A Model-Checking Tool for Families of Services

Patrizia Asirelli, Maurice H. ter Beek, Alessandro Fantechi, Stefania Gnesi

A Model-Checking Tool for Families of Services

P. Asirelli[1], M.H. ter Beek[1], A. Fantechi[1,2], and S. Gnesi[1]

[1] Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo", CNR, Pisa, Italy
{asirelli,terbeek,gnesi}@isti.cnr.it
[2] Dipartimento di Sistemi e Informatica, Università degli Studi di Firenze, Italy
fantechi@dsi.unifi.it

Abstract. We propose a model-checking tool for on-the-fly verification of properties expressed in a branching-time temporal logic based on a deontic interpretation of classical modal and temporal operators over modal transition systems. We apply this tool to the analysis of variability in behavioural descriptions of families of services.

1 Introduction

We present our ongoing research on an emerging topic in the engineering of distributed systems, right on the crossroads between (software) product line engineering and service computing [10, 23, 24]. Our aim is the development of rigorous modelling techniques as well as analysis and verification support tools for assisting organisations to plan, optimise, and control the quality of software service provision. To this aim, we foresee a flexible engineering methodology according to which software service line organisations can develop novel classes of service-oriented applications easily adaptable to customer requirements as well as to changes in the context in which they execute.

Product Line Engineering (PLE) is an approach to develop product families using a common platform and mass customisation [33]. It aims to lower production costs of individual products by letting them share an overall reference model of a product family, while allowing them to differ w.r.t. particular features in order to serve, e.g., different markets. As a result, the production process in PLE is organised so as to maximise commonalities of the products and at the same time minimise the cost of variations. Product variants can be derived from the product family, which allows for reuse and differentiation of a family's products. Variety of products from a single product platform is achieved by identifying variation points as places in design artifacts where a specific decision is reduced to the choice among several *features* but the feature to be chosen for a particular product is left open (like optional, mandatory or alternative features). Software Product Line Engineering (SPLE) is a paradigm for developing a diversity of software products and software-intensive systems based on the underlying architecture of an organisation's product platform [9, 36]. Variability management is the key aspect differentiating SPLE from 'conventional' software engineering.

Service-Oriented Computing (SOC) is a distributed computing paradigm [35]. Services are autonomous, distributed, and platform-independent computational

elements capable of solving specific tasks, which all need to be described, published, categorised, discovered, and then dynamically and loosely coupled in novel ways (orchestrated). They can be used to create distributed, interoperable, and dynamic applications and business processes which span organisational boundaries as well as computing platforms. In the end, SOC systems deliver application functionalities as services to either end-user applications or other services. Their underlying infrastructures are called Service-Oriented Architectures (SOAs).

We recently launched a research effort to, on the one hand, investigate the most promising existing modelling structures that allow (behavioural) variability to be described and product derivation to be defined, and, on the other hand, develop a proper temporal logic that can be interpreted over such structures and which can express interesting properties over families and products alike. In [3], we defined such a temporal logic and an efficient model-checking algorithm.

This paper is a first step towards extending our results to service families. We present a model checker based on a formal framework of vACTL (a variability and action-based branching-time temporal logic) with its natural interpretation structure (MTS, Modal Transition System). Product derivation is defined in our framework and logical formulae are used as variability constraints as well as behavioural properties to be verified for families and products alike. We apply our tool to analyse variability in behavioural descriptions of service families.

After presenting a simple running example in Sect. 2, we formally define MTSs in Sect. 3. In Sect. 4, we introduce vACTL and show how we can manage advanced variability in Sect. 5. In Sect. 6, we describe our tool and apply it to the running example. Related work is discussed in Sect. 7, before Sect. 8 concludes.

2 Running Example: Travel Agency

As motivating example, consider a software company developed a package on sale for those interested in starting a travel agency service (e.g. on the web). This company provides a choice among products of a family with different prices and features. All products provide the features *hotel*, *flight*, and *train* reservation: the coordination component uses *predefined* external services (one per business sector) to retrieve a list of quotes. These products can be enhanced in two ways:

1. By adding as *alternative* feature the possibility to choose, only for flights and hotels, from *multiple* external services in order to retrieve the best quotes through more than one service. This means that proper coordination addresses more hotel and flight services, and proper data fusion is done with the data received by the contacted external services.
2. By adding as *optional* feature the possibility for a customer to book a *leisure tour* during his/her stay at a hotel. This is achieved through an additional component that interacts with an external leisure tour service. However, as the provided tour packages may include a hotel in a different location for a subset of nights, a tour reservation *requires* interaction with the hotel service to offer a feature that allows to *cancel part* of the room reservations at the main hotel location for such nights. A coordination model variant can do so.

When combined, this choice of features leads to the following 8 different products.

features		variability	products							
			1	2	3	4	5	6	7	8
train reservation	predefined services	mandatory	√	√	√	√	√	√	√	√
hotel reservation	predefined services	alternative	√	√			√	√		
	multiple services				√	√			√	√
flight reservation	predefined services	alternative	√			√	√			√
	multiple services			√	√			√	√	
leisure tour reservation		optional					√	√	√	√
cancel part reservation		required					√	√	√	√

3 A Behavioural Model for Product Families

Modal Transition Systems (MTSs) [28] and variants are now an accepted formal model for defining behavioural aspects of product families [3, 12, 14, 27, 29]. An MTS is a Labelled Transition System (LTS) with a distinction between may and must transitions, seen as *optional* or *mandatory* features for a family's products. For a given product family, an MTS can model

- its *underlying behaviour*, shared among all products, and
- its *variation points*, differentiating between products.

An MTS cannot model advanced variability constraints regarding *alternative* features (only one of them may be present) nor those regarding inter-feature relations (a feature's presence *requires* or *excludes* that of another feature) [3]. We will formalise such advanced variability constraints by means of an associated set of logical formulae expressed in the variability and action-based branching-time temporal logic vaCTL that we will define in Sect. 4.

We now formally define MTSs and — to begin with — their underlying LTSs.

