A Logical Framework to Deal with Variability*

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

Abstract. We present a logical framework that is able to deal with variability in product family descriptions. The temporal logic MHML is based on the classical Hennessy–Milner logic with Until and we interpret it over Modal Transition Systems (MTSs). MTSs extend the classical notion of Labelled Transition Systems by distinguishing possible (*may*) and required (*must*) transitions: these two types of transitions are useful to describe variability in behavioural descriptions of product families. This leads to a novel *deontic* interpretation of the classical modal and temporal operators, which allows the expression of both constraints over the products of a family and constraints over their behaviour in a single logical framework. Finally, we sketch model-checking algorithms to verify MHML formulae as well as a way to derive correct products from a product family description.

1 Introduction

Product Line Engineering (PLE) is a paradigm to develop a family of products using a common platform and mass customisation [30,32]. This engineering approach aims to lower production costs of the individual products by letting them share an overall reference model of the product family, while at the same time allowing them to differ with respect to particular characteristics in order to serve, e.g., different markets. As a result, the production process in PLE is organised so as to maximise commonalities of the products and at the same time minimise the cost of variations.

Managing planned *variability* in product families has been the subject of extensive study in the literature on PLE, especially that concerning *feature modelling* [5,13,22], which provides compact representations of all the products of a PL in terms of their features. Variability modelling addresses how to explicitly define the features or components of a product family that are *optional*, *alternative*, or *mandatory*. Formal methods are then developed to show that a certain product belongs to a family, or to derive instead a product from a family, by means of a proper selection of the features or components.

* Research funded by the Italian project D-ASAP (MIUR–PRIN 2007) and by the RSTL project XXL of the CNR.

D. Méry and S. Merz (Eds.): IFM 2010, LNCS 6396, pp. 43–58, 2010.

Many years after their introduction in [26], *Modal Transition Systems* (MTSs) and several variants have been proposed as a formal model for defining product families [1,16,17,19,25,33]. An MTS is a Labelled Transition System (LTS) with a distinction among so-called *may* and *must* transitions, which can be seen as optional or mandatory for the products of the family. Hence, given a family of products, an MTS allows one to model in a single framework:

1. the *underlying architecture*, by means of states and transitions, modelling the product platform shared by all products, and
2. the *variation points*, by means of possible and required transitions, modelling the variability among different products.

Deontic logic [2,29] has recently become popular in computer science for modelling descriptional and behavioural aspects of systems, mainly because of the natural way of formalising concepts like violation, obligation, permission, and prohibition. This makes deontic logic an obvious candidate for expressing the conformance of products of a family with respect to variation points. Such a conformance concerns both *static* requirements, which identify the features that constitute the different products, and *behavioural* requirements, which describe how products differ in their ability to deal with events in time.

Taking into account the Propositional Deontic Logic (PDL) that was proposed in [8,9] and which combines the expression of permission and obligation with concepts from temporal logics, in [3,4] we laid the basis for the application of deontic logic to model variability in product families. We showed how to characterise certain MTSs in terms of deontic logic formulae in [3]. In [4], we presented a first attempt at a logical framework capable of addressing both static and behavioural conformance of products of a family, by defining a deontic extension of an action- and state-based branching-time temporal logic interpreted over so-called doubly-labelled MTSs. Model checking with this logic was left as future work. Modelling and verifying static constraints over the products of a family usually requires separate expressions in a first-order logic [5,18,27], whereas modelling and verifying dynamic behavioural constraints over the products of a family is typically not addressed in feature modelling.

The first contribution of this paper is the introduction of the action-based branching-time temporal logic MHML, which allows expressing both constraints over the products of a family and constraints over their behaviour in a single logical framework. MHML is based on the "Hennessy–Milner logic with Until" defined in [14,24], but it is interpreted over MTSs rather than LTSs. This leads to a novel *deontic* interpretation of the classical modal and temporal operators.

The second contribution is a first step towards a modelling and verification framework based on model-checking techniques for MHML. We do so by providing a global model-checking algorithm to verify MHML formulae over MTSs.

