2011 15th International Software Product Line Conference

Formal Description of Variability in Product Families

Patrizia Asirelli, Maurice H. ter Beek, Stefania Gnesi
CNR, Istituto di Scienza e Tecnologie dell'Informazione
ISTI–CNR, Pisa, Italy
Email: {asirelli,terbeek,gnesi}@isti.cnr.it

Alessandro Fantechi
DSI, University of Florence, Italy
ISTI–CNR, Pisa, Italy
Email: fantechi@dsi.unifi.it

***Abstract*—We illustrate how to manage variability in a single logical framework consisting of a Modal Transition System (MTS) and an associated set of formulae expressed in the branching-time temporal logic MHML interpreted in a deontic way over such MTSs. We discuss the commonalities and differences with the framework of Classen et al. based on Featured Transition Systems and Linear-time Temporal Logic.**

I. Introduction

Decades after their introduction in [29], *Modal Transition Systems* (MTSs) and several variants have been recognised as a formal model for behavioural aspects of product families [1], [3]–[7], [18], [19], [21], [28], [30]. An MTS is a Labelled Transition System (LTS) with a distinction among *may* and *must* transitions, which can be seen as optional and mandatory for the products of a family. Hence, given a product family, a single MTS allows the definition of both:

1) its *underlying behaviour*, by means of states and actions, shared among all products, and
2) its *variation points*, by means of possible and mandatory transitions, differentiating between products.

While it can model *optional* and *mandatory* features, an MTS alone cannot model constraints regarding *alternative* features nor those regarding *requires* and *excludes* inter-feature relations. In [5], we therefore defined a branching-time temporal logic, MHML, able to express such constraints (and behavioural properties alike) and envisioned an algorithm to derive LTSs describing valid products from an MTS describing a product family *and* an associated set of MHML formulae expressing further constraints for the family.

Our approach is thus to manage variability in product families with a single logical framework consisting of an MTS and a set of MHML formulae (interpreted over MTSs).

Also *Featured Transition Systems* (FTSs) [14] were defined to describe the behaviour of a product family in a single model. An FTS is a Doubly-Labelled Transition System (L^2TS) with an associated feature diagram, a distinction among transitions by means of a labelling indicating which transitions correspond to which features, and an ordering of the transitions that correspond to *alternative* features.

Both approaches thus model product families in terms of specific transition systems that define a family's behaviour in terms of actions (features). Likewise, both approaches require the addition of further structural relationships between actions to manage (advanced) variability constraints. In this paper, we illustrate their commonalities and differences by applying both approaches to the same running example.

As such, we continue the research we initiated in [3]–[5]. In [3], we showed how to finitely characterise a subclass of MTSs by means of deontic logic formulae. In [4], we presented an initial attempt at a logical framework capable of addressing both static and behavioural conformance of products of a product family, by defining a deontic extension of an action- and state-based branching-time temporal logic interpreted over doubly-labelled MTSs. In [5], we introduced MHML and presented a first step towards a modelling and verification framework by providing a global model-checking algorithm to verify MHML formulae over MTSs.

Modelling and verifying static constraints over products of a product family usually requires separate expressions in a first-order logic [8], [20], [31], whereas modelling and verifying behavioural properties over the products of a product family is typically not addressed in feature modelling.

Related Work

We first discuss analysis approaches close to ours in detail.

In [21], an algorithm is defined to check conformance of LTSs against MTSs according to a given branching relation, i.e. check 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 [30], 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 [9] are verified by explicit-state model checking, progressing one state at a time. Like modal I/O automata [28], variable I/O automata extend I/O automata with a distinction among *may* and *must* transitions.

978-0-7695-4487-8/11 $26.00 © 2011 IEEE
DOI 10.1109/SPLC.2011.34

IEEE computer society

In [14], an explicit-state model-checking technique to verify linear-time temporal properties over 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 [15], this approach is improved by using symbolic model checking, examining sets of states at a time, and a feature-oriented version of CTL.

Finally, we briefly mention some further related approaches. In [18], [19], we extended MTSs to model advanced variability constraints regarding alternative features. In [24], [25], an algebraic approach to behavioural modelling and analysis of product families was developed, while feature Petri nets were introduced in [34] to model behaviour of product families with a high degree of variability.

Outline

After a brief description in Sect. II of a simple running example used throughout the paper, we discuss MTSs and FTSs in detail in Sect. III. In Sect. IV, we discuss MHML and show how it can be used to manage variability in our framework. In Sect. V, we compare our framework with that of [14]. In Sect. VI, we show how to derive valid products from a product family modelled in our framework. We conclude and outline future work in Sect. VII.

II. Running Example: Family of Coffee Machines

For easy comparison with [3]–[5], [12], [14] we show our contribution through a family of (simplified) coffee machines as running example, with the following list of requirements.

1) The only accepted coins are: one euro coin (1€), exclusively for European products and one dollar coin (1$), exclusively for Canadian products (constraints);
2) After inserting a coin, the user has to choose whether (s)he wants sugar, by pressing one of two buttons, after which (s)he may select a beverage (temporal ordering);
3) The choice of beverage (coffee, tea, cappuccino) varies between products. However, coffee must be offered by all products of the family, while cappuccino is only offered by European products (constraints);
4) Optionally, a ringtone is rung after delivering a beverage. However, a ringtone must be rung in products offering cappuccino (temporal ordering and constraint);[1]
5) After the beverage is taken, the machine returns idle (temporal ordering).

This list of requirements contains (static) constraints (1, 3, and part of 4) defining the differences between products, as well as temporal orderings (2, part of 4, and 5) defining the products' behaviour through admitted sequences of actions.

[1]Note that this requirement 4) has been modified slightly w.r.t. [3]–[5].

III. Background: Behavioural Models for Product Families

We adopt the definitions of two behavioural models capable of dealing with the variability notions characterising product families at different levels of detail: Modal Transition Systems (MTSs) and Featured Transition Systems (FTSs). Their underlying models are (Doubly-)Labelled Transition Systems (L$^{(2)}$TSs).

Definition 1. *A* Labelled Transition System *(LTS) is a 4-tuple $(Q, A, \overline{q}, \delta)$, with set Q of states, set A of actions, initial state $\overline{q} \in Q$, and transition relation $\delta \subseteq Q \times A \times Q$.*

A Doubly-Labelled Transition System *(L^2TS) is a 6-tuple $(Q, A, \overline{q}, \delta, AP, L)$, with* underlying *LTS $(Q, A, \overline{q}, \delta)$, set AP of atomic propositions, and labelling function $L : Q \to 2^{AP}$ associating a subset of AP to each state of the LTS.*

If $(q, a, q') \in \delta$, then we also write $q \xrightarrow{a} q'$. □

Since we intend to characterise the behaviour of product families, we need a notion for the evolution of time in LTSs.