Definition 1. *A* Labelled Transition System *(LTS) is a quadruple* $(Q, A, \overline{q}, \delta)$*, with set* Q *of states, set* A *of actions, initial state* $\overline{q} \in Q$*, and transition relation* $\delta \subseteq Q \times A \times Q$*. We also write* $q \xrightarrow{a} q'$ *for* $(q, a, q') \in \delta$*.* □

To model behaviour of product families, we must define the evolution of time.

Definition 2. *Let* $(Q, A, \overline{q}, \delta)$ *be an LTS and* $q \in Q$*. Then* σ *is a* path *from* q *if* $\sigma = q$ *(an empty path) or* σ *is a (possibly infinite) sequence* $q_1 a_1 q_2 a_2 q_3 \cdots$ *such that* $q_1 = q$ *and* $q_i \xrightarrow{a_i} q_{i+1}$*, for all* $i > 0$*. A* full path *is a path that cannot be extended further, i.e., it is infinite or ends in a state with no outgoing transitions. The set of all full paths from* q *is denoted by* $path(q)$*.*

If $\sigma = q_1 a_1 q_2 a_2 q_3 \cdots$*, then its* i*-th state* q_i *is denoted by* $\sigma(i)$ *and its* i*-th action* a_i *is denoted by* $\sigma\{i\}$*.* □

In an MTS, transitions are defined to be possible (*may*) or mandatory (*must*).

Definition 3. *A Modal Transition System (MTS) is a quintuple* $(Q, A, \overline{q}, \delta^{\Box}, \delta^{\Diamond})$ *such that the quadruple* $(Q, A, \overline{q}, \delta^{\Box} \cup \delta^{\Diamond})$ *is an LTS, called its* underlying *LTS.*

An MTS has two transition relations: $\delta^{\Diamond} \subseteq Q \times A \times Q$ *is the* may *transition relation, expressing* possible *transitions, while* $\delta^{\Box} \subseteq Q \times A \times Q$ *is the* must *transition relation, expressing* mandatory *transitions. By definition,* $\delta^{\Box} \subseteq \delta^{\Diamond}$.

We also write $q \xrightarrow{a}_{\Box} q'$ *for* $(q, a, q') \in \delta^{\Box}$ *and* $q \xrightarrow{a}_{\Diamond} q'$ *for* $(q, a, q') \in \delta^{\Diamond}$. □

The inclusion $\delta^{\Box} \subseteq \delta^{\Diamond}$ formalises that mandatory transitions must also be possible. Reasoning on the existence of transitions is thus like reasoning with a 3-valued logic with the truth values *true*, *false*, and *unknown*: mandatory transitions ($\delta^{\Box}$) are *true*, possible but not mandatory transitions ($\delta^{\Diamond} \setminus \delta^{\Box}$) are *unknown*, and impossible transitions ($(q, a, q') \notin \delta^{\Box} \cup \delta^{\Diamond}$) are *false* [18].

The transition relations of MTSs allow the distinction of special type of paths.

Definition 4. *Let* $\mathcal{F}$ *be an MTS and* $\sigma = q_1 a_1 q_2 a_2 q_3 \cdots$ *a full path in its underlying LTS. Then* σ *is a* must path *(from* q_1*) in* $\mathcal{F}$*, denoted by* $\sigma^{\Box}$*, if* $q_i \xrightarrow{a_i}_{\Box} q_{i+1}$*, for all* $i > 0$*. The set of all must paths from* q_1 *is denoted by* $\Box\text{-}path(q_1)$. □

Recall that features are often used for compact representations of a family's products. To model such product family representations as MTSs one thus needs a 'translation' from features to actions (not necessarily a one-to-one mapping) and the introduction of a behavioural relation (temporal ordering) among them. A family's products are then considered to differ w.r.t. the actions they are able to perform in any given state of the MTS. This means that the MTS of a product family has to accommodate all the possibilities desired for each derivable product, predicating on the choices that make a product belong to that family.

The MTS in Fig. 1 models the Travel Agency product family of Sect. 2: edges labelled may(·) are possible but not mandatory transitions, whereas those labelled must(·) are mandatory.

Given an MTS description of a product family, an MTS describing a subfamily is obtained by preserving at least all must transitions and turning some of the may transitions (that are not must transitions) into must transitions as well as removing some of the remaining may transitions.

Definition 5. *Let* $\mathcal{F} = (Q, A, \overline{q}, \delta^{\Box}, \delta^{\Diamond})$ *be an MTS specifying a product family. A* subfamily *specified as an MTS* $\mathcal{F}_s = (Q_s, A, \overline{q}, \delta_s^{\Box}, \delta_s^{\Diamond})$ *is derived by considering* $\delta_s^{\Box} = \delta^{\Box} \cup R$*, with* $R \subseteq \delta^{\Diamond}$*, and* $\delta_s^{\Diamond} \subseteq \delta^{\Diamond}$*, defined over a set* $Q_s \subseteq Q$ *of states, so that* $\overline{q} \in Q_s$*, and every* $q \in Q_s$ *is reachable from* $\overline{q}$ *via transitions from* $\delta_s^{\Box} \cup \delta_s^{\Diamond}$.

More precisely, we say that $\mathcal{F}_s$ is a subfamily of $\mathcal{F}$*, denoted by* $\mathcal{F}_s \vdash \mathcal{F}$*, iff* $\overline{q}_s \vdash \overline{q}$*, where* $q_s \vdash q$ *holds, for some* $q_s \in Q_s$ *and* $q \in Q$*, iff:*

- *whenever* $q \xrightarrow{a}_{\Box} q'$*, for some* $q' \in Q$*, then* $\exists\, q'_s \in Q_s : q_s \xrightarrow{a}_{\Box} q'_s$ *and* $q'_s \vdash q'$*, and*
- *whenever* $q_s \xrightarrow{a}_{\Diamond} q'_s$*, for some* $q'_s \in Q_s$*, then* $\exists\, q' \in Q : q \xrightarrow{a}_{\Diamond} q'$ *and* $q'_s \vdash q'$. □

An LTS describing a product can be seen (i.e., obtained from an MTS description of a product family) as a subfamily containing only must transitions. Formally:

Definition 6. *Let* $\mathcal{F} = (Q, A, \overline{q}, \delta^{\Box}, \delta^{\Diamond})$ *be an MTS specifying a product family.*