Related Work

In [1,16,17,19,25,33], (variants of) MTSs have been proposed for modelling and verifying the behaviour of product families. We have extended MTSs in [17] to

allow modelling different notions of behavioural variability. A different, algebraic approach to behavioural modelling and verification of product lines instead has been developed in [20,21]. In this paper, we continue research we started in [3,4]. In [3], we showed how to finitely characterise certain MTSs by means of deontic logic formulae. In [4], we presented a first attempt at a logical framework capable of addressing both static and behavioural conformance of products of a family, by defining a deontic extension of an action- and state-based branching-time temporal logic interpreted over so-called doubly-labelled MTSs.

In [12], the authors present a model-checking technique over so-called Featured Transition Systems (FTSs), which are able to describe the combined behaviour of an entire product family. Their main purpose is to provide a means to check that whenever a behavioural property is satisfied by an FTS, then it is also satisfied by every product of the PL, and whenever a property is violated, then not only a counterexample is provided but also the products of the PL that violate the property. The main difference between their approach and ours is our use of a branching-time temporal logic with a deontic flavour that allows us to express and verify in a single framework both behavioural properties and the satisfiability of constraints imposed by features.

Outline

Section 2 contains a simple running example used throughout the paper. After a brief description of feature models in Section 2, we discuss how to use deontic logic to characterise them in Section 4. We introduce the behavioural modelling of product families by means of MTSs in Section 5. In Section 6, we define the temporal logic MHML and show that it can be used to express both static and behavioural requirements of product families. We provide a model-checking algorithm for MHML in Section 7 and we sketch how to use it to derive correct products from a family in Section 8. Section 9 concludes the paper.

2 Running Example: Coffee Machine Product Family

To illustrate the contribution of this paper we consider a family of (simplified) coffee machines as running example, with the following list of requirements:

1. The only accepted coins are the one euro coin (1€), exclusively for European products and the one dollar coin (1$), exclusively for Canadian products;
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;
3. The choice of beverage (coffee, tea, cappuccino) varies for the products. However, delivering coffee is a must for all the family's products, while cappuccino is only offered by European products;
4. After delivering the appropriate beverage, optionally, a ringtone is rung. However, a ringtone must be rung whenever a cappuccino is delivered;
5. The machine returns to its idle state when the cup is taken by the user.

This list contains both static requirements, which identify the features that constitute the different products (see requirements 1, 3 and, partially, 4) and behavioural requirements, which describe the admitted sequences of operations (requirements 2, 5 and, partially, 4).

In the sequel, we will first distill the feature model of this family and provide a formal representation of it in terms of deontic logic formulae. We will then show how the behavioural requirements of this family can be described using an MTS. Finally, we will show how to combine the two approaches by defining a deontic logic framework to check the satisfiability of both static and behavioural requirements over products that should belong to this family.

3 Product Families: Feature Diagrams and Feature Models

Feature diagrams were introduced in [22] as a graphical *and*/*or* hierarchy of features; the features are represented as the nodes of a tree, with the product family as its root. Features come in several flavours. In this paper, we consider:

optional features may be present in a product only if their parent is present;

mandatory features are present in a product if and only if their parent is present;

alternative features are a set of features among which one and only one is present in a product if their parent is present.

When additional constraints are added to a feature diagram, one obtains a *feature model*. Also these constraints come in several flavours. In this paper we consider:

requires is a unidirectional relation between two features indicating that the presence of one feature requires the presence of the other;

excludes is a bidirectional relation between two features indicating that the presence of either feature is incompatible with the presence of the other.