Definition 2. *Let $(Q, A, \overline{q}, \delta)$ be an LTS with $q \in Q$. Then σ is a* path *from q if $\sigma = q$ (empty path) or $\sigma = q_1 a_1 q_2 a_2 q_3 \cdots$ with $q_1 = q$ and $q_i \xrightarrow{a_i} q_{i+1}$ for all $i > 0$ (possibly infinite path).*

A full path *is a path that cannot be extended further, i.e. it is infinite or ends in a state without 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 $\sigma\{i\}$. □

A. MTS: Modal Transition Systems [29]

Definition 3. *A Modal Transition System (MTS) is a 5-tuple $(Q, A, \overline{q}, \delta^\Diamond, \delta^\Box)$, with* underlying *LTS $(Q, A, \overline{q}, \delta^\Diamond \cup \delta^\Box)$. An MTS has two distinct 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, $\delta^\Box \subseteq \delta^\Diamond$.*

If $(q, a, q') \in \delta^\Diamond$, then we also write $q \xrightarrow{a}_\Diamond q'$ and likewise we also write $q \xrightarrow{a}_\Box q'$ for $(q, a, q') \in \delta^\Box$. □

MTSs thus allow the distinction of a special type of path.

Definition 4. *Let $\mathcal{M}$ be an MTS and let $\sigma = q_1 a_1 q_2 a_2 q_3 \cdots$ be a full path in its underlying LTS. Then σ is a* must path *(from q_1) in $\mathcal{M}$ 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 $\Box\text{-}path(q_1)$. A must path σ is denoted by $\sigma^\Box$. □

An MTS provides an abstract description of a product family's set of (valid) products, based on a 'translation' from constraints to transitions labelled with actions and a temporal ordering among them. The products differ w.r.t. the actions they can perform in any given state: the MTS must allow all possibilities desired for each derivable valid product, affirming the choices that make a product belong to the family.

The MTS depicted in Fig. 1, in which dashed arcs are used for the may transitions that are not must transitions ($\delta^\Diamond \setminus \delta^\Box$)

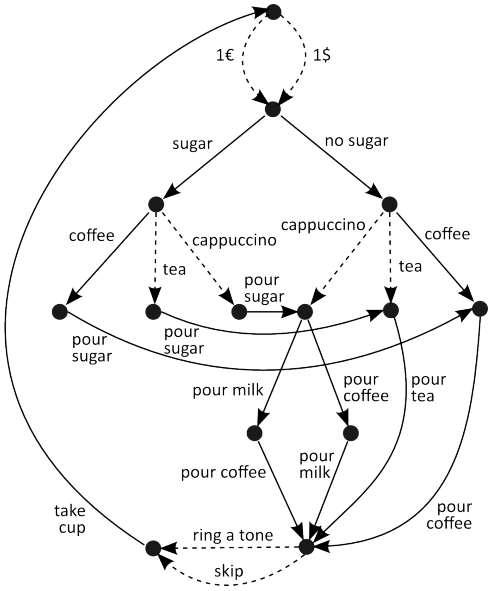

Figure 1. MTS modelling the family of coffee machines.

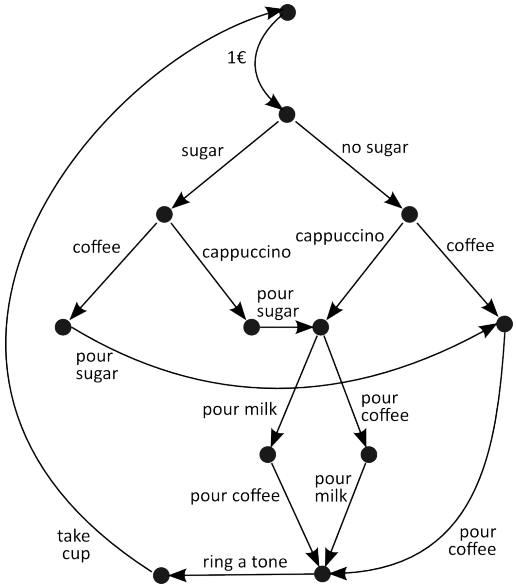

Figure 2. LTS modelling a European coffee machine.

and solid ones for must transitions ($\delta^{\Box}$), is an attempt to model all possible behaviours conceived for the family of coffee machines described in Sect. II.

Note that an MTS is thus able to model the constraints concerning *optional* and *mandatory* features (by means of may and must transitions). However, no MTS is able to model the constraint that 1€ and 1$ are exclusive nor that a cappuccino is not offered in Canadian products. We now make this claim somewhat more clear by defining how to generate products (LTSs) from a product family (MTS).

Definition 5. *Given an MTS* $\mathcal{M} = (Q, A, \overline{q}, \delta^{\Diamond}, \delta^{\Box})$ *that specifies a family, 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* $\delta^{\Box} \cup R$*, for some* $R \subseteq \delta^{\Diamond}$*, defined over a set of states* $Q_i \subseteq Q$*, such 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 $\mathcal{M}$*, denoted by* $\mathcal{P}_i \vdash \mathcal{M}$*, if and only if* $\overline{q}_i \vdash \overline{q}$*, where* $q_i \vdash q$ *holds, for some* $q_i \in Q_i$ *and* $q \in Q$*, if and only if:*

1) *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*
2) *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* $q'_i \vdash q'$*.* □

From the MTS depicted in Fig. 1, Def. 5 allows for the derivation of products (LTSs), including the European and Canadian coffee machines depicted in Figs. 2 and 3. Note, however, that Def. 5 also allows for the derivation of the LTS depicted in Fig. 4, but the product it models violates the constraints of requirements 1 and 3 (cf. Sect. II) by allowing the insertion of 1€ and 1$ and offering cappuccino.

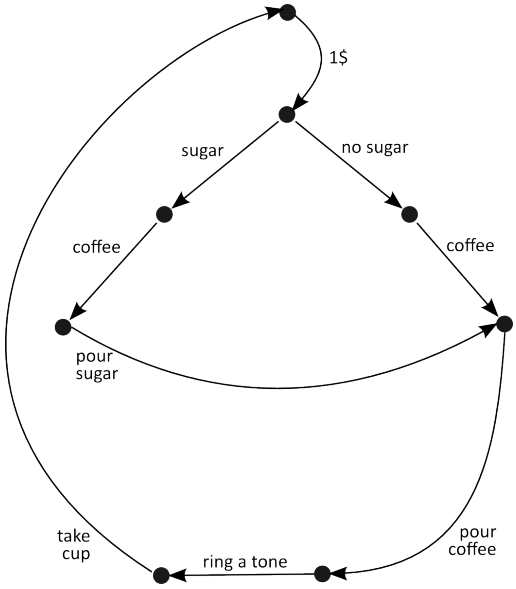

Figure 3. LTS modelling a Canadian coffee machine.

B. FTS: Featured Transition Systems [14]

A feature model for the coffee machine family of Sect. II is depicted in Fig. 5. It defines a family of 12 different valid products (coffee machines defined by their sets of features):

$\{\{m, s, o, b, c, \in\}, \{m, s, o, b, c, \in, r\}, \{m, s, o, b, c, \in, t\},$
$\{m, s, o, b, c, \$\}, \{m, s, o, b, c, \$, r\}, \{m, s, o, b, c, \$, t\},$
$\{m, s, o, b, c, \in, t, r\}, \{m, s, o, b, c, \in, p, r\},$
$\{m, s, o, b, c, \$, t, r\}, \{m, s, o, b, c, \in, p, r, t\}\}$

The LTSs depicted in Figs. 2 and 3 correspond to two of these 12 products: $\{m, s, o, b, c, \in, p, r\}$ and $\{m, s, o, b, c, \$, r\}$.

