit{Dangling}(\mathcal{K}_0) \setminus \mathit{Dangling}(\mathcal{A})$ where all transitions t with source in $\vec{q}$ have been pruned in $\mathcal{K}_0$. This means that from $\vec{q}$ it is not possible to reach a final state $\vec{q}_f$ without firing either a request or a forbidden action a, hence $\vec{q} \in U_{p\mathcal{A}\mathcal{K}_n}$. The last case is $\{ \vec{q} \mid (\vec{q} \rightarrow) = t \in T^{\Box}_{\mathcal{A}}$ *on a unc. in* $\mathcal{K}_0$, $(t\ \mathit{request} \vee a \in \mathit{Forbidden}(p)) \}$, i.e. by Definition 5.4, in $U_{p\mathcal{A}\mathcal{K}_n}$. Note that if a transition t is unc. in $\mathcal{K}_i$ for some i, then for all $j:\ i \leq j \leq n, t$ is unc. in $\mathcal{K}_j$.

For the inductive step, by inductive hypothesis we know $R_{i-1} \subseteq U_{p\mathcal{A}\mathcal{K}_n} \cup \mathit{Dangling}(\mathcal{A})$ and we prove $R_i \subseteq U_{p\mathcal{A}\mathcal{K}_n} \cup \mathit{Dangling}(\mathcal{A})$.

We proceed again by cases on Definition 5.10. First case is $S = \{ \vec{q} \mid (\vec{q} \rightarrow \vec{q}') \in T^{\Box}_{\mathcal{K}_i}$ *unc. in* $\mathcal{K}_i,\ \vec{q} \notin R_{i-1} \wedge \vec{q}' \in R_{i-1} \} \cup \{ \vec{q} \mid \vec{q} \rightarrow\ \in T^{\Box}_{\mathcal{A}}$ *uglr or ulm in* $\mathcal{K}_i \}$. By Definition 5.4 the transitions used in S are forced in $\mathcal{A}$ because are unc. in $\mathcal{K}_i$ (hence in $\mathcal{K}_n$); and by inductive hypothesis their target state is in $U_{p\mathcal{A}\mathcal{K}_n} \cup \mathit{Dangling}(\mathcal{A})$. It follows that $S \subseteq U_{p\mathcal{A}\mathcal{K}_n} \cup \mathit{Dangling}(\mathcal{A})$.

Second (and last) case is $\mathit{Dangling}(\mathcal{K}_i)$. These states $\vec{q}$ have all outgoing transitions pruned because (by Definition 5.10) either $\vec{q} \in R_{i-1}$, and the thesis follows by inductive hypothesis, or all their outgoing transitions were controllable in $\mathcal{K}_{i-1}$ and with target state in R_{i-1}. Similarly to the base case, this means that from $\vec{q}$ it is not possible to reach a final state $\vec{q}_f$ without passing through a state in $U_{p\mathcal{A}\mathcal{K}_n} \cup \mathit{Dangling}(\mathcal{A})$, and the thesis follows.

We now prove Equation (13). By hypothesis $S_w \cap \mathit{Dangling}(\mathcal{A}) = \emptyset$, and the thesis follows by Equation (12). □

Theorem 5.11 (mpc of a 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 5.10 is the mpc of 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 $\mathcal{A}$.

We first ensure that the algorithm always terminates by proving the existence of the least fixed point of f. The function f is monotonic because it is defined on the cpo P and at each iteration the set R_i can only increase, hence by the Knaster-Tarski theorem the least fixed point of f exists.

We now 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) $\mathit{Dangling}(\mathcal{K}) = \emptyset$ (trivial), (3) $\mathscr{L}(\mathcal{K}) = \emptyset$ or $\forall a \in \mathit{Required}(p)\ \exists w \in \mathscr{L}(\mathcal{K})$ s.t. w contains basic action a, and (4) $\nexists w \in \mathscr{L}(\mathcal{K})$ s.t. w contains actions $a \in \mathit{Forbidden}(p)$ or $\vec{q_0}_{\mathcal{K}} \xrightarrow{w}{}^* \vec{q}_{\mathcal{K}}, \vec{q_0} \xrightarrow{w}{}^* \vec{q}$ and $\vec{q}$ is in uncontrollable disagreement in p of $\mathcal{A}$.