An example feature model for the Coffee Machine family of Section 2 is given in Fig. 1; the **requires** constraint obligates feature Ringtone to be present whenever

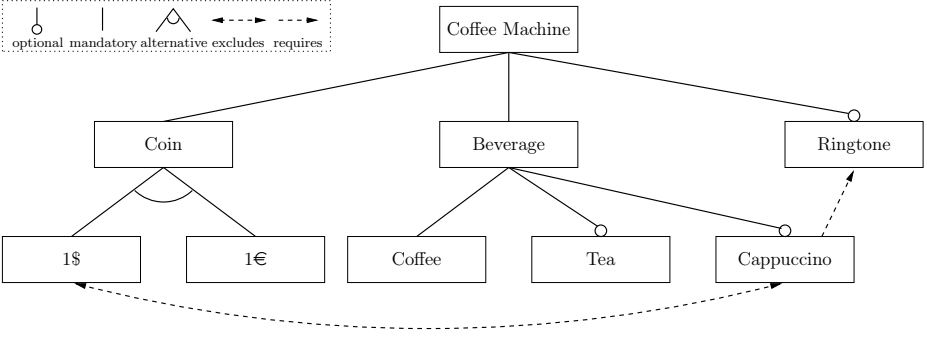


Fig. 1. Feature model of the Coffee Machine family

Cappuccino is and the **excludes** constraint prohibits features 1$ and Cappuccino to both be present in any product of this family. Obviously, this feature model satisfies the static requirements (i.e. 1, 3 and, part of, 4) of our running example.

4 Deontic Logic Applied to Feature Models

Deontic logic has been an active field of research in computer science for many years now [2,29]. Most deontic logics contain the standard operators of classical propositional logic, i.e. negation ($\neg$), conjunction ($\wedge$), disjunction ($\vee$) and implication ($\Longrightarrow$), augmented with deontic operators. In this paper, we consider only two of the most common deontic operators, namely *it is obligatory that* (O) and *it is permitted that* (P), which in the most classical versions of deontic logic enjoy the duality property

$$P(\alpha) = \neg O(\neg\alpha),$$

i.e. something is permitted if and only if its negation is not obligatory.

The way deontic logics formalise concepts such as violation, obligation, permission and prohibition is very useful for system specification, where these concepts arise naturally. In particular, deontic logics seem to be very useful to formalise product family specifications, since they allow one to capture the notions of optional and mandatory features.

4.1 A Deontic Characterisation of Feature Models

In [4], we have presented a deontic characterisation of feature models. Such a characterisation consists of a set of deontic formulae which, taken as a conjunction, precisely characterise the feature model of a product family. If we assume that a name of a feature A is used as the atomic proposition indicating that A is present, then the deontic characterisation is constructed as follows:

- If A is a feature, and A_1 and A_2 are two subfeatures (marked **alternative**, **optional** or **mandatory**), then add the formula $A \implies \Phi(A_1, A_2)$, where $\Phi(A_1, A_2)$ is defined as

  $$\Phi(A_1, A_2) = (O(A_1) \vee O(A_2)) \wedge \neg(P(A_1) \wedge P(A_2))$$

  if A_1 and A_2 are marked **alternative**, whereas $\Phi(A_1, A_2)$ is otherwise defined as

  $$\Phi(A_1, A_2) = \phi(A_1) \wedge \phi(A_2),$$

  in which $\phi(A_i)$, for $i \in \{1, 2\}$, is defined as:

  $$\phi(A_i) = \begin{cases} P(A_i) \text{ if } A_i \text{ is } \textbf{optional} \text{ and} \\ O(A_i) \text{ if } A_i \text{ is } \textbf{mandatory}. \end{cases}$$

  Moreover, since the presence of the root feature is taken for granted, the premise of the implication related to that feature can be removed.[1]

[1] Hence, we tacitly do not deal with trivially inconsistent graphs whose root is involved in an **excludes** relation with a feature.

– If A **requires** B, then add the formula $A \implies O(B)$.
– If A **excludes** B (and hence B **excludes** A), then add the formula $(A \implies \neg P(B)) \wedge (B \implies \neg P(A))$.

This deontic characterisation is a way to provide semantics to feature models. The resulting conjunction of deontic formulae, expressing features and the constraints between them, is called a *characteristic formula* and it can be used to verify whether or not a certain product belongs to a specific family.

5 Behavioural Models for Product Families

In this section we present a behavioural modelling framework able to deal with the variability notions characterising product families at different levels of detail. The underlying model of this framework is a Labelled Transition System (LTS).