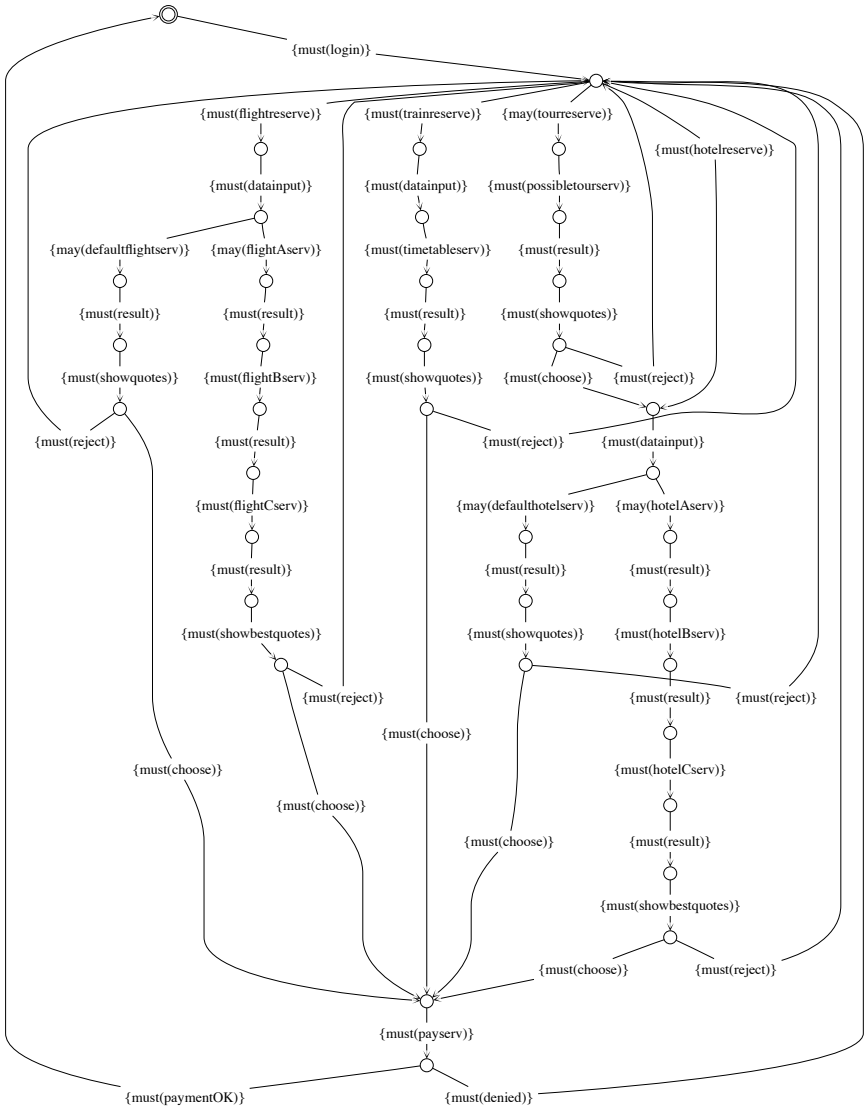

Fig. 1. MTS of the Travel Agency product family as produced by FMC.

A set of products *specified as a set of LTSs* $\{\,\mathcal{P}_i = (Q_i, A, \overline{q}, \delta_i) \mid i > 0\,\}$ *is derived by considering each transition relation* δ_i *to be* $\delta^{\Box} \cup R$*, with* $R \subseteq \delta^{\Diamond}$*, defined over a set of states* $Q_i \subseteq Q$*, so that* $\overline{q} \in Q_i$*, and every* $q \in Q_i$ *is reachable from* $\overline{q}$ *via transitions from* δ_i*.*

More precisely, we say that $\mathcal{P}_i$ is a product of $\mathcal{F}$*, denoted by* $\mathcal{P}_i \vdash \mathcal{F}$*, iff* $\overline{q}_i \vdash \overline{q}$*, where* $q_i \vdash q$ *holds, for some* $q_i \in Q_i$ *and* $q \in Q$*, iff:*

– *whenever* $q \xrightarrow{a}_{\Box} q'$*, for some* $q' \in Q$*, then* $\exists\, q'_i \in Q_i : q_i \xrightarrow{a}_i q'_i$ *and* $q'_i \vdash q'$*, and*
– *whenever* $q_i \xrightarrow{a}_i q'_i$*, for some* $q'_i \in Q_i$*, then* $\exists\, q' \in Q : q \xrightarrow{a}_{\Diamond} q'$ *and* $q'_i \vdash q'$*.* □

The subfamilies and products derived by Def. 5 and Def. 6 obviously might not satisfy the aforementioned advanced variability constraints that MTSs cannot model. However, as said before, we will show in Sect. 5 how to use the variability and action-based branching-time temporal logic vaCTL that we will define in Sect. 4 to express those constraints. Moreover, in [3] we outlined an algorithm to derive from an MTS all products that are valid w.r.t. constraints expressed in a temporal logic and we recently adapted this algorithm to vaCTL.

4 Logics for MTSs

In this section, we first introduce a minor extension of a well-known logic from the literature, after which we thoroughly extend the resulting logic into an action-based branching-time temporal logic with a semantics that is specifically well suited for capturing the aforementioned advanced variability constraints.

4.1 HML+UNTIL

HML+UNTIL extends the classical Hennessy–Milner logic with Until by incorporating existential and universal state operators (quantifying over paths) from CTL [5]. As such, HML+UNTIL is derived from the logics defined in [18,25,26].[3]

HML+UNTIL is a logic of state formulae (denoted by ϕ) and path formulae (denoted by π) defined over a set of atomic actions $A = \{a, b, \ldots\}$.

Definition 7. *The syntax of HML+UNTIL is:*

$$\phi ::= \mathit{true} \mid \neg\, \phi \mid \phi \wedge \phi' \mid \langle a \rangle\, \phi \mid [a]\, \phi \mid E\, \pi \mid A\, \pi$$
$$\pi ::= \phi\ U\ \phi'$$
□

While formally interpreted over MTSs, the semantics of HML+UNTIL does not actually consider the different type of transitions typical of MTSs. In fact, the informal meaning of the nonstandard operators of HML+UNTIL is the following.