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 $\vec{q} \xrightarrow{\vec{a}\Box}$, 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 5.10. By Lemma 8.2(11), we have $\vec{q_0} \in R_n$ and 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 5.10.

We conclude by proving (4). Assume $\vec{q_0}_{\mathcal{K}} \xrightarrow{w}{}^* \vec{q}_{\mathcal{K}} \xrightarrow{\vec{a}}$ with $\vec{a}$ on action $a \in \mathit{Forbidden}(p)$ holds. By Definition 5.10 $\vec{q}_{\mathcal{K}} \xrightarrow{\vec{a}}$ 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 $\mathcal{K}_{\mathcal{A}_p}$, contradiction. Assume that $(w, \vec{q_0}_{\mathcal{K}}) \xrightarrow{w}{}^* (\varepsilon, \vec{q}_{\mathcal{K}})$ and $(w, \vec{q_0}) \xrightarrow{w}{}^* (\varepsilon, \vec{q})$, with $\vec{q}$ in uncontrollable disagreement holds. Since $\mathcal{K}_{\mathcal{A}_p}$ is derived from $\mathcal{A}$, we have $\vec{q_0}_{\mathcal{K}} = \vec{q_0}$ and $\vec{q}_{\mathcal{K}} = \vec{q}$. By Lemma 8.2(12), $\vec{q} \in R_n$ and since it is reachable from $\vec{q_0}$, by Lemma 8.2(11), it must be that $\vec{q_0} \in R_n$ 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 $w_1 \in \mathscr{L}(\mathcal{K}')$ which is such that $w_1 \notin \mathscr{L}(\mathcal{K}_{\mathcal{A}_p})$. Let S_{w1} be the set of states traversed by $\mathcal{K}'$ to recognise w_1. Since $\mathcal{K}'$ is a controller, $S_{w1} \cap U_{p\mathcal{A}\mathcal{K}_n} = \emptyset$. By Lemma 8.2(13), $S \cap R_n = \emptyset$. Then, by Definition 5.10 all states in S_{w1} are in $\mathcal{K}_{\mathcal{A}_p}$. Moreover, all transitions used for recognizing w_1 are not requests nor forbidden because $\mathcal{K}'$ is a controller, and since $S \cap R_n = \emptyset$ no state in S is in any of $R_0, \ldots, R_n$. By Definition 5.10 these are all possible cases for which a transition is removed. Hence, all transitions used for recognizing 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 following lemma is auxiliary and useful for proving the following results.

Lemma 8.3. *Let $\mathcal{A}$ be an FMCA, $p \in [\![\varphi]\!]$, and $\mathcal{K}_{\mathcal{A}_p} \neq \langle\rangle$ be its mpc computed through Definition 5.10. Then*

$$\mathcal{K}_{\mathcal{A}_p} \preceq \mathcal{A}$$

Proof. The refinement relation is exactly $\mathcal{R} = \{(\vec{q}_{\mathcal{A}}, \vec{q}_{\mathcal{K}}) \mid \vec{q}_{\mathcal{K}} = \vec{q}_{\mathcal{A}} \in Q_{\mathcal{K}}\}$. From Theorem 5.11, we know that $Q_{\mathcal{K}} \subseteq Q$, $Q_{\mathcal{K}} \cap \mathit{Dangling}(\mathcal{K}_{\mathcal{A}_p}) = \emptyset$ and $T^{\Diamond}_{\mathcal{K}} \subseteq T^{\Diamond}$. It remains to prove that (1) all uncontrollable transitions of $\mathcal{A}$ and (2) all controllable transitions of $\mathcal{A}$ that are uncontrollable in $\mathcal{K}_{\mathcal{A}_p}$, both (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 $t = (\vec{q}, \vec{a}, \vec{q}')$ s.t. t is uncontrollable in $\mathcal{A}$ and $t \notin T^{\Box}_{\mathcal{K}}$. By Definition 5.10 $t \notin T^{\Box}_{\mathcal{K}}$ only if $\vec{q} \in R_n$, and by Lemma 8.2(12) $\vec{q} \in U_{p\mathcal{A}\mathcal{K}_n}$. If $\vec{q} \in U$, then by Lemma 8.2(11) $\mathcal{K}_{\mathcal{A}_p} = \langle\rangle$, a contradiction.

