From Featured Transition Systems to Modal Transition Systems with Variability Constraints

From Featured Transition Systems to Modal Transition Systems with Variability Constraints***

Maurice H. ter Beek[1], Ferruccio Damiani[2], Stefania Gnesi[1], Franco Mazzanti[1], and Luca Paolini[2]

[1] ISTI–CNR, Via G. Moruzzi 1, 56124 Pisa, Italy
{terbeek,gnesi,mazzanti}@isti.cnr.it
[2] Università di Torino, C.so Svizzera 185, 10149 Torino, Italy
{damiani,paolini}@di.unito.it

Abstract. We present an automatic technique to transform a subclass of featured transition systems into modal transition systems with additional sets of variability constraints in the specific format accepted by the variability model checker VMC. Both formal models are widely used in the field of software product line engineering and both come with a dedicated model checker. The transformation serves two purposes. First, it contributes to a better understanding of the fundamental differences between the two approaches, basically concerning the way in which variability constraints are represented (in terms of features and actions, respectively). Second, it paves the way to compare the modelling and analysis of product line behaviour in two different settings.

1 Introduction

Modern software systems come in many variants in order to satisfy multiple varying user requirements [24]. Such variant-rich, configurable systems are developed and managed by techniques from the field known as software product line engineering (SPLE) [23]. Feature-oriented software development (FOSD) [1] is currently one of the most widely used approaches for modelling variability. A

* We received support by project HyVar (which has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 644298), EU FP7-ICT FET-Proactive project QUANTICOL (600708), Italian MIUR project CINA (PRIN 2010LHT4KM), Ateneo/CSP SALT project, ICT COST Action IC1402 ARVI, and ICT COST Action IC1201 BETTY.

** **This is the authors' version of the paper:** Maurice H. ter Beek, Ferruccio Damiani, Stefania Gnesi, Franco Mazzanti, Luca Paolini. From Featured Transition Systems to Modal Transition Systems with Variability Constraints. In R. Calinescu and B. Rumpe (Eds.): SEFM 2015, LNCS 9276, pp. 344-359, 2015. (Proceeding of 13th International Conference on Software Engineering and Formal Methods, SEFM 2015, York, UK, September 7-11, 2015.) DOI: 10.1007/978-3-319-22969-0_24 **The final publication is available at Springer via http://dx.doi.org/10.1007/978-3-319-22969-0_24**

feature characterises a stakeholder visible piece of functionality or aspect of a system and a *feature diagram* models all possible products of a configurable system (e.g. a software product line) in a compact way in terms of their features [25].

Basically, a feature diagram is a hierarchical tree structure of features that defines their presence in products (thus defining the valid product configurations): *optional* features may be present provided their parent is, *mandatory* features must be present provided their parent is, exactly one of the features involved in an *alternative* relation must be present provided their parent is, and at least one of the features involved in an *or* relation must be present provided their parent is. Additional cross-tree constraints may be used to indicate that the presence of one feature *requires* that of another or *excludes* the presence of another feature (i.e. they are mutually exclusive).

Featured transition systems (FTS) were introduced in [14] as a semantic model for the concise description of the behaviour of variability-intensive systems. An FTS is a doubly-labelled transition system (L^2TS) with an additional feature diagram. Each state is labelled with an atomic proposition while each transition is labelled with an action and, using the improved definition from [13], an associated *feature expression* (a Boolean formula defined over the set of features) that needs to hold for this specific transition to be part of the executable product behaviour. Hence an FTS models a family of labelled transition systems (LTS), one per product, which can be obtained by projection.

Modal transition systems (MTS) were originally introduced in [21] to model successive refinements (implementations) of partial specifications. They were first proposed for the compact description of all possible operational behaviour of the products of a product line in [18] and form the basis of numerous successive approaches in SPLE [2, 3, 17, 20, 22]. An MTS is an LTS that distinguishes between *admissible* (may) and *necessary* (must) transitions. In this paper, we use a specific variant that will be introduced in Sect. 3.

Variants of FTS and MTS are widely used in SPLE and they come with dedicated model checkers. FTS model checkers like SNIP [12], now integrated in the product line of model checkers ProVeLines [15], allow efficient *family-based* SPL model checking capable of relating errors and undesired behaviour to the exact set of products in which they occur. Such verification techniques operate on an entire product line using variability knowledge about valid feature configurations to deduce results for products, as opposed to enumerative *product-based* verification in which individually generated products (or at most a subset) are examined [26]. The MTS-based variability model checker VMC (`fmt.isti.cnr.it/vmc`) [9, 10] combines elements of both analysis strategies.

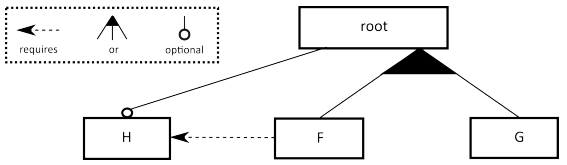
There is an obvious trade-off between brute-force product-based analysis with highly optimised model checkers for single product engineering, like SPIN (`spinroot.com`), NuSMV (`nusmv.fbk.eu`) and mCRL2 (`www.mcrl2.org`), and dedicated family-based analysis with SPL model checkers, like SNIP [12] and the NuSMV extension of [11]. One of the goals of this paper is to set the stage for a full-fledged comparison between SNIP and VMC.

In this paper, we present an automatic technique to transform FTS[3] into MTS (with additional sets of variability constraints and in the specific format accepted by VMC). The transformation serves two purposes. First, it contributes to a better understanding of the fundamental differences between the two models, basically concerning the way in which variability constraints are represented (in terms of features and actions, respectively). Second, it paves the way to compare the modelling and analysis of product line behaviour in two different frameworks.

The paper starts with a running example in Sect. 2. In Sect. 3 we provide the necessary background on MTS, after which we point out the differences with respect to FTS in the way each deals with variability (constraints) in Sect. 4. The main contribution of this paper, the transformation from FTS to MTS, is defined in Sect. 5. Some model-checking features of VMC are presented in Sect. 6. In Sect. 7, the transformation is performed on an FTS from the literature, after which VMC is applied to the result. Conclusions and future work close the paper.