– $\langle a \rangle\, \phi$: a next state exists, reachable by a *may* transition executing action a, in which ϕ holds
– $[a]\, \phi$: in all next states, reachable by a *may* transition executing a, ϕ holds
– $E\, \pi$: there exists a full path on which π holds
– $A\, \pi$: on all possible full paths, π holds
– $\phi\ U\ \phi'$: in a state of a path, ϕ' holds, whereas ϕ holds in all preceding states

The HML+UNTIL semantics is thus interpreted over MTSs as if they were LTSs.

[3] These logics use recursion defined by fixed points to extend HML into a temporal logic; we prefer to directly include the Until operator.

Definition 8. *Let $(Q, A, \overline{q}, \delta^{\Box}, \delta^{\Diamond})$ be an MTS, with $q \in Q$ and σ a full path. The satisfaction relation $\models$ of HML+UNTIL over MTSs is defined as follows:*

- $q \models true$ *always holds*
- $q \models \neg\, \phi$ *iff not* $q \models \phi$
- $q \models \phi \wedge \phi'$ *iff* $q \models \phi$ *and* $q \models \phi'$
- $q \models \langle a \rangle\, \phi$ *iff* $\exists\, q' \in Q$ *such that* $q \xrightarrow{a}_{\Diamond} q'$, *and* $q' \models \phi$
- $q \models [a]\, \phi$ *iff* $\forall\, q' \in Q$ *such that* $q \xrightarrow{a}_{\Diamond} q'$, *we have* $q' \models \phi$
- $q \models E\, \pi$ *iff* $\exists\, \sigma' \in path(q)$ *such that* $\sigma' \models \pi$
- $q \models A\, \pi$ *iff* $\forall\, \sigma' \in path(q)$ *such that* $\sigma' \models \pi$
- $\sigma \models \phi\ U\ \phi'$ *iff* $\exists\, j \geq 1 \colon \sigma(j) \models \phi'$ *and* $\forall\, 1 \leq i < j \colon \sigma(i) \models \phi$ □

A number of further operators can now be derived in the usual way: *false* abbreviates $\neg\, true$, $\phi \vee \phi'$ abbreviates $\neg(\neg\phi \wedge \neg\phi')$, $\phi \Longrightarrow \phi'$ abbreviates $\neg\phi \vee \phi'$. Moreover, $F\, \phi$ abbreviates $(true\ U\, \phi)$: there exists a future state in which ϕ holds. Finally, $AG\, \phi$ abbreviates $\neg EF\, \neg\phi$: in every state on every path, ϕ holds.

4.2 Variability and Action-based CTL: vaCTL

We now introduce the variability and action-based logic vaCTL. It extends HML+UNTIL by implicitly incorporating two of the most classical *deontic* [2] modalities, namely O (*it is obligatory that*) and P (*it is permitted that*), as well as an action-based Until operator, both with and without a deontic interpretation.

vaCTL defines state formulae (denoted by ϕ) and path formulae (denoted by π), but also action formulae (boolean compositions of actions, denoted by φ, with the usual semantics, taken from [11]) over a set of atomic actions $A = \{a, b, \ldots\}$.

Definition 9. *The syntax of vaCTL is:*

$$\phi ::= true \mid \neg\, \phi \mid \phi \wedge \phi' \mid \langle a \rangle\, \phi \mid [a]\, \phi \mid \langle a \rangle^{\Box}\, \phi \mid [a]^{\Box}\, \phi \mid E\, \pi \mid A\, \pi$$
$$\pi ::= \phi\ \{\varphi\}\, U\, \{\varphi'\}\ \phi' \mid \phi\ \{\varphi\}\, U^{\Box}\, \{\varphi'\}\ \phi'$$ □

The informal meaning of the operators we added to HML+UNTIL is as follows.[4]

- $\langle a \rangle^{\Box}\, \phi$: a next state exists, reachable by a *must* transition executing a, in which ϕ holds
- $[a]^{\Box}\, \phi$: in all next states, reachable by a *must* transition executing a, ϕ holds
- $\phi\ \{\varphi\}\, U\, \{\varphi'\}\ \phi'$: in a state of a path reached by an action satisfying φ', ϕ' holds, whereas ϕ holds in all preceding states and all actions executed meanwhile along the path satisfy φ
- $\phi\ \{\varphi\}\, U^{\Box}\, \{\varphi'\}\ \phi'$: in a state of a path reached by an action satisfying φ', ϕ' holds, whereas ϕ holds in all preceding states and the path leading to that state is a *must* path along which all actions executed meanwhile satisfy φ

Also the formal semantics of vaCTL is given over MTSs, but in a deontic way by taking into account an MTS's different type of transitions.

[4] The operators $\langle a \rangle^{\Box}$ and $[a]^{\Box}$ represent the classical deontic modalities O and P, resp.

Definition 10. *Let $(Q, A, \overline{q}, \delta^{\Box}, \delta^{\Diamond})$ be an MTS, with $q \in Q$ and σ a full path. Then the satisfaction relation $\models$ of* vACTL *over MTSs is defined as follows:*

- $q \models true$ *always holds*
- $q \models \neg\,\phi$ *iff not* $q \models \phi$
- $q \models \phi \wedge \phi'$ *iff* $q \models \phi$ *and* $q \models \phi'$
- $q \models \langle a \rangle\,\phi$ *iff* $\exists\, q' \in Q$ *such that* $q \xrightarrow{a}_{\Diamond} q'$, *and* $q' \models \phi$
- $q \models [a]\,\phi$ *iff* $\forall\, q' \in Q$ *such that* $q \xrightarrow{a}_{\Diamond} q'$, *we have* $q' \models \phi$
- $q \models \langle a \rangle^{\Box}\,\phi$ *iff* $\exists\, q' \in Q$ *such that* $q \xrightarrow{a}_{\Box} q'$, *and* $q' \models \phi$
- $q \models [a]^{\Box}\,\phi$ *iff* $\forall\, q' \in Q$ *such that* $q \xrightarrow{a}_{\Box} q'$, *we have* $q' \models \phi$
- $q \models E\,\pi$ *iff* $\exists\, \sigma' \in path(q)$ *such that* $\sigma' \models \pi$
- $q \models A\,\pi$ *iff* $\forall\, \sigma' \in path(q)$ *such that* $\sigma' \models \pi$
- $\sigma \models \phi\ \{\varphi\}\,U\,\{\varphi'\}\ \phi'$ *iff* $\exists\, j \geq 1 \colon \sigma(j) \models \phi'$, $\sigma\{j\} \models \varphi'$, *and* $\sigma(j+1) \models \phi'$, *and* $\forall 1 \leq i < j \colon \sigma(i) \models \phi$ *and* $\sigma\{i\} \models \varphi$
- $\sigma \models \phi\ \{\varphi\}\,U^{\Box}\,\{\varphi'\}\ \phi'$ *iff* σ *is a must path* $\sigma^{\Box}$ *and* $\sigma^{\Box} \models \phi\ \{\varphi\}\,U\,\{\varphi'\}\ \phi'$ $\Box$