For (2), by contradiction, let $\vec{q} \in Q_{\mathcal{K}}$, and let $t = (\vec{q}, \vec{a}, \vec{q}')$ s.t. t is controllable in $\mathcal{A}$, uncontrollable in $\mathcal{K}_{\mathcal{A}_p}$ and $t \notin T^{\Box}_{\mathcal{K}}$. Hence t must be either *uglr* or *ulm* in $\mathcal{K}_{\mathcal{A}_p}$, and in both cases $\vec{q} \in R_n$ by Definition 5.10. Finally, by Lemma 8.2(12) $\vec{q} \in U_{p\mathcal{A}\mathcal{K}_n}$, and by Lemma 8.2(11) $\mathcal{K}_{\mathcal{A}_p} = \langle\rangle$, a contradiction. □

Theorem 5.12 (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 5.10 and let $\Pi_i(\mathcal{K}_{\mathcal{A}_p}) = \mathcal{A}_{r_i}$ be its projections on ith principals. Then the following holds:*

$$\forall i \in I : \mathcal{A}_{r_i} \preceq \mathcal{A}_i \quad (5)$$

$$\forall \mathcal{K}' \neq \langle\rangle \text{ controller of } p \text{ of } \mathcal{A} \text{ and } \forall i : \Pi_i(\mathcal{K}') = \mathcal{A}_{r'_i} \preceq \mathcal{A}_{r_i} \quad (6)$$

Proof. Equation (5) follows from Lemma 8.3 and Definition 7.1. To prove Equation (6), assume by contradiction that for some i we have $\mathcal{A}_{r'_i} \not\preceq \mathcal{A}_{r_i}$. By Equation (5) and Lemma 8.3, there must be $t \in T^{\diamond}_{\mathcal{K}'} \setminus T^{\diamond}_{\mathcal{K}_{\mathcal{A}_p}}$. By Theorem 5.11 $\mathscr{L}(\mathcal{K}') \subseteq \mathscr{L}(\mathcal{K}_{\mathcal{A}_p})$, a contradiction. □

Lemma 8.4 (Controller refinement). *Let $\mathcal{A}$ be an FMCA and let $p, p' \in [\![\varphi_{\mathcal{A}}]\!]$. Then the following holds:*

$$p \preceq p' \text{ and } \mathscr{L}(\mathcal{K}_{\mathcal{A}_p}) \neq \emptyset \quad \text{implies} \quad \mathcal{K}_{\mathcal{A}_p} \preceq \mathcal{K}_{\mathcal{A}_{p'}} \quad (7)$$

Proof. By hypothesis all $a \in Required(p) \setminus Required(p')$ are in $\mathcal{K}_{\mathcal{A}_p}$, and also in $\mathcal{K}_{\mathcal{A}'_p}$, because it is an *mpc* and $p \preceq p'$.