2 Running Example

We illustrate our transformation technique on a small running example (we will present a larger example from the SPL literature in Sect. 7). We assume a product line with three features (`F`, `G`, and `H`) and the feature diagram depicted in Fig. 1, which defines the four valid product configurations depicted alongside.



$p_1 = \{\mathtt{F}, \mathtt{G}, \mathtt{H}\}$
$p_2 = \{\mathtt{F}, \mathtt{H}\}$
$p_3 = \{\mathtt{G}, \mathtt{H}\}$
$p_4 = \{\mathtt{G}\}$

Fig. 1. Feature diagram of running example

The allowed behaviour of the four products is modelled by the FTS in Fig. 2. Formally, an FTS is a transition system with an associated feature diagram and a labelling function that labels the transitions with an action and an additional feature expression (i.e. a Boolean expression over the features). For instance, the transition $1 \xrightarrow{a/\mathtt{F} \wedge \mathtt{G}} 2$ means that a only occurs in products having both features `F` and `G` (i.e. in p_1). We moreover require any action occurring more than once in an FTS to be tagged with one and the same feature expression. Note that this can easily be achieved by renaming or indexing possible multiple occurrences. The specific behaviour of each of the products is modelled by the LTS in Fig. 2.

[3] We consider a subclass of *action-based* FTS in which we ignore their state labels (atomic propositions) and consider only their transition labels (actions).

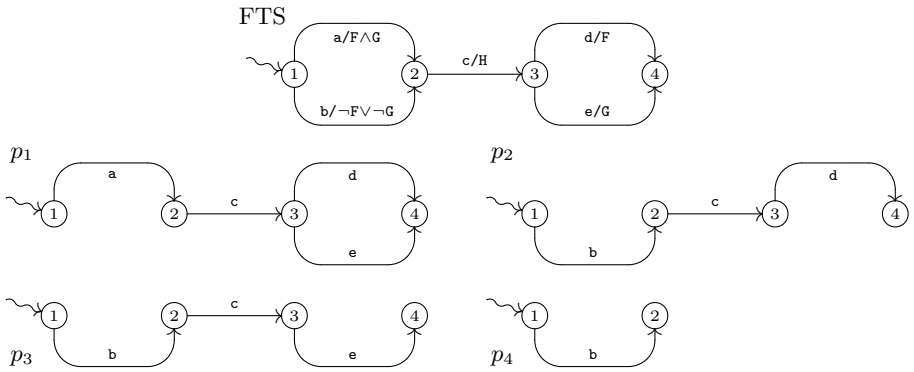


Fig. 2. FTS of running example and LTS of product configurations p_1, p_2, p_3, and p_4

3 Modal Transition Systems with Variability Constraints

We assume some familiarity with the principles of labelled transition systems (LTS), model checking and action-based computation tree logic (ACTL) [4,6,16].

Recall that an MTS is an LTS that distinguishes *admissible* (may) from *necessary* (must) transitions. By definition, every necessary transition is also an admissible transition, while admissible but not necessary transitions are called *optional*. Graphically, solid edges model necessary transitions while dotted edges model optional transitions. Here we focus on the elaboration of MTS into a modelling and analysis framework for the specification and verification of behavioural variability in SPLE in [2,3,5]. This concerns a different semantics for refinement of MTS into LTS (implementations) and the addition of an associated set of so-called variability constraints. Next we explain this in more detail and from now on we always intend this specific type of MTS when we speak of MTS. Some commonalities and differences with the FTS of [14] are discussed in [3].

Like FTS, also an MTS models a family of LTS (one per product) which can be obtained by turning each optional transition into a necessary transition or by removing it; this differs fundamentally from the classical definition of refinement [21]. An MTS has to respect the notion of *coherence* (i.e. the set of labels of the necessary transitions and that of the optional transitions must be disjoint) and the refinement operation has to respect the notion of *consistency* (i.e. the decision to turn one optional a-transition into a necessary one must be repeated for all other optional a-transitions). Moreover, an MTS does not have an associated feature diagram. Instead, it has an associated set of *variability constraints* (expressed over action labels rather than over features), which each product must satisfy. Let a range over LTS actions. Given an LTS $\mathcal{L}$ the following six different kind of variability constraints may be defined over $\mathcal{L}$ (where "occurrence of an action a in $\mathcal{L}$" is defined as "a being the label of a reachable transition in $\mathcal{L}$"):

a_1 **ALT** $\cdots$ **ALT** a_n : precisely one of the $n \geq 2$ actions $a_1, ..., a_n$ must occur in $\mathcal{L}$;

a_1 `OR` $\cdots$ `OR` a_n : at least one of the $n \geq 2$ actions $a_1, ..., a_n$ must occur in $\mathcal{L}$;
a_1 `EXC` a_2 : at most one of the actions a_1 and a_2 may occur in $\mathcal{L}$;
a_1 `REQ` a_2 : action a_2 must occur in $\mathcal{L}$ whenever a_1 occurs in $\mathcal{L}$;
a_1 `IFF` (a_2 `ALT` $\cdots$ `ALT` a_n) : precisely one of the $n \geq 2$ actions $a_2, ..., a_n$ must occur in $\mathcal{L}$ if and only if a_1 occurs in $\mathcal{L}$;
a_1 `IFF` (a_2 `OR` $\cdots$ `OR` a_n) : at least one of the $n \geq 2$ actions $a_2, ..., a_n$ must occur in $\mathcal{L}$ if and only if a_1 occurs in $\mathcal{L}$.

These constraints express exactly the standard type of relations that may be modelled by means of a feature diagram (expressed in terms of actions, though).

VMC [9,10] is a dedicated model checker for this type of MTS modelling product line variability. It accepts the specification of an MTS in process-algebraic terms together with an optional set of variability constraints, upon which it allows to perform two kinds of behavioural variability analyses (cf. Sect. 6):

1. The actual set of all valid product behaviour can explicitly be generated and the resulting LTS can all be verified against one and the same logic property (expressed in ACTL, cf. Sect. 6 for a definition).
2. A logic property (expressed in *variability-aware* ACTL, cf. Sect. 6 for a definition) can directly be verified against the MTS, relying on the fact that under certain syntactic conditions validity over the MTS guarantees validity of the same property for all its products (cf. Theorems 2 and 3 in Sect. 6).