Again, further operators can be derived in the usual way. $F\phi$ abbreviates $(true\ \{true\}\,U\,\{true\}\ \phi)$: there exists a future state in which ϕ holds; $AG\,\phi$ abbreviates $\neg EF\,\neg\phi$: in all states on all paths, ϕ holds. $F\,\{\varphi\}\ true$ abbreviates $true\ \{true\}\,U\,\{\varphi\}\ true$: there exists a future state reached by an action satisfying φ; $F^{\Box}\{\varphi\}\,true$ abbreviates $true\,\{true\}\,U^{\Box}\,\{\varphi\}\,true$: there exists a future state of a must path reached by an action satisfying φ; $F^{\Box}\phi$ abbreviates $true\,\{true\}\,U^{\Box}\,\{true\}\,\phi$: there exists a future state of a must path in which ϕ holds; $AG^{\Box}\,\phi$ abbreviates $\neg EF^{\Box}\,\neg\phi$: in all states on all must paths, ϕ holds.

5 Advanced Variability Management

vACTL can complement the behavioural description of an MTS by expressing the constraints over possible products of a family that an MTS cannot model, i.e. regarding *alternative* features and *requires* and *excludes* inter-feature relations.

We formalise these three types of constraints as follows as vACTL templates.

Template ALT Features F1 and F2 are *alternative*:

$$(EF^{\Box}\,\{F1\}\,true \vee EF^{\Box}\,\{F2\}\,true) \wedge \neg(EF\,\{F1\}\,true \wedge EF\,\{F2\}\,true)$$

Template EXC Feature F1 *excludes* feature F2:

$$((EF\,\{F1\}\,true) \Longrightarrow (AG\,\neg\langle F2\rangle\,true)) \wedge ((EF\,\{F2\}\,true) \Longrightarrow (AG\,\neg\langle F1\rangle\,true))$$

Both these vACTL templates combine constraints represented by a deontic interpretation with behavioural relations among actions expressed by a temporal part.

Template REQ Feature F1 *requires* feature F2:

$$(EF\,\{F1\}\,true) \implies (EF^{\Box}\,\{F2\}\,true)$$

Note that this vACTL template does not imply any ordering among the related features: a product allowing F1 to be performed before F2 cannot be excluded as member of the product family on the basis of this formula (expressing a static relation among features). It is the duty of the behavioural LTS (MTS) description of a product (family) to impose orderings, which can consequently be verified by vACTL formulae such as the ones we present in the next section.

6 Model Checking a Family of Services

In [3], we defined a global model-checking algorithm for a deontic extension of HML+UNTIL by extending classical algorithms for HML and (A)CTL [5, 6, 11, 34]. Recently, we actually implemented an on-the-fly model-checking algorithm of linear complexity for VACTL as a particularization of the FMC model checker [31] for ACTL [11] over networks of automata, specified in a CCS-like language [17].

The MTS of Fig. 1 was automatically generated by FMC from the following CCS-like specification of the Travel Agency product family described in Sect. 2:

```
TravAgFam = must(login).Menu

Menu = must(trainreserve).TrainRes + must(flightreserve).FlightRes
     + must(hotelreserve).HotelRes + may(tourreserve).TourRes

TrainRes = must(datainput).must(timetableserv).must(result).
           must(showquotes).(must(choose).Pay + must(reject).Menu)

FlightRes = must(datainput).
          ( may(flightAserv).must(result).must(flightBserv).must(result).
            must(flightCserv).must(result).must(showbestquotes).
            (must(choose).Pay + must(reject).Menu)
          + may(defaultflightserv).must(result).must(showquotes).
            (must(choose).Pay + must(reject).Menu) )

HotelRes = must(datainput).
         ( may(hotelAserv).must(result).must(hotelBserv).must(result).
           must(hotelCserv).must(result).must(showbestquotes).
           (must(choose).Pay + must(reject).Menu)
         + may(defaulthotelserv).must(result).must(showquotes).
           (must(choose).Pay + must(reject).Menu) )

TourRes = must(possibletourserv).must(result).must(showquotes).
          (must(choose).HotelRes + must(reject).Menu)

Pay = must(payserv).(must(paymentOK).TravAgFam + must(denied).Menu)
```

In this TravAgFam specification of the Travel Agency family we actually use typed actions to implement the distinction between may and must transitions.

The above specification is extracted from the requirements of Sect. 2, based on the following assumptions on constraints and behaviour:

1. Only service orchestration is modelled, ignoring data exchange (for instance, a cart maintaining unpaid reservations could be used);
2. Services are invoked by a simple request-response interaction interface: requests are modelled by actions suffixed with 'serv', responses by correlated actions like 'result', 'paymentOK', and 'denied' actions;
3. All remaining actions model interaction with the client;

4. The alternative of contacting multiple reservation services is modelled by a sequential invocation of three services. A parallel invocation would be more realistic: while our framework allows this, it would only make our example more complex with no added information concerning family-related aspects.

The rest of this section illustrates the use of FMC. The property "A travel agency service always provides a selection of best quotes" can be formalised in vACTL as

$$AF\ \langle must(showbestquotes)\rangle\ true$$

This property clearly does not hold for TravAgFam, as contacting multiple services for best quotes is an alternative. Moreover, it is only available for flight and hotel reservations. Indeed, upon verifying this property FMC produces the result shown in the screenshot in Fig 2: the formula is false and a path through the MTS is given as counterexample (a successfully completed train reservation).