Conversely all $a \in Forbidden(p) \setminus Forbidden(p')$ are not in $\mathcal{K}_{\mathcal{A}_p}$ but could be in $\mathcal{K}_{\mathcal{A}'_p}$, unless $\mathcal{K}_{\mathcal{A}_p} = \mathcal{K}_{\mathcal{A}'_p}$ (in this case the thesis follows trivially), because again $\mathcal{K}_{\mathcal{A}_p}$ is an *mpc* and $p \preceq p'$. Hence by hypothesis, Theorem 5.11 and Definition 3.1, we have $\emptyset \neq T_{\mathcal{K}_{\mathcal{A}_p}} \subseteq T_{\mathcal{K}_{\mathcal{A}'_p}}$. Moreover by Lemma 8.2(12) all states of both *mpc* are not in $U_{p\mathcal{A}\mathcal{K}_n}$ (resp. $U_{p'\mathcal{A}\mathcal{K}_n}$), hence (following the same reasoning as the proof of Lemma 8.3): (1) all uncontrollable transitions of $\mathcal{K}_{\mathcal{A}_{p'}}$ and (2) all controllable transitions of $\mathcal{K}_{\mathcal{A}_{p'}}$ that are uncontrollable in $\mathcal{K}_{\mathcal{A}_p}$, both (1–2) reachable, are also available in $\mathcal{K}_{\mathcal{A}_p}$. Otherwise, we would reach the contradiction that their source state is in $U_{p\mathcal{A}\mathcal{K}_n}$ (resp. $U_{p'\mathcal{A}\mathcal{K}_n}$) and $\mathcal{K}_{\mathcal{A}_p} \preceq \mathcal{K}_{\mathcal{A}_{p'}}$ holds. □

Theorem 5.18 (mpc of a product family). *Let $\mathcal{A}$ be an FMCA. Then the mpc of a product family $\mathcal{A}$ is $\mathcal{K}_{\mathcal{A}} = \bigcup_{p \in CP(\mathcal{A})} \mathcal{K}_{\mathcal{A}_p}$ and:*

$$\forall p \in [\![\varphi_{\mathcal{A}}]\!] : \mathscr{L}(\mathcal{K}_{\mathcal{A}_p}) \neq \emptyset \text{ implies } \mathcal{K}_{\mathcal{A}_p} \preceq \mathcal{K}_{\mathcal{A}} \quad (8)$$

$$\forall p' \in CP(\mathcal{A})\ \exists p'' \in [\![\varphi_{\mathcal{A}}]\!] : \mathscr{L}(\mathcal{K}_{\mathcal{A}_{p''}}) \neq \emptyset \text{ and } \mathcal{K}_{\mathcal{A}_{p''}} \not\preceq \bigcup_{p \in CP(\mathcal{A}),\, p \neq p'} \mathcal{K}_{\mathcal{A}_p} \quad (9)$$

Proof. Equation (8) follows straightforward from Lemma 5.14, Lemma 5.8 and hypothesis. For Equation (9), it suffices to consider $p' = p''$, after which by Definition 5.16 the statement follows. □

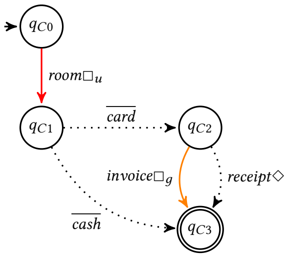
Theorem 5.20 (Quick mpc synthesis). *Let $\mathcal{A}$ be an FMCA, let $\ell \in \mathbb{N}$, let $Depth(\mathcal{A}, \ell) = \{ p \mid p \in [\![\varphi_{\mathcal{A}}]\!]$ and $|Required(p)| + |Forbidden(p)| = \ell \}$ and let $p \in Depth(\mathcal{A}, n)$. Then it holds that:*

$$\text{if } \mathcal{K}_{\mathcal{A},p} \neq \langle\rangle \text{ then } \mathcal{K}_{\mathcal{A},p} \preceq \bigcap_{p' \in Depth(\mathcal{A}, n-1) \neq \emptyset,\, p \preceq p'} \mathcal{K}_{\mathcal{A},p'} \quad (10)$$