4 From Feature Constraints to Action Constraints

We use a simple example to show the role that reachability plays when transforming an FTS (with constraints in terms of features and action labels tagged with feature expressions) into an MTS with variability constraints (expressed in terms of actions). Consider the FTS in Fig. 3 (left) and imagine that the feature diagram gives rise to the constraint `A` requires `C`. It is immediate that a product that contains the features `A` and `C` but not `B` is valid. The FTS projection for this product (obtained by first removing all transitions whose feature expression is not satisfied by $\mathtt{A} \wedge \mathtt{C} \wedge \neg \mathtt{B}$ and then all states and transitions that are no longer reachable from the initial state) results in the LTS in Fig. 3 (right).

However, it is far from trivial to obtain this LTS in Fig. 3 (right) among the valid products of an MTS with constraints on its actions, since this LTS apparently violates the obvious translation of the (feature) constraint `A REQ C` into the (action) constraint `a REQ c`, meaning that whenever action `a` occurs (i.e. is reachable) then so does action `c`. The solution we propose is to introduce:

1. a new action for each feature (which allows to handle more complex feature expressions);
2. a dummy transition for each action (which is used to verify the constraints).

The resulting MTS would be the one shown in Fig. 4 (left), where 

actually is a shorthand notation for a separate (may) transition for each action

and each feature. This MTS actually has the LTS in Fig. 4 (right) among its valid products (note that `a REQ c` is now satisfied).

It is important to underline that our transformation is such that we are able to ignore all dummy transitions when model checking. It is the combination of the presence of dummy transitions and the aforementioned notion of consistency (cf. Sect. 3), that makes this solution work. In the example, consistency guarantees that whenever a c-labelled may transition from the initial state is preserved in the LTS, then also any other reachable c-labelled may transition must be preserved.

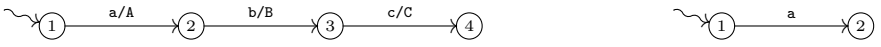


Fig. 3. FTS (left) and a valid product LTS (right)

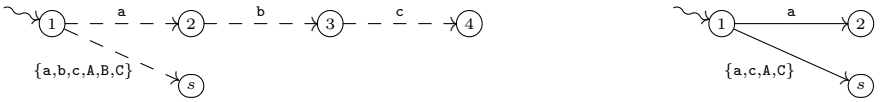


Fig. 4. MTS (left) and a valid product LTS (right)

5 Model Transformation

We assume, without loss of generality, that any action occurring more than once in an FTS is tagged with one and the same feature expression (cf. Sect. 2).

Step 1: definition of valid products in terms of features. The type of variability constraints accepted by VMC (cf. Sect. 3) and the fact that (in step 3) we will add dummy transitions labelled with actions that represent features (as anticipated in Sect. 4) allow to directly translate the feature diagram in a set of variability constraints on features. For our running example we obtain the following constraints: `F OR G` and `F REQ H`.

Step 2: definition of valid products in terms of actions. We define a logic formula of the form $\mathtt{a} \leftrightarrow \phi$ for each transition $\xrightarrow{\mathtt{a}/\phi}$ in the FTS, i.e. we link each action with its associated feature expression via a biconditional (iff). Moreover, all feature expressions not directly translatable in one of the type of variability constraints accepted by VMC (cf. Sect. 3) are transformed into conjunctive normal form (CNF). For our running example we obtain the following propositional formulae:

$$\mathtt{a} \leftrightarrow (\mathtt{F} \wedge \mathtt{G}) \equiv (\sim\mathtt{a} \vee \mathtt{F}) \wedge (\sim\mathtt{a} \vee \mathtt{G}) \wedge (\mathtt{a} \vee \sim\mathtt{F} \vee \sim\mathtt{G})$$
$$\mathtt{b} \leftrightarrow (\sim\mathtt{F} \vee \sim\mathtt{G}) \equiv (\mathtt{b} \vee \mathtt{F}) \wedge (\mathtt{b} \vee \mathtt{G}) \wedge (\sim\mathtt{b} \vee \sim\mathtt{F} \vee \sim\mathtt{G})$$
$$\mathtt{c} \leftrightarrow \mathtt{H}$$
$$\mathtt{d} \leftrightarrow \mathtt{F}$$
$$\mathtt{e} \leftrightarrow \mathtt{G}$$

To be able to accept any formula in CNF, we have slightly extended the set of variability constraints accepted by VMC. In VMC v6.1, the constraint concerning `OR`, i.e. $\texttt{a}_1$ `OR` $\cdots$ `OR` $\texttt{a}_n$, can contain either $\texttt{a}_i$ (as before) or its negation $\sim\texttt{a}_i$.

Step 3: definition of valid products in MTS and additional variability constraints. We define the FTS depicted in Fig. 2 in a process-algebraic setting, which can be seen as the natural encoding of the graph (FTS) of Fig. 2, with the process terms corresponding to the nodes of the graph and all actions 'tagged' with `may` rather than with a feature expression. Actions in the FTS without an associated feature expression are not tagged with `may`, i.e. they are considered 'must' actions.

We moreover create a dummy action for each resulting 'may' action and for each non-mandatory feature, whose executions all result in a deadlock. Finally, we create a new initial process from which the execution of a special action `behaviour` leads to the FTS encoding, whereas a special action `signature` leads to the execution of dummy actions.

In process algebra, the basic mechanism for constructing behavioural expressions is action prefixing. The process `a.P` executes `a` and subsequently behaves as process `P`. The process `P + Q` non-deterministically chooses to behave as either process `P` or process `Q`. Finally, `nil` stands for both successful termination and deadlock. We use `net SYS` to indicate the initial process of a process model. For our running example, we obtain the process-algebraic definition of an MTS with an additional set of variability constraints given in Fig. 5 (on the left-hand side).