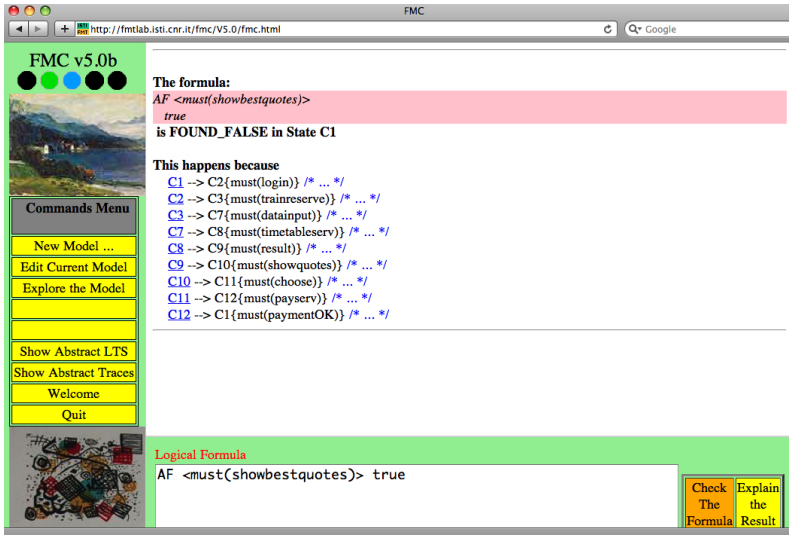

Fig. 2. FMC screenshot of result of model checking $AF\ \langle must(showbestquotes)\rangle\ true$.

Since this alternative is not available for train reservations, which is a service that is mandatory, the above formula is also false for any product of the family. If we were to change it into the following existential path formula (expressing the property "A travel agency service may provide a selection of best quotes")

$$EF\ \langle must(showbestquotes)\rangle\ true$$

then this formula would hold for the family. However, due to the distinction between may and must transitions in MTSs and in Def. 6 in particular, not

all properties (expressed as vACTL formulae) that hold for a product family (modelled as an MTS) continue to hold for all its products (modelled as LTSs).

We can experiment also this in FMC. Consider for instance the product that is obtained by removing the alternative of contacting multiple services from flight and hotel reservations. Recall from Def. 6 that a product can be obtained from a family by preserving at least all must transitions and turning some of the may transitions that are not must transitions into must transitions as well as removing all of the remaining may transitions. Doing so, we obtain the product shown in the FMC screenshot in Fig. 3. If we now verify the latter formula, FMC produces the result shown in the screenshot in Fig 4: the formula is false as there is no path through the MTS on which *must*(*showbestquotes*) is executed.

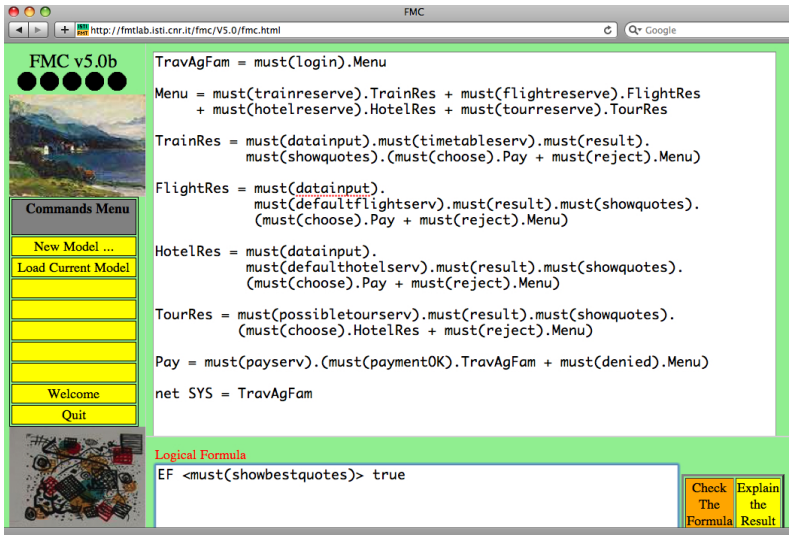

Fig. 3. FMC screenshot of a product obtained from TravAgFam.

Now consider the property "A travel agency service must always provide the possibility to reject a proposal (of quotes)". In vACTL this can be formalised as

$$AG\ [must(result)]\ AF^{\Box}\ \{must(reject)\}\ true$$

FMC can show it holds for the family. Moreover, universal formulae of this form dictate the existence of a must path with certain characteristics, which by Def. 6 are necessarily found in all products. This property thus holds for all products.

Finally, consider the variability constraint that contacting multiple reservation services and contacting a single reservation service are alternative features. This can be formalised in vACTL by instantiating the Template ALT of Sect. 5:

$$(EF^{\Box}\ \{may(hotelAserv)\}\ true \vee EF^{\Box}\ \{may(defaulthotelserv)\}\ true) \wedge \\ \neg(EF\ \{may(hotelAserv)\}\ true \wedge EF\ \{may(defaulthotelserv)\}\ true)$$

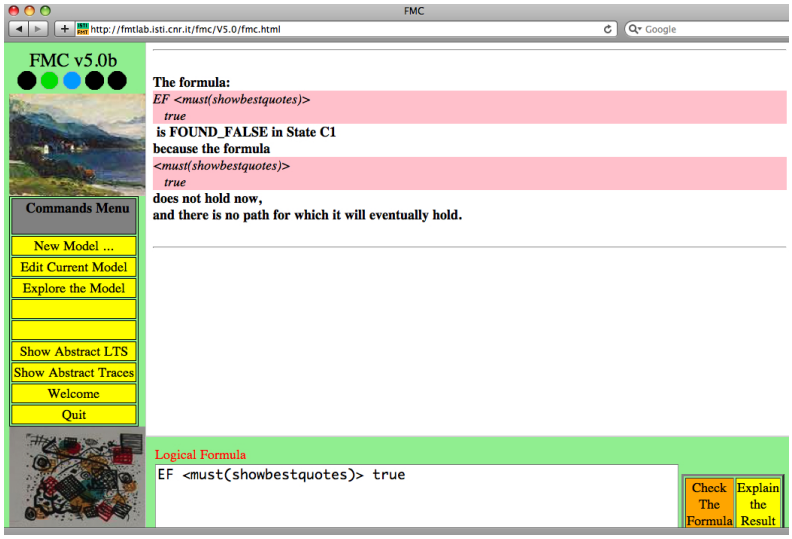

Fig. 4. FMC screenshot of result of model checking $EF\ \langle must(showbestquotes)\rangle\ true$.