FTSs model constraints by explicitly referring to features.

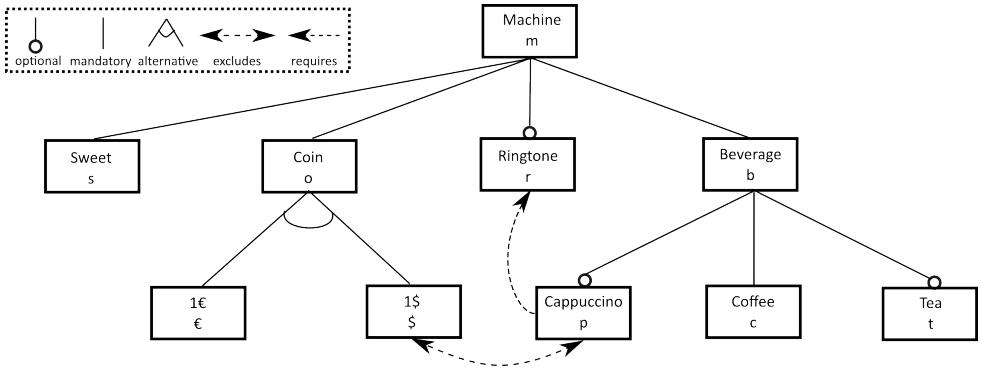

Figure 5. Feature model of the family of coffee machines (with shorthand names).

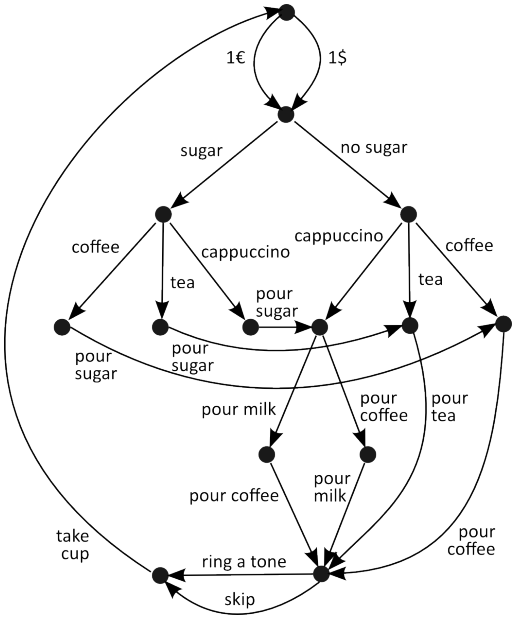

Figure 4. LTS derived from the MTS depicted in Fig. 1.

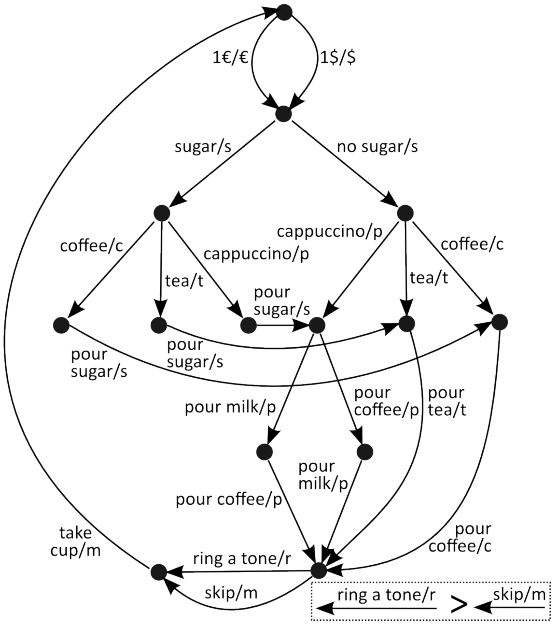

Figure 6. FTS modelling the family of coffee machines.

Definition 6. *A* Featured Transition System *(FTS) is a 9-tuple* $(Q,A,\overline{q},\delta,AP,L,FD,\gamma,>)$*, with* underlying L^2TS $(Q,A,\overline{q},\delta,AP,L)$*, feature diagram FD over a set* $\mathbb{F}$ *of features, total function* $\gamma : \delta \to \mathbb{F}$ *labelling transitions with features, and partial order* $> \subseteq \delta \times \delta$ *defining an ordering among transitions.* □

A transition of an FTS is thus labelled with an action *and* a feature and multiple outgoing transitions may be ordered.

The L^2TS depicted in Fig. 6,[2] with the feature label of a transition shown next to its action label (separated by a /), and the feature diagram depicted in Fig. 5 together form an FTS that models the family of coffee machines described in Sect. II. The partial order $\overset{\text{ring a tone/r}}{\longleftarrow} > \overset{\text{skip/m}}{\longleftarrow}$ establishes that all products containing both the features *r* and *m* only have the transition $\overset{\text{ring a tone/r}}{\longleftarrow}$ out of these two transitions.

[2]For readability, the set *AP* of atomic propositions was omitted in Fig. 6.

It is important to note that it is the inclusion of the full feature diagram that allows an FTS to manage variability.

A (*correct*) product (L^2TS) is obtained from a product family (FTS) by projection on that product's set of features.

Definition 7. *Given an FTS* $\mathcal{F} = (Q, A, \overline{q}, \delta, AP, L, FD, \gamma, >)$ *that specifies a family, a set of products specified as a set of* L^2TSs $\{\mathcal{P}_p = (Q_p, A_p, \overline{q}, \delta_p, AP_p, L_p) \mid p \subseteq 2^{\mathbb{F}}\}$ *may be derived by considering each transition relation* δ_p *to be obtained from* δ *by pruning (i) all transitions labelled with features not in p and (ii) all transitions overriden by transitions preceding them in the partial order.*

More precisely, we say that $\mathcal{P}_p$*, with* $p \subseteq 2^{\mathbb{F}}$*,* is a product of $\mathcal{F}$*, denoted by* $\mathcal{P}_p \vdash \mathcal{F}$*, if and only if:*

$$\delta_p = \{\, q \xrightarrow{a} q_1 \in \delta \mid \gamma(q \xrightarrow{a} q_1) \in p \wedge \nexists\, q \xrightarrow{b} q_2 \in \delta : \gamma(q \xrightarrow{b} q_2) \in p \wedge q \xrightarrow{b} q_2 > q \xrightarrow{a} q_1 \,\}. \quad \square$$

From the FTS depicted in Fig. 6, Def. 7 allows the derivation of all 12 products (L^2TSs) of the coffee machine family.