Step 4: definition of live action sets and transformation into must transitions. We present two optimisations for model-checking purposes: the explicit definition of additional live action sets (explained in more detail in the next section) and the transformation of may transitions into must transitions. For both, we explore the behaviour process created in step 3.

1. For each subprocess `T` that can be reached from n other subprocesses by performing one of the actions $\texttt{a}_1, \ldots, \texttt{a}_n$ (possibly tagged with `may`) while from `T` itself a 'may' action `a(may)` can be executed, the latter is substituted by 'must' action `a` *whenever* $\bigwedge_{1 \leq i \leq n} (\texttt{a}_i \rightarrow \texttt{a})$ is a tautology with respect to all other constraints. Furthermore, the corresponding dummy action is eliminated together with the associated constraints.
2. For each subprocess `T` (corresponding to a node in the FTS) from which $n > 1$ 'may' actions $\texttt{a}_1\texttt{(may)}, \ldots, \texttt{a}_n\texttt{(may)}$ (and no 'must' actions) can be executed, $\texttt{a}_1 \vee \cdots \vee \texttt{a}_n$ is added to the set of variability constraints (if not already present) *whenever* it is a tautology with respect to all other constraints.

In our running example, no action can be transformed, while `a OR b` and `d OR e` are added to the set of variability constraints according to 2.

These optimisations help the model checker to understand a model's live states and to take full advantage of the specificities of variability-aware ACTL (i.e. the so-called 'boxed' operators). Both will become more clear in Sect. 6.

```
Behaviour = behaviour.T1
T1 = a(may).T2 + b(may).T2
T2 = c.T3
T3 = d(may).T4 + e(may).T4
T4 = nil

Signature = signature.(
  -- may actions
  a(may).nil + b(may).nil +
  c(may).nil + d(may).nil +
  e(may).nil +
  -- optional features
  F(may).nil + G(may).nil +
  H(may).nil
)

net SYS = Behaviour + Signature

Constraints {
  -- Directly from the feature diagram
  F OR G
  F REQ H
  -- Relating feature expressions to actions:
  -- a IFF (F AND G) in CNF:
  not a OR F
  not a OR G
  a OR not F OR not G
  -- b IFF (~F OR ~G) in CNF:
  not b OR not F OR not G
  b OR F
  b OR G
  --
  c IFF H
  d IFF F
  e IFF G
}
```

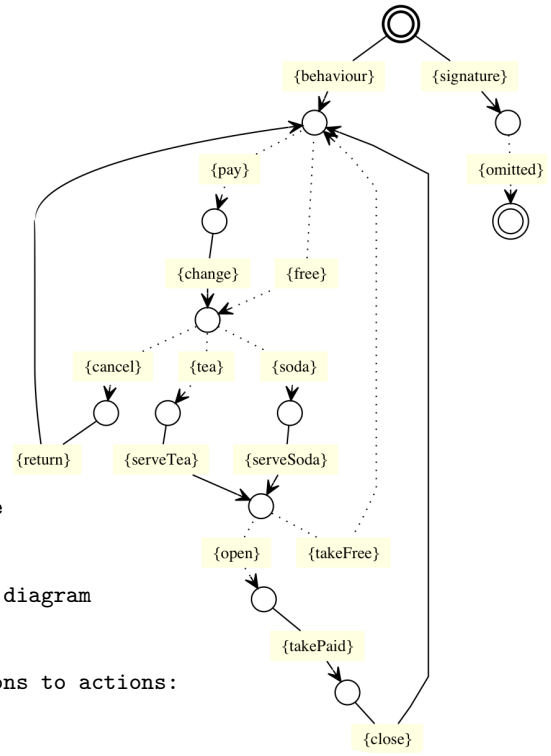


Fig. 5. VMC input model for the running example of Sect. 2 (left) and VMC generated MTS of vending machine product line of Sect. 7 (right)

Soundness of model transformation. Given an FTS S and an MTS S', let $[\![S]\!]$ denote the set of valid product configurations for S, and let $\mathrm{FTS}(S)$ and $\mathrm{MTS}(S')$ denote the set of LTS products of S and S', respectively.

Theorem 1 (Soundness of model transformation). *Let S be an FTS and S' be the MTS obtained by the model transformation procedure described above.*

1. *There exist a bijection between* $[\![S]\!]$ *and MTS(S′) such that each* $p \in [\![S]\!]$ *is associated to an LTS that contains a (dummy) transition with label* `F` *for each feature* $\texttt{F} \in p$ *and no transitions labelled with a feature not in* p.
2. *The set FTS(S) and the set of LTS obtained by omitting the dummy transitions from the LTS in MTS(S′) are equal.*

Proof (sketch). Each valid product configuration $p \in [\![S]\!]$ determines an LTS S_p, called FTS projection.

1. Let p be a valid product configuration for S. Consider the LTS P obtained by extending S_p with a transition for each action `a` in S_p (labelled by `a`) and for each selected feature $\texttt{F} \in p$ (labelled by `F`), whose executions result in a deadlock. Then, $P \in MTS(S')$ because the MTS variability constraints mimic (by construction) the feature constraints of S and from the way in which the MTS process generation is carried out. On the other hand, given an LTS $P' \in MTS(S')$ it is straightforward to recover a valid product configuration, by dummy transitions labelled by features which occur in P'.
2. Straightforward, reasoning as above. □

Patently, removing the dummy transitions in the LTS in MTS(S') may collapse some LTS. This happens exactly when the FTS S is ambiguous (i.e. there are at least two different valid product configurations that generate the same LTS).

6 Model Checking

The model transformation described in Sect. 5 allows to use VMC to verify properties over the entire product line or over its individual products alike. These properties can be specified in the action-based branching-time temporal modal logic ACTL (for products, i.e. LTS) or one of the fragments of its variability-aware extension v-ACTL (for product lines, i.e. MTS) defined next. ACTL defines action formulae (denoted by ψ), state formulae (denoted by ϕ), and path formulae (denoted by π). Action formulae are Boolean compositions of actions.