As expected, FMC can show this formula is false for the family, but true for the product shown in the FMC screenshot in Fig. 3 (i.e., this product satisfies the above alternative constraint and might hence be a valid product of the family).

FMC can thus be used to experiment with products, using vACTL to guide the derivation of valid products that satisfy the advanced variability constraints by construction. As said before, we recently adapted the algorithm designed in [3] to vACTL, thus allowing us to automatically derive, from an MTS description of a product family *and* an associated set of vACTL formulae expressing further constraints for this family, *all* valid products (a set of LTS descriptions of products, each correct w.r.t. all vACTL constraints). The algorithm's complexity is bounded by $O(n \times m \times w)$, with n the number of transitions of the MTS (family), m the number of LTSs (products) returned, and w the maximum width of may transitions with labels actually referred to in the vACTL formulae (constraints). The actual implementation of this algorithm is ongoing work.

7 Related Work

In [3,12,14,27,29], MTS variants are used to model and analyse product families. Part of our recent work was described previously (cf. also [3] and its references). In [12], we extended MTSs to model advanced variability constraints regarding alternative features. In [27], modal I/O automata were defined as one of the first attempts at behavioural modelling in SPLE. In [19,20], an algebraic approach to behavioural modelling and analysis of product families was developed. In [32], Feature Petri Nets were defined to model the behaviour of product families with a high degree of variability. We discuss some approaches close to ours in detail.

In [14], the authors present an algorithm for checking conformance of LTSs against MTSs according to a given branching relation, i.e. checking conformance of the behaviour of a product against that of its product family. It is a fixed-point algorithm that starts with the Cartesian product of the states and iteratively eliminates pairs that are invalid according to the given relation. The algorithm is implemented in a tool that allows one to check whether or not a given LTS conforms to a given MTS according to a number of different branching relations.

In [29], variable I/O automata are introduced to model product families, together with a model-checking approach to verify conformance of products w.r.t. a family's variability. This is achieved by using variability information in the model-checking algorithm (while exploring the state space an associated variability model is consulted continuously). Properties expressed in CTL [5] are verified by explicit-state model checking, progressing one state at a time.

In [7], an explicit-state model-checking technique to verify linear-time temporal properties over Featured Transition Systems (FTSs) is defined. This results in a means to check that whenever a behavioural property is satisfied by an FTS modelling a product family, then it is also satisfied by every product of that family, and whenever a property is violated, then not only a counterexample is provided but also the products violating the property. In [8], this approach is improved by using symbolic model checking, examining sets of states at a time, and a feature-oriented version of CTL.

In business process modelling, configurable process models were introduced to capture a family of related business process models in a single artifact. Inspired by methods from SPLE [4, 21, 22, 30], alternatives are defined as variation points (cf. [1, 37, 38] and their references). Tool support allows the selection of valid configurations. Such correctness is related only to the static definition of a family. Our research goes beyond this static view: we adopt a specific logic interpreted over MTSs and use it to model check behavioural aspects of families of services.

8 Conclusions and Future Work

We addressed model checking families of services, starting from the addition of variability to a simple action-based branching-time temporal logic interpreted over a basic form of variable transition systems. Services are traditionally modelled with richer transition systems, which need to be addressed in future work. In particular, FMC is closely related to the CMC model checker for SocL (a service-oriented logic) formulae over the process algebra COWS (Calculus for Orchestration of Web Services) [13]. Adding variability management to this tool will allow us to handle families of services that use more complex mechanisms typical of SOC, like asynchronous communication, compensation and correlation.

Acknowledgments The research reported in this paper was partially funded by two Italian projects: D-ASAP (MIUR–PRIN 2007) and XXL (CNR–RSTL).

We thank F. Mazzanti for stimulating discussions on the vaCTL logic and for his help in using FMC for model checking vaCTL formulae over MTSs.

References

1. van der Aalst, W.M.P., Dumas, M., Gottschalk, F., ter Hofstede, A.H.M., La Rosa, M., Mendling, J.: Preserving correctness during business process model configuration. Formal Asp. Comput. 22 (3-4), 459–482 (2010)
2. Åqvist, L.: Deontic Logic. In: Gabbay, D., Guenthner, F. (eds.) Handbook of Philosophical Logic, 2nd edition, vol. 8, pp. 147–264. Kluwer, Dordrecht (2002)
3. Asirelli, P., ter Beek, M.H., Fantechi, A., Gnesi, S.: A Logical Framework to Deal with Variability. In: Méry, D., Merz, S. (eds.) IFM 2010. LNCS, vol. 6396, pp. 43–58. Springer, Berlin (2010)
4. Bertolino, A., Fantechi, A., Gnesi, S., Lami, G., Maccari, A.: Use Case Description of Requirements for Product Lines. In [16], pp. 12–18 (2002)
5. Clarke, E.M., Emerson, E.A., Sistla, A.P.: Automatic Verification of Finite State Concurrent Systems using Temporal Logic Specifications. ACM Trans. Program. Lang. Syst. 8 (2), 244–263 (1986)
6. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT, Cambridge (1999)
7. Classen, A., Heymans, P., Schobbens, P.-Y., Legay, A., Raskin, J.-F.: Model Checking Lots of Systems: Efficient Verification of Temporal Properties in Software Product Lines. In: Proceedings 32nd International Conference on Software Engineering (ICSE 2010), pp. 335–344. ACM Press, New York (2010)
8. Classen, A., Heymans, P., Schobbens, P.-Y., Legay, A.: Symbolic Model Checking of Software Product Lines. To appear in: Proceedings 33rd International Conference on Software Engineering (ICSE 2011). ACM Press, New York (2011)
9. Clements, P.C., Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley, Reading (2002)
10. Cohen, S.G., Krut, R.W. (eds.): Proceedings 1st Workshop on Service-Oriented Architectures and Software Product Lines: What is the Connection? (SOAPL 2007). Technical Report CMU/SEI-2008-SR-006, Carnegie Mellon University (2008)
11. De Nicola, R., Vaandrager, F.W.: Three Logics for Branching Bisimulation. J. ACM 42 (2), 458–487 (1995)
12. Fantechi, A., Gnesi, S.: Formal Modelling for Product Families Engineering. In [15], pp. 193–202 (2008)
13. Fantechi, A., Lapadula, A., Pugliese, R., Tiezzi, F., Gnesi, S., Mazzanti, F.: A Logical Verification Methodology for Service-Oriented Computing. To appear in *ACM Trans. Softw. Eng. Methodol.* (2011)
14. Fischbein, D., Uchitel, S., Braberman, V.A.: A Foundation for Behavioural Conformance in Software Product Line Architectures. In: Hierons, R.M., Muccini, H. (eds.) Proceedings ISSTA 2006 Workshop on Role of Software Architecture for Testing and Analysis (ROSATEA 2006), pp. 39–48. ACM Press, New York (2006)
15. Geppert, B., Pohl, K. (eds.): Proceedings 12th Software Product Lines Conference (SPLC 2008). IEEE Press, Los Alamitos (2008)
16. Geppert, B., Schmid, K. (eds.): Proceedings International Workshop on Requirements Engineering for Product Lines (REPL 2002). Technical Report ALR-2002-033, Avaya Labs Research (2002)
17. Gnesi, S., Mazzanti, F.: On the Fly Verification of Networks of Automata. In: Arabnia, H.R. et al. (eds.) Proceedings International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 1999), pp. 1040–1046. CSREA Press, Athens (2009)
18. Godefroid, P., Huth, M., Jagadeesan, R.: Abstraction-Based Model Checking Using Modal Transition Systems. In: Larsen, K.G., Nielsen, M. (eds.) CONCUR 2001. LNCS, vol. 2154, pp. 426–440. Springer, Berlin (2001)