It is important to note a fundamental difference between product derivation as defined at this point for MTSs (Def. 5) and as defined for FTSs (Def. 7) [14], illustrated in Fig. 7: in the latter case, only valid products, i.e. satisfying the requirements, are derived by explicitly using the feature diagram, whereas products derived correctly by Def. 5 may violate constraints that MTSs cannot model, i.e. those concerning *alternative* features and the *requires* and *excludes* inter-feature relations, and thus need not satisfy the requirements. The LTS depicted in Fig. 4 is an example of a correct product of the MTS depicted in Fig. 1 (according to Def. 5) that is not a valid product of the family of coffee machines.

IV. MHML: A Logic to Express Variability

In [5], we defined a logical framework in which to express both variability constraints over the products of a family and behavioural properties over products and families alike. To this aim, we defined the action-based and branching-time temporal logic MHML based on the "Hennessy–Milner logic with Until" defined in [16], [27], but we interpreted it over MTSs rather than LTSs. MHML extends this logic by considering the different type of transitions of an MTS and by incorporating the existential and universal state operators (quantifying over paths) from CTL. As such, MHML is derived from the logics defined in [16], [23], [26], [27] and it is an action-based variant of the logic proposed in [4].

MHML defines state formulae (denoted by ϕ), path formulae (denoted by π), and action formulae (boolean compositions of actions, denoted by φ, with the usual semantics taken from [16]) over a set A of atomic actions $\{a, b, \ldots\}$. For model-checking purposes, its syntax is slightly extended w.r.t. [5].

Definition 8. *The syntax of MHML is:*

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

The intuitive interpretation of the nonstandard operators is:

- $\langle a\rangle\phi$: a next state exists, reachable by a *must* transition executing action a, in which ϕ holds
- $[a]\phi$: in all next states, reachable by a *may* or *must* transition executing action a, ϕ holds
- $E\pi$: there exists a full path on which π holds
- $A\pi$: on all possible full paths, π holds
- $\phi\ \{\varphi\}\ U\ \{\varphi'\}\ \phi'$: in a future state reached by an action satisfying φ', ϕ' holds, while ϕ holds from the current state until that state is reached and all actions executed in the meanwhile along the path satisfy φ
- $\phi\ \{\varphi\}\ U^{\Box}\ \{\varphi'\}\ \phi'$: in a future state reached by an action satisfying φ', ϕ' holds, while ϕ holds from the current state until that state is reached and the path leading to that state is a *must* path such that all actions executed in the meanwhile along the path satisfy φ

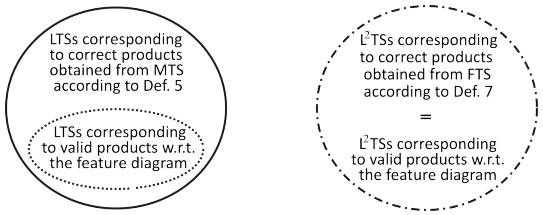

Figure 7. Venn diagrams of product derivation for MTSs (l) and FTSs (r).

The formal semantics of MHML is interpreted over MTSs.

Definition 9. *Let $(Q, A, \overline{q}, \delta^{\Diamond}, \delta^{\Box})$ be an MTS, with $q \in Q$ and a full path σ. The satisfaction relation $\models$ of MHML is:*

- $q \models true$ *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* $q \xrightarrow{a}_{\Box} q'$, *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* $\exists \sigma' \in path(q)\colon \sigma' \models \pi$
- $q \models A\pi$ *iff* $\forall \sigma' \in path(q)\colon \sigma' \models \pi$
- $\sigma \models \phi\ \{\varphi\}\ U\ \{\varphi'\}\ \phi'$ *iff* $\exists j \geq 1 :\ \sigma(j) \models \phi'$, $\sigma\{j\} \models \varphi'$, *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* $\sigma^{\Box} \models \phi\ \{\varphi\}\ U\ \{\varphi'\}\ \phi'$ $\Box$

The classical duality rule of Hennessy–Milner logic, stating $\langle a\rangle\phi = \neg[a]\neg\phi$, does not hold for MHML. In fact, $\neg[a]\neg\phi$ corresponds to a weaker version of the classical diamond operator which we denote as $P(a)\,\phi$: a next state *may* exist, reachable by executing action a, in which ϕ holds.

A number of further operators can be derived in the usual way: *false* abbreviates $\neg\, true$, $\phi \vee \phi'$ abbreviates $\neg(\neg\phi \wedge \neg\phi')$, and $\phi \Longrightarrow \phi'$ abbreviates $\neg\phi \vee \phi'$. $F\phi$ abbreviates $(true\ \{true\}\ U\ \{true\}\ \phi)$: there exists a future state in which 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. $AG\,\phi$ abbreviates $\neg EF\,\neg\phi$: in all states on all paths, ϕ holds. $AG^{\Box}\,\phi$ abbreviates $\neg EF^{\Box}\,\neg\phi$: in all states on all must paths, ϕ holds.

MHML interprets classical modal and temporal operators in a *deontic* way. Deontic logic formalises notions like violation, obligation, permission, and prohibition [2]. For product families, deontic logic allows to suitably capture constraints regarding optional and mandatory features. Indeed, MHML can express both permitted and obligatory actions.

MHML can also be interpreted over LTSs. Since LTSs consist of only *must* transitions, all MHML modalities must be interpreted as for the classical Hennessy–Milner logic: MHML's weaker version of the diamond operator, $P(a)\,\phi$, collapses onto the classical diamond operator, and MHML's box operator collapses onto the classical box operator.

A. Model-Checking Algorithm for MHML

The problem model checking aims to solve is: given a desired property, expressed as a formula ψ in a logic, and a model T, in the form of a transition system, decide whether $T \models \psi$ holds, where $\models$ is the logic's satisfaction relation. If $T \not\models \psi$, then it is usually easy to generate a counterexample. If T is finite, model checking reduces to a graph search.

Based on the parameters T and ψ, different strategies can be pursued to design model-checking algorithms. In [5], we defined a global model-checking algorithm for MHML by extending the classical algorithms for Hennessy–Milner logic and for (A)CTL [9], [10], [16], [33]. Recently [7], we implemented an on-the-fly model-checking algorithm for MHML as a particularization of the FMC model checker [32] for ACTL [16] over networks of automata (which can be specified in a CCS-like process algebra [22]).

On the basis of the algorithm presented in [5], model checking MHML formulae over MTSs can be achieved in a complexity that is linear w.r.t. the state space size. More details of our logical framework are given in [6], [7].

B. Managing Advanced Variability

We consider product families modelled by constraints (defining differences among products) and temporal orderings (defining product behaviour as admitted action sequences).

We complement MTS models of families with MHML formulae expressing the constraints that MTSs cannot capture. Recall from Sect. III that these concern the constraints regarding *alternative* features and those regarding *requires* and *excludes* inter-feature relations. We formalise an example of each of these constraints for our running example.[3]

Property A Coffee machines may not accept both 1€ and 1$ coins (1€ and 1$ are *alternative*):

$$(EF\,\langle 1€\rangle\, true \vee EF\,\langle 1\$\rangle\, true) \wedge$$
$$\neg\,(EF\,P(1€)\, true \wedge EF\,P(1\$)\, true)$$

This MHML formula formalises requirement 1) of Sect. II.