Definition 1. *Action formulae are built over a set* $\{a, b, \ldots\}$ *of atomic actions:*

$$\psi ::= \mathit{true} \mid a(e) \mid \neg\,\psi \mid \psi \wedge \psi$$

Definition 2. *The syntax of ACTL as accepted by VMC is defined as follows:*

$$\begin{aligned}\phi ::= {} & \mathit{true} \mid \neg\,\phi \mid \phi \wedge \phi \mid [\chi]\,\phi \mid \langle\chi\rangle\,\phi \mid E\,\pi \mid A\,\pi \mid \mu\,Y.\phi(Y) \mid \nu\,Y.\phi(Y)\\ \pi ::= {} & [\phi\ \{\chi\}\,U\,\{\chi'\}\ \phi'] \mid [\phi\ \{\chi\}\,U\ \phi'] \mid [\phi\ \{\chi\}\,W\,\{\chi'\}\ \phi'] \mid [\phi\ \{\chi\}\,W\ \phi'] \mid\\ & X\,\{\chi\}\ \phi \mid F\,\phi \mid F\,\{\chi\}\ \phi \mid G\,\phi\end{aligned}$$

where Y *is a propositional variable and* $\phi(Y)$ *is syntactically monotone in* Y.

In VMC, propositional operators $\neg$, $\vee$, $\wedge$, and the least and greatest fixed-point operators μ and ν are written as `not`, `or`, `and`, `min`, and `max`, respectively.

We provide some intuition for the less common (action-based) operators. The action-based until operators $[\phi\ \{\chi\}\, U\ \phi']$ ($[\phi\ \{\chi\}\, U\, \{\chi'\}\ \phi']$) say that ϕ' holds at some future state of the path (reached by a final action satisfying χ'), while ϕ holds from the current state until that state is reached and all the actions executed meanwhile along the path satisfy χ. The action-based weak until operators $[\phi\ \{\chi\}\, W\ \phi']$ and $[\phi\ \{\chi\}\, W\, \{\chi'\}\ \phi']$ (also called unless) hold on a path either if the corresponding strong until operator holds or if for all states of the path the formula ϕ holds and all actions executed on the path satisfy χ.

To make ACTL variability-aware, for the box, diamond and F operators we defined also an interpretation that takes the modality of the transitions (may or must) into account, resulting in v-ACTL. The intuitive interpretation of the different variants of these operators is as follows. $[\chi]\,\phi$: in all next states reachable by a may transition executing an action satisfying χ, ϕ holds. $[\chi]^{\Box}\,\phi$: in all next states reachable by a must transition executing an action satisfying χ, ϕ holds. $F\,\phi$: there exists a future state in which ϕ holds. $F^{\Box}\,\phi$: there exists a future state in which ϕ holds and all transitions until that state are must transitions. $F\,\{\chi\}\ \phi$: there exists a future state, reached by an action satisfying χ, in which ϕ holds. $F^{\Box}\,\{\chi\}\ \phi$: there exists a future state, reached by an action satisfying χ, in which ϕ holds and all transitions until that state are must transitions.

We now present two fragments of v-ACTL, called v-ACTL$^{\Box}$ and v-ACTLive$^{\Box}$, which suffice for the specification of many interesting properties for product lines and, moreover, enjoy some convenient properties concerning the preservation of results from MTS to LTS (elaborated on below) which allow to perform a type of family-based verification with linear complexity.[4]

Due to space limitation, we only present the syntax of these logics. We refer to [5,8,9] for their semantics (and for proofs of the preservation theorems below[5]).

Definition 3. *The syntax of the fragment v-ACTL$^{\Box}$ of v-ACTL is defined as:*

$$\phi ::= \mathit{false} \mid \mathit{true} \mid \phi \wedge \phi \mid \phi \vee \phi \mid [\chi]\,\phi \mid \langle\chi\rangle^{\Box}\,\phi \mid EF^{\Box}\,\phi \mid EF^{\Box}\,\{\chi\}\ \phi \mid AF^{\Box}\,\phi \mid AF^{\Box}\,\{\chi\}\ \phi \mid AG\,\phi \mid \neg\,\psi$$

where

$$\psi ::= \mathit{false} \mid \mathit{true} \mid \psi \wedge \psi \mid \psi \vee \psi \mid \langle\chi\rangle\,\psi \mid EF\,\psi \mid EF\{\chi\}\ \psi \mid \neg\,\phi$$

Note that v-ACTL$^{\Box}$ consists of two parts. The first part is such that any formula expressed in it that is true for the MTS, is also true for all products. The second part (which in v-ACTL$^{\Box}$ appears negated) is such that any formula expressed in it that is false for the MTS, is also false for all products.

For the sequel, let S be an MTS. A formula ϕ is said to be *preserved by refinement* if $S \models \phi$ implies $S_p \models \phi$, for all products (i.e. refinements) S_p of S.

Theorem 2 (Preservation by refinement). *Any formula ϕ expressed in v-ACTL$^{\Box}$ is preserved by refinement.*

[4] The complexity of verification with either v-ACTL$^{\Box}$ or v-ACTLive$^{\Box}$ in VMC is linear with respect to the size of the state space and with respect to the size of the formula.

[5] Actually, the results presented in Theorems 2 and 3 are slight extensions of those presented in [5,8,9] by including the neXt and Until operators not considered there.

We also define a wider fragment of v-ACTL, which again has two parts, but with a slightly different characteristic: all formulae expressed in it that are valid over a *live MTS* preserve their validity for all valid products of that MTS. An MTS is live if all its states are live. Intuitively, a *live state* of an MTS is a state that does not occur as a final state in any of its products. So-called *live action sets* are used to define such states. For instance, a state q with two outgoing transitions whose actions labels `a` and `b` are in an or relation, is a live state based on the fact that `a OR b` gives rise to a live action set $\{\texttt{a}, \texttt{b}\}$: it guarantees that in any product in which q occurs, q has at least one outgoing transition.