19. Gruler, A., Leucker, M., Scheidemann, K.D.: Modelling and Model Checking Software Product Lines. In: Barthe, G., de Boer, F.S. (eds.) FMOODS 2008. LNCS, vol. 5051, pp. 113–131. Springer, Berlin (2008)
20. Gruler, A., Leucker, M., Scheidemann, K.D.: Calculating and Modelling Common Parts of Software Product Lines. In [15], pp. 203–212 (2008)
21. Halmans, G., Pohl, K.: Communicating the Variability of a Software-Product Family to Customers. Software and System Modeling 2 (1), 15–36 (2003)
22. John, I, Muthig, D.: Tailoring Use Cases for Product Line Modeling. In [16], pp. 26–32 (2002)
23. Krut, R.W., Cohen, S.G. (eds.): Proceedings 2nd Workshop on Service-Oriented Architectures and Software Product Lines: Putting Both Together (SOAPL 2008). In: Thiel, S., Pohl, K. (eds.) Workshop Proceedings 12th Software Product Lines Conference (SPLC 2008), pp. 115–147. Lero Centre, University of Limerick (2008).
24. Krut, R.W., Cohen, S.G. (eds.): Proceedings 3rd Workshop on Service-Oriented Architectures and Software Product Lines: Enhancing Variation (SOAPL 2009). In: Muthig, D., McGregor, J.D. (eds.) Proceedings 13th Software Product Lines Conference (SPLC 2009), pp. 301–302. ACM Press, New York (2009)
25. Larsen, K.G.: Modal Specifications. In: Sifakis, J. (ed.) Automatic Verification Methods for Finite State Systems. LNCS, vol. 407, pp. 232–246. Springer, Berlin (1989)
26. Larsen, K.G.: Proof Systems for Satisfiability in Hennessy-Milner Logic with Recursion. Theor. Comput. Sci. 72 (2–3), 265–288 (1990)
27. Larsen, K.G., Nyman, U., Wąsowski, A.: Modal I/O Automata for Interface and Product Line Theories. In: De Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 64–79. Springer, Berlin (2007)
28. Larsen, K.G., Thomsen, B.: A Modal Process Logic. In: Proceedings 3rd Annual Symposium on Logic in Computer Science (LICS 1988), pp. 203–210. IEEE Press, Los Alamitos (1988)
29. Lauenroth, K., Pohl, K., Töhning, S.: Model Checking of Domain Artifacts in Product Line Engineering. In: Proceedings 24th International Conference on Automated Software Engineering (ASE 2009), pp. 269–280. IEEE Press (2009)
30. von der Maßen, T., Lichter, H.: Modeling Variability by UML Use Case Diagrams. In [16], pp. 19–25 (2002)
31. Mazzanti, F.: FMC v5.0b, `http://fmt.isti.cnr.it/fmc` (2011)
32. Muschevici, R., Clarke, D., Proenca, J.: Feature Petri Nets. In: Schaefer, I., Carbon, R. (eds.) Proceedings 1st International Workshop on Formal Methods in Software Product Line Engineering (FMSPLE 2010). University of Lancaster (2010)
33. Meyer, M.H., Lehnerd, A.P.: The Power of Product Platforms: Building Value and Cost Leadership. The Free Press, New York (1997)
34. Müller–Olm, M., Schmidt, D.A., Steffen, B.: Model-Checking: A Tutorial Introduction. In: Cortesi, A., Filé, G. (eds.) SAS 1999. LNCS, vol. 1694, pp. 330–354. Springer, Berlin (1999)
35. Papazoglou, M., Traverso, P., Dustdar, S., Leymann, F.: Service-oriented computing: State of the art and research challenges. IEEE Comput. 40 (11), 38–45 (2007)
36. Pohl, K., Böckle, G., van der Linden, F.: Software Product Line Engineering: Foundations, Principles, and Techniques. Springer, Berlin (2005)
37. Razavian, M., Khosravi, R.: Modeling Variability in Business Process Models Using UML. In: Proceedings 5th International Conference on Information Technology: New Generations (ITNG 2008), pp. 82–87. IEEE Press, Los Alamitos (2008)
38. Rosemann, M., van der Aalst, W.M.P.: A configurable reference modelling language. Inf. Syst. 32 (1), 1–23 (2007)