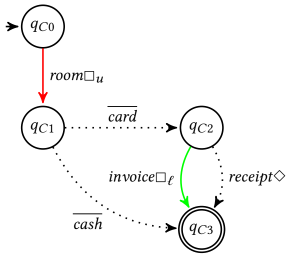
Proof. Straightforward from Lemma 5.14. □

9 FMCA OF MOTIVATING EXAMPLE

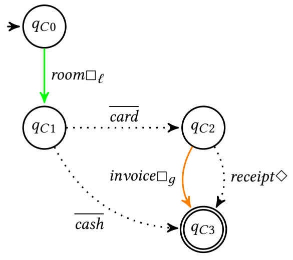
All the FMCA mentioned in the running example discussed throughout this paper are reported in Fig. 3.



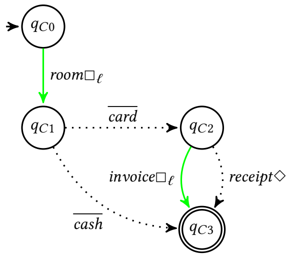
(a) BusinessClientG



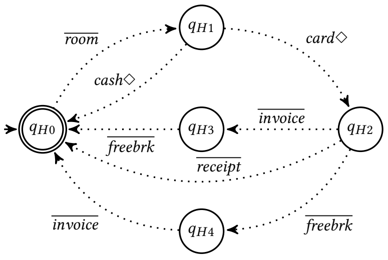
(b) BusinessClientL



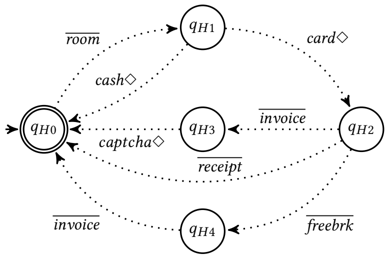
(c) EconomyClientG



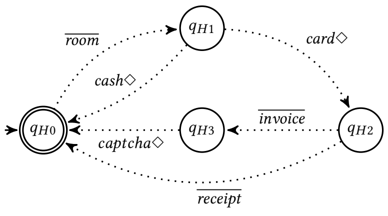
(d) EconomyClientL



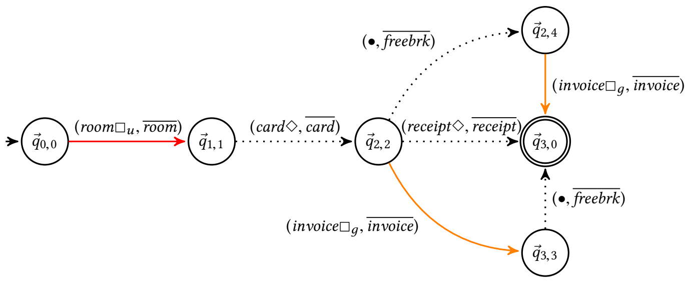
(e) Hotel



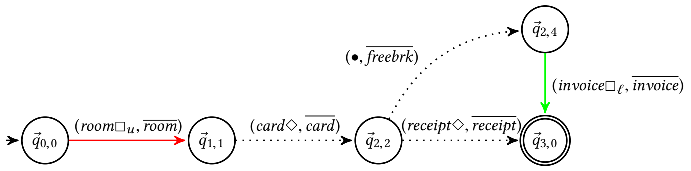
(f) HotelGreedyBad



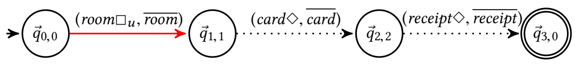
(g) HotelLazyBad


$\vec{q}_{0,0}$
$(room\Box_u, \overline{room})$
$\vec{q}_{1,1}$
$(card\Diamond, \overline{card})$
$\vec{q}_{2,2}$
$(\bullet, \overline{freebrk})$
$\vec{q}_{2,4}$
$(invoice\Box_g, \overline{invoice})$
$(receipt\Diamond, \overline{receipt})$
$\vec{q}_{3,0}$
$(\bullet, \overline{freebrk})$
$(invoice\Box_g, \overline{invoice})$
$\vec{q}_{3,3}$