Definition 4. *The syntax of the fragment v-ACTLive$^\Box$ of v-ACTL is defined as:*

$$\begin{aligned}\phi ::= {} & \mathit{false} \mid \mathit{true} \mid \phi \wedge \phi \mid \phi \vee \phi \mid [\chi]\,\phi \mid \langle\chi\rangle^\Box\,\phi \mid EF^\Box\,\phi \mid EF^\Box\,\{\chi\}\ \phi \mid \\ & A[\phi\ \{\chi\}\,U\,\{\chi'\}\ \phi'] \mid A[\phi\ \{\chi\}\,U\ \phi'] \mid A[\phi\ \{\chi\}\,W\,\{\chi'\}\ \phi'] \mid A[\phi\ \{\chi\}\,W\ \phi'] \mid \\ & AX\,\{\chi\}\ \phi \mid AF\,\phi \mid AF\,\{\chi\}\ \phi \mid AF^\Box\,\phi \mid AF^\Box\,\{\chi\}\ \phi \mid AG\,\phi \mid \neg\,\psi\end{aligned}$$

where

$$\begin{aligned}\psi ::= {} & \mathit{false} \mid \mathit{true} \mid \psi \wedge \psi \mid \psi \vee \psi \mid \langle\chi\rangle\,\psi \mid E\,[\phi\ \{\chi\}\,U\,\{\chi'\}\ \phi'] \mid E\,[\phi\ \{\chi\}\,U\ \phi'] \mid \\ & E[\phi\ \{\chi\}\,W\,\{\chi'\}\ \phi'] \mid E[\phi\ \{\chi\}\,W\ \phi'] \mid EX\,\{\chi\}\ \phi \mid EF\,\psi \mid EF\,\{\chi\}\ \psi \mid \neg\,\phi\end{aligned}$$

A product S_p of S is said to be a *live refinement* (of S) if $S_p \models AG\,\langle\mathit{true}\rangle\,\mathit{true}$, i.e. S_p has only infinite (full) paths. A formula ϕ is said to be *preserved by live refinement* if $S \models \phi$ implies $S_p \models \phi$, for all live refinements S_p of S.

Theorem 3 (Preservation by live refinement). *Any formula ϕ expressed in v-ACTLive$^\Box$ is preserved by live refinement.*

VMC notifies the user whenever preservation of a verification result is applicable.

The preservation of v-ACTLive$^\Box$ formulae obviously is an important improvement over the preservation of v-ACTL$^\Box$ formulae, since it allows family-based verification in a lot more cases. Finally, it is worthwhile to remark that an MTS in which every path is infinite is by definition live and while this might seem a rather strong condition, many reactive systems actually exhibit infinite behaviour, so the class of live MTS includes many models of practical interest.

If we want to actually verify a v-ACTL formula ϕ over the behavioural MTS model that encodes the original FTS behaviour, it suffices to verify the formula $[\texttt{behaviour}]\,\phi$. This guarantees that the signature is ignored.

7 Example in VMC

In this section, we illustrate the transformation on the beverage vending machine example SPL from [13]. The feature diagram in Fig. 6 models its valid products, defining 12 vending machines based on the features `Soda`, `Tea`, `FreeDrinks` and `CancelPurchase`. The allowed product behavior is modelled by the FTS in Fig. 7.

Figure 8 shows the input model in VMC after having applied the transformation described in Sect. 5 to the FTS in Fig. 7. The corresponding MTS, as

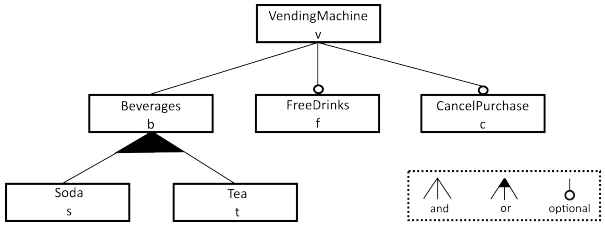


Fig. 6. Feature diagram of vending machine product line from [13]

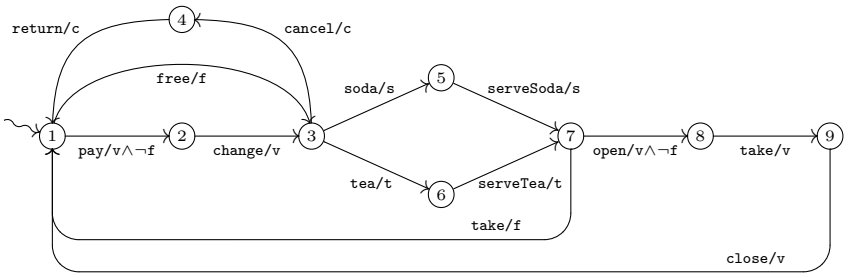


Fig. 7. FTS of vending machine product line from [13]

generated by VMC, is shown in Figure 5 (on the right-hand side). Note that we have omitted all dummy actions in the signature part (for ease of presentation).

Some sample formulae/properties that can be verified over the example are:

1. [`behaviour`] `AG AF`{`pay or free`}`true`: *Infinitely often, either action pay or action free occurs.*
2. [`behaviour`] `AG`[`open`]`AF`{`close`}`true`: *It is always the case that action open is eventually followed by action close.*
3. [`behaviour`] `AG AF`{ `cancel or serveSoda or serveTea` }`true`: *Infinitely often, either action cancel or action serveSoda or action serveTea occurs.*
4. [`behaviour`] `not E`[`true` {`not tea`} U {`serveTea`} `true`]: *It is not possible that action serveTea occurs without being preceded by action tea.*
5. [`behaviour`] [`pay`] `AF`{`takePaid`}`true`: *Whenever action pay occurs, eventually action takePaid occurs.*

Figure 9 shows the result of verifying formula 4 over the MTS. We see that this formula is true and, since it is a v-ACTLive$^{\Box}$ formula, VMC reports that this result is preserved by all products of the product line (hence in particular by the valid ones). VMC can also generate all valid products, upon which it lists all 12 valid products of the input model, providing for each a list of the action labels of all may transitions that have been preserved (as must transitions) in that product. These can then be used to perform product-based verification.