Property B Cappuccino may not be offered in Canadian coffee machines (cappuccino *excludes* 1$):

$$((EF\,P(cappuccino)\, true) \implies (AG\,\neg P(1\$)\, true)) \wedge$$
$$((EF\,P(1\$)\, true) \implies (AG\,\neg P(cappuccino)\, true))$$

This MHML formula formalises (a part of) requirement 3).

Property C Coffee machines offering cappuccino must ring a tone (cappuccino *requires* ring a tone):

$$(EF\,P(cappuccino)\, true) \implies (EF\,\langle ring\ a\ tone\rangle\, true)$$

This MHML formula formalises (a part of) requirement 4).

Each of these formulae combines constraints represented by a deontic interpretation in MHML with behavioural relations on actions expressed by the temporal part of MHML.

[3]Recall that the deontic permission operator P is represented by MHML's weaker version of the classical diamond modality, i.e. $P(a)\,\phi = \neg[a]\neg\phi$.

Property	Fig. 1	Fig. 2	Fig. 3	Fig. 4
A	*false*	*true*	*true*	*false*
B	*false*	*true*	*true*	*false*
C	*false*	*true*	*true*	*true*
D	*true*	*true*	*true*	*true*
E	*true*	*true*	*true*	*true*

Table I
VERIFYING PROPERTIES A–E ON FIGS. 1–4 WITH THE ALGORITHM FROM [5].

Property C expresses a constraint regarding a *requires* relation. Note that such a static relation does not imply any ordering: a coffee machine that rings a tone before producing a cappuccino cannot be excluded as a valid product of the family of coffee machines on the basis of this relation. It is the duty of the behavioural description of a product (family) as provided by an LTS (MTS) to impose temporal orderings.

We give two MHML formulae for such temporal orderings.

Property D Once a coffee has been selected, a coffee is eventually delivered:

$$AG\ [coffee]\ AF^{\Box}\ \{pour\ coffee\}\ true$$

Property E A coffee machine may never deliver a coffee before a coin has been inserted:

$$A\ [true\ \{\neg\ coffee\}\ U^{\Box}\ \{1\$ \vee 1€\}\ true]$$

Verifying Properties A–E over the MTS and LTSs depicted in Figs. 1–4 with FMC (by means of the model-checking algorithm from [5]) leads to the results presented in Table I.

V. A COMPARISON WITH THE FRAMEWORK OF [14]

Recall from Sect. III the following difference between MTSs and FTSs regarding product derivation (cf. Fig. 7). In case of FTSs, all and only products that are correct w.r.t. the requirements are derived (at the price of including a feature diagram in each FTS), whereas in case of MTSs also correctly derived products may violate constraints of the type that MTSs cannot model (e.g. the LTS depicted in Fig. 4). As a result, in our framework not all properties (expressed as MHML formulae) that hold for a product family (modelled as MTS) also hold for all the family's products (modelled as LTSs). However, if the frameworks are applied to product families in which variability is limited to *optional* and *mandatory* features, an FTS still includes a feature diagram whereas an MTS can do without an associated set of MHML formulae.

We have the following results regarding the preservation of properties from a family to its products in our framework.

Theorem 1. *Given an MTS $\mathcal{M}$ and an MHML formula ϕ, if $\mathcal{M} \models \phi$ then $\exists \mathcal{P} \vdash \mathcal{M}$ by Def. 5 such that $\mathcal{P} \models \phi$.*

Proof: Consider the product obtained by turning all may transitions into must ones: ϕ's may modalities collapse to must ones, and ϕ's interpretation on this LTS does not change. ■

Examples are Properties D and E that hold for the MTS depicted in Fig. 1 and for the LTSs depicted in Figs. 2 and 3.

Theorem 2. *Given an MTS $\mathcal{M}$ and an MHML formula ϕ, if $\mathcal{M} \not\models \phi$ then $\exists \mathcal{P} \vdash \mathcal{M}$ by Def. 5 such that $\mathcal{P} \not\models \phi$.*

Proof: This is a corollary of Thm. 1, taking $\phi = \neg\psi$. ■

Examples are Properties A and B that hold neither for the MTS depicted in Fig. 1 nor for the LTS depicted in Fig. 4.

The preservation of properties expressed as logical formulae from one transition system to another, on the basis of a *simulation* relation between these two transition systems, is well-studied in the literature. Intuitively, transition system $\mathcal{T}_1$ *is simulated by* transition system $\mathcal{T}_2$ if every transition in $\mathcal{T}_1$ can be 'mimicked' by some transition in $\mathcal{T}_2$. For an MTS $\mathcal{M}$ and any of its products $\mathcal{P}$ (an LTS), it immediately follows from Def. 5 that every must transition in $\mathcal{M}$ can indeed be 'mimicked' by some transition in $\mathcal{P}$. As a result, a sufficient condition for the preservation of MHML properties from a product family to its products is that these properties can be expressed as formulae in the fragment $\text{MHML}_{\Box}$ of MHML.

Definition 10. *The syntax of the fragment $\text{MHML}_{\Box}$ is:*
$\phi ::= true \mid \phi \wedge \phi' \mid \phi \vee \phi' \mid \langle a \rangle \phi \mid [a]\phi \mid EF^{\Box}\{\varphi\} \mid AF^{\Box}\{\varphi\} \mid E(\phi\{\varphi\} U^{\Box}\{\varphi\}\phi') \mid A(\phi\{\varphi\} U^{\Box}\{\varphi\}\phi') \mid EG^{\Box}\phi \mid AG^{\Box}\phi \mid AG\phi$ □

In particular, a formula of the form $[a]\phi$ is a universal formula which is preserved on the same basis as that of the classical result of preservation by simulation for CTL [11]. Instead, a formula of the form $\langle a \rangle \phi$ is preserved as it dictates the existence of a must path with certain characteristics, which by Def. 5 will necessarily be found in all products.

Formally, we thus obtain the following result (cf. Thm. 1).

Theorem 3. *Given an MTS $\mathcal{M}$ and an $\text{MHML}_{\Box}$ formula ϕ, if $\mathcal{M} \models \phi$ then $\forall \mathcal{P} \vdash \mathcal{M}$ by Def. 5 we have $\mathcal{P} \models \phi$.* □

An example is Property D, which holds for the MTS depicted in Fig. 1 and for the LTSs depicted in Figs. 2 and 3.

Consider the following Linear-time Temporal Logic (LTL) to express properties over FTSs, which is equivalent to that used in [14], but redefined with a slightly different syntax.[4]

Definition 11. $\phi ::= true \mid \neg\phi \mid \phi \wedge \phi' \mid X_a \phi \mid \phi\ U\ \phi'$ □

This allows us to reformulate two results from [14], which compared with the above three theorems regarding our framework show the main differences with the framework of [14].