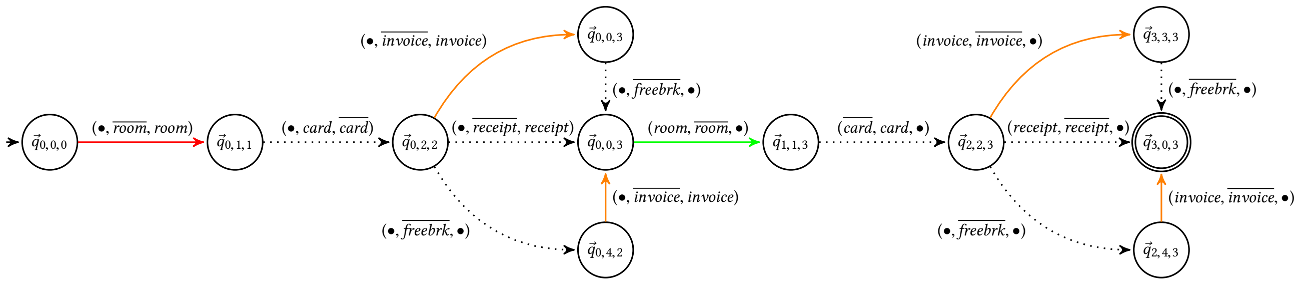
(h) $\mathcal{K}_{\text{BusinessClientG} \otimes \text{Hotel}_{p_1}}$


$\vec{q}_{0,0}$
$(room\Box_u, \overline{room})$
$\vec{q}_{1,1}$
$(card\Diamond, \overline{card})$
$\vec{q}_{2,2}$
$(\bullet, \overline{freebrk})$
$\vec{q}_{2,4}$
$(invoice\Box_\ell, \overline{invoice})$
$(receipt\Diamond, \overline{receipt})$
$\vec{q}_{3,0}$

(i) $\mathcal{K}_{\text{BusinessClientL} \otimes \text{HotelGreedyBad}_{p_1}}$


$\vec{q}_{0,0}$
$(room\Box_u, \overline{room})$
$\vec{q}_{1,1}$
$(card\Diamond, \overline{card})$
$\vec{q}_{2,2}$
$(receipt\Diamond, \overline{receipt})$
$\vec{q}_{3,0}$

(j) $\mathcal{K}_{\text{ill}}$


$\vec{q}_{0,0,0}$
$(\bullet, \overline{room}, room)$
$\vec{q}_{0,1,1}$
$(\bullet, card, \overline{card})$
$\vec{q}_{0,2,2}$
$(\bullet, \overline{invoice}, invoice)$
$\vec{q}_{0,0,3}$
$(\bullet, \overline{freebrk}, \bullet)$
$(\bullet, \overline{receipt}, receipt)$
$\vec{q}_{0,0,3}$
$(room, \overline{room}, \bullet)$
$\vec{q}_{1,1,3}$
$(\overline{card}, card, \bullet)$
$\vec{q}_{2,2,3}$
$(invoice, \overline{invoice}, \bullet)$
$\vec{q}_{3,3,3}$
$(\bullet, \overline{freebrk}, \bullet)$
$(receipt, \overline{receipt}, \bullet)$
$\vec{q}_{3,0,3}$
$(\bullet, \overline{invoice}, invoice)$
$(\bullet, \overline{freebrk}, \bullet)$
$\vec{q}_{0,4,2}$
$(invoice, \overline{invoice}, \bullet)$
$(\bullet, \overline{freebrk}, \bullet)$
$\vec{q}_{2,4,3}$

(k) $\mathcal{K}_{\text{EconomyClientG} \otimes (\text{Hotel} \otimes \text{BusinessClientG})} = \mathcal{K}_{(\text{EconomyClientG} \otimes \text{Hotel}) \boxtimes \text{BusinessClientG}}$

Figure 3: All the FMCA of the running example used throughout the paper.