Figure 10 shows the result of verifying the v-ACTLive$^{\Box}$ formula 5 over all valid products. We see that this formula is true for all products, except for those

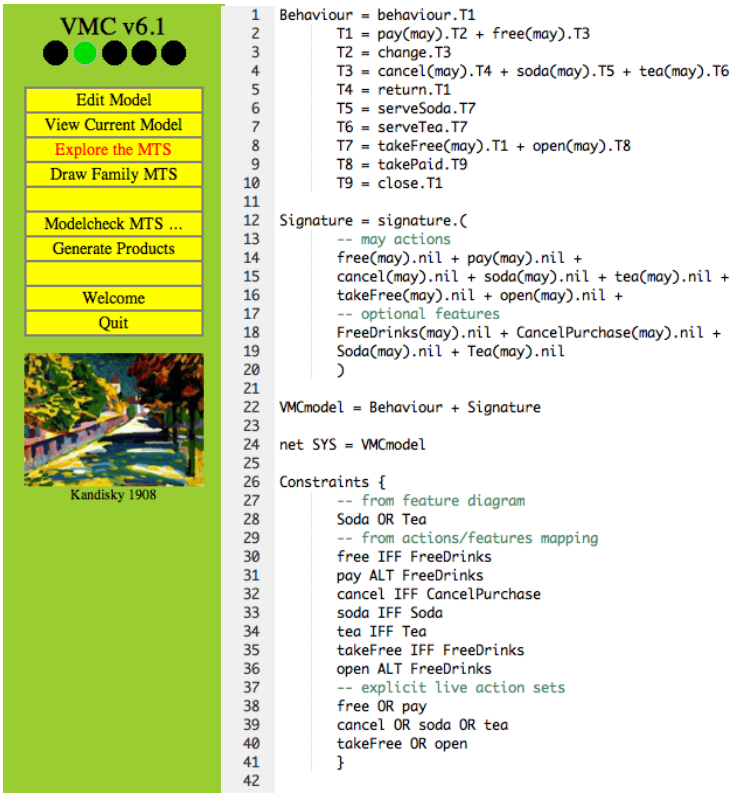


Fig. 8. VMC input model of vending machine product line

that allow to cancel a payment, i.e. those that have the `CancelPurchase` feature but at the same time lack the `FreeDrinks` feature.

Clicking one of these products, VMC loads it and opens a new window with the product's process model. Subsequently, the corresponding LTS can be visualised or properties can be verified directly over this product.

8 Conclusions and Future Work

We have presented an automatic technique to transform FTS into the constrained form of MTS accepted by VMC. The crux of this transformation is to go from variability constraints expressed in terms of features to variability constraints expressed in terms of actions. This paper thus contributes to a better understanding of the fundamental characteristics of the two models. Finally, we have showed how a well-known FTS example from the literature can be transformed and analysed with VMC.

VMC is a product of the KandISTI family of model checkers developed at ISTI–CNR in Pisa [7, 19]. This modelling and verification framework is publicly accessible online at the URL `http://fmt.isti.cnr.it/kandisti`. KandISTI

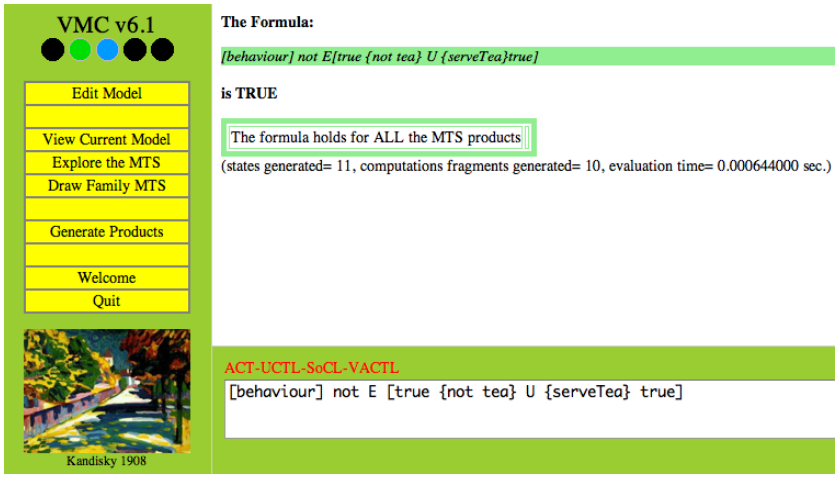


Fig. 9. Formula 4 verified by VMC over vending machine product line

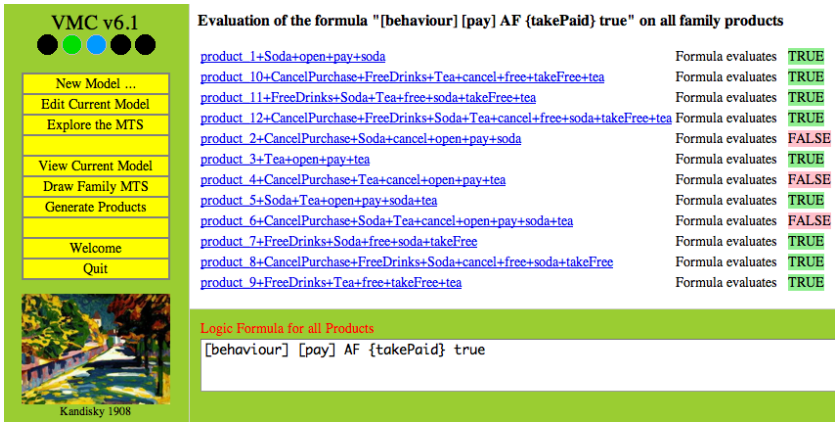


Fig. 10. Formula 5 verified by VMC over all products of vending machine product line

is an experimental analysis environment whose target is not primarily full-scale industrial-sized system/software verification, but rather the development of and experimentation with new ideas and approaches concerning the analysis of system designs. KandISTI is a framework in continuous evolution. VMC is its most recent extension developed for the purpose of exploring verification strategies for configurable systems (such as product lines). The basic idea underlying VMC is the use of 'constrained' MTS for the modelling of variability. Since FTS are the input model of other highly successful approaches to modelling (and model checking) variability-intensive systems, it is important to understand the relation between these two approaches in detail. This involves comparing them on larger examples, and comparing also their analysis capabilities. This paper is another

step in this direction, after the preliminary comparison in [3]. In the future, we intend to perform a quantitative evaluation of the expressivity, complexity and scalability of both approaches, as well as of the complexity of the transformation. Finally, we intend to consider also the state labelling of FTS by switching from a purely process-algebraic description of MTS in VMC to a richer modelling language. Other KandISTI members, with whom VMC shares the underlying verification engine, in fact have both an action and a state labelling [6].