Theorem 4. *Given an FTS $\mathcal{F}$ and an LTL formula ϕ, if $\mathcal{F} \models \phi$ then $\forall \mathcal{P} \vdash \mathcal{F}$ by Def. 7 we have $\mathcal{P} \models \phi$.* □

Theorem 5. *Given an FTS $\mathcal{F}$ and an LTL formula ϕ, if $\mathcal{F} \not\models \phi$ then $\exists \mathcal{P} \vdash \mathcal{F}$ by Def. 7 such that $\mathcal{P} \not\models \phi$.* □

Specifically note a subtle but important difference between Thm. 2 and Thm. 5, due to the difference between product derivation for MTSs (Def. 5) and that for FTSs (Def. 7) illustrated in Fig. 7. In case of Thm. 5, a product violating a desired logical property is nonetheless a valid product (according to the feature model), making the FTS an incorrect model of the family that must be corrected: either the feature model or its 'translation' into an FTS need to be redefined.

In case of Thm. 2, on the other hand, we have already seen that a product that violates the logical property need not be a valid product of the product family (cf. Fig. 7). While all products that can be derived from a product family by Def. 5 are correct w.r.t. all the requirements that MTSs can model (i.e. constraints regarding *optional* and *mandatory* features, but also temporal orderings), these products may be incorrect w.r.t. the constraints that MTSs cannot model (i.e. those regarding *alternative* features and the *requires* and *excludes* inter-feature relations). An example is the LTS depicted in Fig. 4, in which 1€ and 1$ are not *alternative*.

VI. Derivation of Valid Products from a Product Family

In Sect. IV-B, we showed how to formalise in MHML the constraints regarding *alternative* features and regarding the *requires* and *excludes* inter-feature relations. These example properties A, B, and C can easily be generalised into MHML template formulae ALT, EXC, and REQ for such constraints.

We define an algorithm to derive LTS descriptions of products; not from the MTS description of a product family alone, but also from an associated set of MHML formulae expressing further constraints. The outcome is a set of LTS descriptions of valid products, i.e. each satisfying all constraints (expressed in MHML). The underlying idea is to construct intermediate LTSs (partially covering the original MTS to a given depth) in a step-by-step fashion as follows. First unfold the initial MTS according to the type of transitions, meanwhile verifying formulae of type ALT and EXC, and continue only with the LTSs satisfying all formulae. In the end, when no more unfolding step can be applied to the LTSs, verify formulae of type REQ and (partially) ALT over these LTSs.

We illustrate this algorithm for the MTS depicted in Fig. 1. A first unfolding based on its type of transitions results in:

Next, the algorithm would consider the relevant MHML formula of type ALT (Property A: 1€ and 1$ are *alternative*):

$$(EF\,\langle 1\$\rangle\, true \vee EF\,\langle 1€\rangle\, true) \wedge$$
$$\neg(EF\,P(1\$)\, true \wedge EF\,P(1€)\, true)$$

Verifying this formula over the above intermediate LTSs implies that the algorithm would only continue to unfold the original MTS from the leaf states of the above intermediate LTSs (ii) and (iii) onwards. The final result of the algorithm would thus be a set of valid products, the behaviour of each guaranteeing that initially only a 1€ or a 1$ can be inserted.

[4] $X_a\,\phi$ says that in the next state of the path, reached by action a, ϕ holds.

A. Problem Statement

Given an MTS $\mathcal{M}$ and a set Φ of constraints expressed as MHML formulae ϕ_i, for some $i \geq 0$, derive the set P containing all LTSs $\mathcal{P}_j$, for some $j \geq 0$, for which $\mathcal{P}_j \vdash \mathcal{M}$ and $\mathcal{P}_j \models \phi_i$ for all $i \geq 0$.

B. Assumptions and Remarks

We assume that the formulae in Φ express only alternative, excludes and requires constraints in the form of the MHML template formulae ALT, EXC, and REQ obtained as generalizations of the properties A, B, and C provided in Sect. IV-B. We denote by Φ_{REQ} all formulae of type REQ and by Φ'_{ALT} the premises of all formulae of type ALT (i.e. the part that requires the *presence* of one of the *alternative* features).

For simplicity, whenever we refer to "a may transition $p \xrightarrow{a}_{\diamond} q$", we do not intend to include may transitions which are also must transitions (recall from Def. 3 that $\delta^{\Box} \subseteq \delta^{\diamond}$). Hence, we intend only transitions belonging to the transition relation $\delta^{\diamond} \setminus \delta^{\Box}$ (i.e. those drawn as dashed arcs in figures).

C. Derivation Step

A derivation step $derive(\mathcal{M}, p, \mathcal{L}, q)$ can be applied to a state p of the MTS $\mathcal{M}$ and a state q of a (partially developed) LTS $\mathcal{L}$ of a working set L of intermediate LTSs. Initially, q is a leaf state (i.e. with no outgoing transition) not visited before. Possible effects of applying $derive(\mathcal{M}, p, \mathcal{L}, q)$ are:

1) add must transitions to $\mathcal{L}$ from q to newly created states;
2) add must transitions to $\mathcal{L}$ from q to states visited before;
3) replace $\mathcal{L}$ in L by copies of $\mathcal{L}$, which differ from $\mathcal{L}$ only w.r.t. the outgoing transitions from q due to the application of one of the previous two points, and replace derivation steps in D involving $\mathcal{L}$ accordingly;
4) mark q as visited before, due to the application of one of the previous three points;
5) update the set D of derivation steps to be performed next by the algorithm.

The precise definition of the derivation steps is given next.

D. Derivation Algorithm

The algorithm starts with $\mathcal{M}$ and Φ as defined in Sect. VI-A, a set D of derivation steps which initially contains only $derive(\mathcal{M}, \overline{q}, \mathcal{L}, \overline{q})$, with $\mathcal{L}$ consisting only of the initial state $\overline{q}$ of $\mathcal{M}$, and a working set $L = \{\mathcal{L}\}$. The algorithm then applies derivation steps from D as long as it can do so.

A derivation step $derive(\mathcal{M}, p, \mathcal{L}, q)$ is defined as follows.

1) For each must transition $p \xrightarrow{a}_{\Box} p'$ in $\mathcal{M}$:
   a) if $p' = p$, add a must transition $q \xrightarrow{a}_{\Box} q$ to $\mathcal{L}$;
   b) if state p' was visited before, either in some derivation step $derive(\mathcal{M}, p', \mathcal{L}, q')$ or when handling some transition $p \xrightarrow{b}_{\Box} p'$ in the current derivation step, add a must transition $q \xrightarrow{a}_{\Box} q'$ to $\mathcal{L}$;
   c) else, add a new state q' and a must transition $q \xrightarrow{a}_{\Box} q'$ to $\mathcal{L}$, and add the derivation step $derive(\mathcal{M}, p', \mathcal{L}, q')$ to D;
   d) finally, for each formula $\phi \in \Phi \setminus \Phi_{\text{REQ}}$ in which a occurs, verify whether or not $\mathcal{L} \models \phi$. If $\mathcal{L} \not\models \phi$, remove $\mathcal{L}$ from the working set L and remove $derive(\mathcal{M}, p, \mathcal{L}, q)$ from D.
2) For all may transitions $p \xrightarrow{a_1}_{\diamond} p_1, \ldots, p \xrightarrow{a_n}_{\diamond} p_n$ in $\mathcal{M}$, step 1) needs to be 'copied' for all possible combinations to solve optionality, by adding must transitions. Hence, for each $Act \subseteq \{a_1, \ldots, a_n\}$:
   a) replace $\mathcal{L}$ in the working set L by the copies $\mathcal{L}_{Act}$;
   b) replace all derivation steps $derive(\mathcal{M}, p, \mathcal{L}, q)$ that involve $\mathcal{L}$, for some $\mathcal{M}$, p, and q, in the set D of derivation steps by the copies $derive(\mathcal{M}, p, \mathcal{L}_{Act}, q)$;
   c) for each may transition $p \xrightarrow{a_j}_{\diamond} p_j$ in $\mathcal{M}$, for some $1 \leq j \leq n$, which is such that $a_j \in Act$:
      i) if $p_j = p$, add a must transition $q \xrightarrow{a_j}_{\Box} q$ to $\mathcal{L}_{Act}$;
      ii) if state p_j was visited before, either in some derivation step $derive(\mathcal{M}, p', \mathcal{L}, q')$ or when handling some transition $p \xrightarrow{b}_{\diamond} p_j$ or $p \xrightarrow{b}_{\Box} p_j$ in the current derivation step, add a must transition $q \xrightarrow{a_j}_{\Box} q'$ to $\mathcal{L}_{Act}$;
      iii) else, add a new state q' and a must transition $q \xrightarrow{a_j}_{\Box} q'$ to $\mathcal{L}_{Act}$, and add the derivation step $derive(\mathcal{M}, p_j, \mathcal{L}_{Act}, q')$ to D;
   d) for each formula $\phi \in \Phi \setminus \Phi_{\text{REQ}}$ in which an action from Act occurs, verify whether or not $\mathcal{L}_{Act} \models \phi$. If $\mathcal{L}_{Act} \not\models \phi$, remove $\mathcal{L}_{Act}$ from the working set L and remove all $derive(\mathcal{M}, p, \mathcal{L}_{Act}, q)$, for some $\mathcal{M}$, p, and q, from D.

When no more derivation step of D can be applied, the algorithm verifies whether $\forall \mathcal{L} \in L,\ \phi \in \Phi_{\text{REQ}} \cup \Phi'_{\text{ALT}} : \mathcal{L} \models \phi$. If $\exists \mathcal{L} \in L,\ \phi \in \Phi_{\text{REQ}} \cup \Phi'_{\text{ALT}} : \mathcal{L} \not\models \phi$, this $\mathcal{L}$ is removed from L. The resulting L is the set P of valid products of $\mathcal{M}$.

E. Correctness and Complexity

An aspect that is currently not taken into account at the end of the algorithm is the deletion of LTSs that have no outgoing transitions from their initial states. We conclude from the example prior to the algorithm that such LTSs might result (in this example it is consequently deleted due to the MHML formulae for Property A, but this is of course purely circumstantial). In this regard, a question that deserves a more thorough examination is whether LTSs that reach a depth that is less than that reached by the original MTS are to be considered valid products or not. If not, then such undesired LTSs should be deleted in a final step.

The algorithm avoids a full exponential unfolding by applying the following method of reduction during derivation.

Constraints are applied as early as possible to cut undesired products. Some violations of constraints are detected early when the involved may transitions are all found in a certain state, but in general constraints relate transitions from separate states and can therefore halt exponential unfolding only when the algorithm comes across the last involved (may or must) transition. In case of the *requires* constraint, the right-hand side of the MHML template formula REQ ($EF\,\langle a\rangle\,true$) implies that this formula may only be used to cut undesired final products and *not* to cut intermediate products. This is because formulae of this type (based on the EF operator) may be invalid over an intermediate LTS, but valid over an LTS derived from the same MTS once fully unfolded.

In the future, we intend to extend this algorithm to allow also additional temporal ordering constraints (expressed in MHML) to be verified while deriving valid products.

VII. Conclusions and Future Work

In this paper, we continue the research initiated in [3]–[5]. In particular, we illustrate how to derive valid products from a product family by managing variability in a single logical framework consisting of an MTS and an associated set of MHML formulae. This framework can also be used to verify behavioural properties over products and families. Moreover, we outline some commonalities and differences with the framework of Classen et al. based on FTSs [14], [15].

We conclude that both frameworks are able to express families of products considering notions of variability commonly addressed in product line engineering. In both cases, the formal family description requires the following two ingredients, which cannot be expressed in a single notation:

1) a transition system, expressing product behaviour as well as some variation points;
2) further variability constraints, expressed separately by a feature diagram in the FTS-based framework, and by a set of MHML formulae in the MTS-based framework.

We observe, however, that our framework based on MTSs can actually be seen as a single formal framework, consisting of a logic (MHML) with its natural interpretation structure (MTSs). The notion of product derivation is defined inside this framework and logical formulae are used both as constraints and as behavioural properties to be verified for families and products alike. In the framework based on FTSs, the expression and verification of properties requires the introduction of a third ingredient, namely the logic.

Regarding the algorithms to derive the valid products from a product family, we observe that in the framework based on FTSs these are obtained by pruning transitions not involved according to the feature diagram. In our algorithm, on the other hand, an exponential explosion of possible products after unfolding is effectively contained by locally applying derivation steps that include the verification of constraints.

Several issues deserve further investigation, as suggested in the previous sections. A major issue that remains unsolved, is the definition of design and verification techniques and tools that, based on the principles expressed in this paper, but hiding most of the formality, can be routinely used by product line engineers. Ideally, engineers should be able to use high-level languages hiding all semantic details. This requires, among many other issues, a thorough investigation of the relation between features and actions when translating feature models into transition systems. Engineers should moreover be able to use a predefined taxonomy for properties specified in MHML (like the specification patterns repository for LTL, (A)CTL, etc. [17]), containing more template formulae than those presented in the previous section, allowing them to select an appropriate predefined property and model check it over the product (family) of interest. Another important issue is to study how our approach scales up to large, industrial-size product families with many variation points and many features.

Acknowledgments

We thank Franco Mazzanti and Aldi Sulova for detailed discussions on the algorithm defined in Sect. VI and we thank Andreas Classen for his comments that improved the paper.

The research for this paper was funded by the project D-ASAP (MIUR–PRIN 2007) and by the RSTL project XXL of the National Research Council of Italy (CNR).