Acknowledgments We thank the anonymous reviewers for their useful comments.

References

1. S. Apel, D. S. Batory, C. Kästner, and G. Saake. *Feature-Oriented Software Product Lines: Concepts and Implementation.* Springer, 2013.
2. P. Asirelli, M. H. ter Beek, A. Fantechi, and S. Gnesi. A Logical Framework to Deal with Variability. In *IFM*, volume 6396 of *LNCS*, pages 43–58. Springer, 2010.
3. P. Asirelli, M. H. ter Beek, A. Fantechi, and S. Gnesi. Formal Description of Variability in Product Families. In *SPLC*, pages 130–139. IEEE, 2011.
4. C. Baier and J.-P. Katoen. *Principles of Model Checking.* MIT Press, 2008.
5. M. H. ter Beek, A. Fantechi, S. Gnesi, and F. Mazzanti. Modelling and Analysing the Variability in Product Families: Model Checking of Modal Transition Systems.
6. M. H. ter Beek, A. Fantechi, S. Gnesi, and F. Mazzanti. A state/event-based model-checking approach for the analysis of abstract system properties. *Sci. Comput. Program.*, 76(2):119–135, 2011.
7. M. H. ter Beek, S. Gnesi, and F. Mazzanti. From EU Projects to a Family of Model Checkers. In *Software, Services and Systems*, volume 8950 of *LNCS*, pages 312–328. Springer, 2015.
8. M. H. ter Beek, S. Gnesi, and F. Mazzanti. Model Checking Value-Passing Modal Specifications. In *PSI*, volume 8974 of *LNCS*, pages 304–319. Springer, 2015.
9. M. H. ter Beek and F. Mazzanti. VMC: Recent Advances and Challenges Ahead. In *SPLC*, volume 2, pages 70–77. ACM, 2014.
10. M. H. ter Beek, F. Mazzanti, and A. Sulova. VMC: A Tool for Product Variability Analysis. In *FM*, volume 7436 of *LNCS*, pages 450–454. Springer, 2012.
11. A. Classen, M. Cordy, P. Heymans, A. Legay, and P. Schobbens. Formal semantics, modular specification, and symbolic verification of product-line behaviour. *Sci. Comput. Program.*, 80(B):416–439, 2014.
12. A. Classen, M. Cordy, P. Heymans, A. Legay, and P.-Y. Schobbens. Model checking software product lines with SNIP. *STTT*, 14(5):589–612, 2012.
13. A. Classen, M. Cordy, P.-Y. Schobbens, P. Heymans, A. Legay, and J.-F. Raskin. Featured Transition Systems: Foundations for Verifying Variability-Intensive Systems and Their Application to LTL Model Checking. *IEEE TSE*, 39(8):1069–1089, 2013.
14. A. Classen, P. Heymans, P.-Y. Schobbens, A. Legay, and J.-F. Raskin. Model Checking Lots of Systems: Efficient Verification of Temporal Properties in Software Product Lines. In *ICSE*, pages 335–344. ACM, 2010.
15. M. Cordy, A. Classen, P. Heymans, P.-Y. Schobbens, and A. Legay. ProVeLines: a product line of verifiers for software product lines. In *SPLC*, pages 141–146. ACM, 2013.

16. R. De Nicola, A. Fantechi, S. Gnesi, and G. Ristori. An Action Based Framework for Verifying Logical and Behavioural Properties of Concurrent Systems. In *CAV*, volume 575 of *LNCS*, pages 37–47. Springer, 1991.
17. A. Fantechi and S. Gnesi. Formal modeling for product families engineering. In *SPLC*, pages 193–202. IEEE, 2008.
18. D. Fischbein, S. Uchitel, and V. A. Braberman. A Foundation for Behavioural Conformance in Software Product Line Architectures. In *ROSATEA*, pages 39–48. ACM, 2006.
19. S. Gnesi and F. Mazzanti. An Abstract, on the Fly Framework for the Verification of Service-Oriented Systems. In *Rigorous Software Engineering for Service-Oriented Systems*, volume 6582 of *LNCS*, pages 390–407. Springer, 2011.
20. K. Larsen, U. Nyman, and A.Wąsowski. Modal I/O automata for interface and product line theories. In *ESOP*, volume 4421 of *LNCS*, pages 64–79. Springer, 2007.
21. K. Larsen and B. Thomsen. A Modal Process Logic. In *LICS*, pages 203–210. IEEE, 1988.
22. K. Lauenroth, K. Pohl, and S. Töhning. Model Checking of Domain Artifacts in Product Line Engineering. In *ASE*, pages 269–280. IEEE, 2009.
23. K. Pohl, G. Böckle, and F. J. van der Linden. *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer, 2005.
24. I. Schaefer, R. Rabiser, D. Clarke, L. Bettini, D. Benavides, G. Botterweck, A. Pathak, S. Trujillo, and K. Villela. Software diversity: state of the art and perspectives. *STTT*, 14(5):477–495, 2012.
25. P. Schobbens, P. Heymans, and J. Trigaux. Feature Diagrams: A Survey and a Formal Semantics. In *RE*, pages 136–145. IEEE, 2006.
26. T. Thüm, S. Apel, C. Kästner, I. Schaefer, and G. Saake. A Classification and Survey of Analysis Strategies for Software Product Lines. *ACM Comput. Surv.*, 47(1), 2014.