References

[1] Antonik, A., Huth, M., Larsen, K.G., Nyman, U., Wąsowski, A.: 20 Years of modal and mixed specifications. B. EATCS 95, 94–129 (2008)

[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.: Deontic logics for modeling behavioural variability. In: Benavides, D., Metzger, A., Eisenecker, U. (eds.) Proceedings Int. Workshop on Variability Modelling of Software-intensive Systems (VaMoS 2009), pp. 71–76. ICB research rep. 29, Univ. Duisburg–Essen (2009)

[4] Asirelli, P., ter Beek, M.H., Fantechi, A., Gnesi, S.: A deontic logical framework for modelling product families. In: Benavides, D., Batory, D., Grünbacher, P. (eds.) Proceedings Int. Workshop on Variability Modelling of Software-intensive Systems (VaMoS 2010), pp. 37–44. ICB research rep. 37, Univ. Duisburg–Essen (2010)

[5] Asirelli, P., ter Beek, M.H., Fantechi, A., Gnesi, S.: A logical framework to deal with variability. In: Méry, D., Merz, S. (eds.) Proceedings Int. Conference on Integrated Formal Methods (IFM 2010). LNCS, vol. 6396, pp. 43–58. Springer, Berlin (2010)

[6] Asirelli, P., ter Beek, M.H., Fantechi, A., Gnesi, S.: A model-checking tool for families of services. In: Bruni, R., Dingel, J. (eds.) Proceedings Int. Conference on Formal Methods for Open Object-based Distributed Systems (FMOODS 2011). LNCS, vol. 6722, pp. 44–58. Springer, Berlin (2011)

[7] Asirelli, P., ter Beek, M.H., Fantechi, A., Gnesi, S., Mazzanti, F.: Design and validation of variability in product lines. To appear in Proceedings Int. Workshop on Product LinE Approaches in Software Engineering (PLEASE 2011) in Companion Proceedings Int. Conference on Software Engineering (ICSE 2011), ACM, New York (2011)

[8] Batory, D.: Feature Models, Grammars and propositional formulas. In: Obbink, J., Pohl, K. (eds.) Proceedings Int. Software Product Line Conference (SPLC 2005). LNCS, vol. 3714, pp. 7-20. Springer, Berlin (2005)

[9] 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)

[10] Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT, Cambridge (1999)

[11] Clarke, E.M., Jha, S., Lu, Y., Veith, H.: Tree-like counterexamples in model checking. In: Proceedings Symposium on Logic In Computer Science (LICS 2002), pp. 19–29. IEEE, Washington (2002)

[12] Classen, A.: Modelling with FTS: a collection of illustrative examples. Tech. rep. P-CS-TR SPLMC-00000001, PReCISE research center, Univ. of Namur (2010)

[13] Classen, A.: CTL model checking for software product lines in NuSMV. Tech. rep. P-CS-TR SPLMC-00000002, PReCISE research center, Univ. of Namur (2010)

[14] 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 Int. Conference on Software Engineering (ICSE 2010), pp. 335–344. ACM, New York (2010)

[15] Classen, A., Heymans, P., Schobbens, P.-Y., Legay, A.: Symbolic model checking of software product lines. To appear in: Proceedings Int. Conference on Software Engineering (ICSE 2011). ACM, New York (2011)

[16] De Nicola, R., Vaandrager, F.W.: Three logics for branching bisimulation. J. ACM 42 (2), 458–487 (1995)

[17] Dwyer, M.B., et al., Spec Patterns web site, http://patterns.projects.cis.ksu.edu/

[18] Fantechi, A., Gnesi, S.: A behavioural model for product families. In: Proceedings Int. Symposium on Foundations of Software Engineering (FSE 2007), pp. 521–524. ACM, New York (2007)

[19] Fantechi, A., Gnesi, S.: Formal modelling for product families engineering. In: Proceedings Int. Software Product Line Conference (SPLC 2008), pp. 193–202. IEEE, Washington (2008)

[20] Fantechi, A., Gnesi, S., Lami, G., Nesti, E.: A methodology for the derivation and verification of use cases for product lines. In: Nord, R.L. (ed.) Proceedings Int. Software Product Line Conference (SPLC 2004). LNCS, vol. 3154, pp. 255–265. Springer, Berlin (2004)

[21] Fischbein, D., Uchitel, S., Braberman, V.A.: A foundation for behavioural conformance in software product line architectures. In: Proceedings ISSTA 2006 workshop on Role of Software Architecture for Testing and Analysis (ROSATEA 2006), pp. 39–48. ACM, New York (2006)

[22] Gnesi, S., Mazzanti, F.: On the fly verification of networks of automata. In: Proceedings Int. Conference on Parallel and Distributed Processing Techniques & Applications (PDPTA 1999)

[23] Godefroid, P., Huth, M., Jagadeesan, R.: Abstraction-based model checking using modal transition systems. In: Larsen, K.G., Nielsen, M. (eds.) Int. Conference on Concurrency Theory (CONCUR 2001). LNCS, vol. 2154, pp. 426–440. Springer, Berlin (2001)

[24] Gruler, A., Leucker, M., Scheidemann, K.D.: Modelling and model checking software product lines. In: Barthe, G., de Boer, F.S. (eds.) Proceedings Int. Conference on Formal Methods for Open Object-based Distributed Systems (FMOODS 2008). LNCS, vol. 5051, pp. 113–131. Springer, Berlin (2008)

[25] Gruler, A., Leucker, M., Scheidemann, K.D.: Calculating and modelling common parts of software product lines. In: Proceedings Int. Software Product Line Conference (SPLC 2008), pp. 203–212. IEEE, Washington (2008)

[26] 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)

[27] Larsen, K.G.: Proof systems for satisfiability in Hennessy-Milner logic with recursion. Theor. Comput. Sci. 72 (2–3), 265–288 (1990)

[28] Larsen, K.G., Nyman, U., Wąsowski, A.: Modal I/O automata for interface and product line theories. In: De Nicola, R. (ed.) Proceedings European Symposium on Programming (ESOP 2007). LNCS, vol. 4421, pp. 64–79. Springer, Berlin (2007)

[29] Larsen, K.G., Thomsen, B.: A modal process logic. In: Proceedings Symposium on Logic In Computer Science (LICS 1988), pp. 203–210. IEEE, Washington (1988)

[30] Lauenroth, K., Pohl, K., Töhning, S.: Model checking of domain artifacts in product line engineering. In: Proceedings Int. Conference on Automated Software Engineering (ASE 2009), pp. 269–280. IEEE, Washington (2009)

[31] Mannion, M., Camara, J.: Theorem proving for product line model verification. In: van der Linden, F. (ed.) Proceedings Int. Workshop on Software Product-Family Engineering (PFE 2003). LNCS, vol. 3014, pp. 211–224. Springer, Berlin (2004)

[32] Mazzanti, F.: FMC v5.0, http://fmt.isti.cnr.it/fmc/ (2011)

[33] Müller–Olm, M., Schmidt, D.A., Steffen, B.: Model-checking: A tutorial introduction. In: Cortesi, A., Filé, G. (eds.) Proceedings Int. Symposium on Static Analysis (SAS 1999). LNCS, vol. 1694, pp. 330–354. Springer, Berlin (1999)

[34] Muschevici, R., Clarke, D., Proenca, J.: Feature Petri nets. In: Schaefer, I., Carbon, R. (eds.) Proceedings Int. Workshop on Formal Methods in Software Product Line Engineering (FMSPLE 2010). Tech. Rep., Univ. of Lancaster (2010)