Definition 1. *A* Labelled Transition System *(LTS) is a quadruple* $(Q, A, \overline{q}, \rightarrow)$*, where* Q *is a set of states,* A *is a set of actions,* $\overline{q} \in Q$ *is the initial state, and* $\rightarrow \subseteq Q \times A \times Q$ *is the transition relation.*

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

Since we are interested in characterising the dynamic behaviour of product families, we need a notion for the evolution of time in an LTS.

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

A full path *is a path that cannot be extended any further, i.e. which is infinite or ends in a state without outgoing transitions. The set of all full paths from* q *is denoted by path*(q)*.*

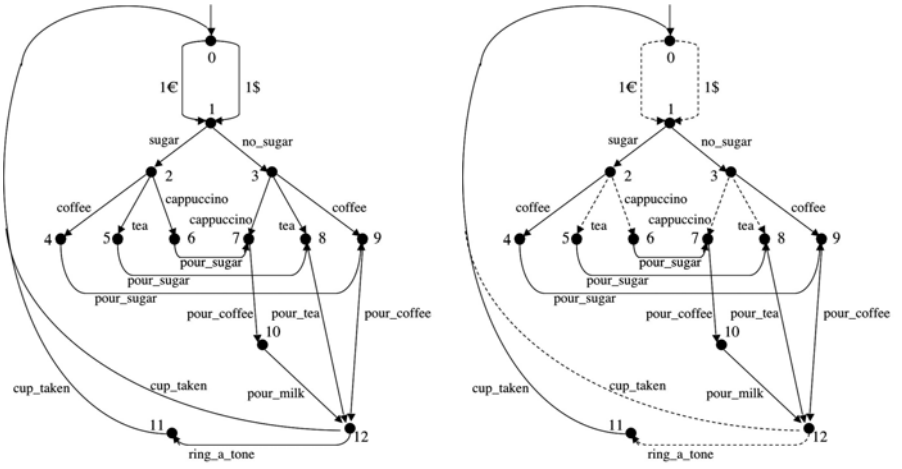
If $\sigma = q_1 a_1 q_2 a_2 q_3 \cdots$*, then its* i*-th state* q_i *is denoted by* $\sigma(i)$ *and its* i*-th action* a_i *is denoted by* $\sigma\{i\}$*.*

When modelling a product family as an LTS, the products of a family are considered to differ with respect to the actions that they are able to perform in any given state. This means that the definition of a family has to accommodate all the possibilities desired for each derivable product, predicating on the choices that make a product belong to the family.

An LTS representing all the possible behaviours conceived for the family of coffee machines described in Section 2 is presented in Fig. 2(a). Note that this LTS cannot distinguish **optional** transitions from **mandatory** ones, since variation points in the family definition are modelled as nondeterministic choices (i.e. alternative paths), independent from the type of variability.

5.1 Modal Transition Systems

To overcome the limitation pointed out earlier of using LTSs as modelling framework for product families, Modal Transition Systems (MTSs) [26] and several variants have been proposed to capture variability in product family specifications [1,16,17,19,25,33].



(a) LTS of the family. (b) MTS of the family.

Fig. 2. (a)-(b) Modelling the family of coffee machines

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

An MTS has two distinct transition relations: $\rightarrow_{\Diamond} \subseteq Q \times A \times Q$ *is the* may *transition relation, which expresses* possible *transitions, while* $\rightarrow_{\Box} \subseteq Q \times A \times Q$ *is the* must *transition relation, which expresses* required *transitions.*

By definition, any required transition is also possible, i.e. $\rightarrow_{\Box} \subseteq \rightarrow_{\Diamond}$.

In an MTS, transitions are either possible (*may*) or required (*must*), corresponding to the notion of **optional** or **mandatory** features in product families. This allows the distinction of a special type of path.

Definition 4. *Let* $(Q, A, \overline{q}, \rightarrow_{\Box}, \rightarrow_{\Diamond})$ *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*) if* $q_i \xrightarrow{a_i}_{\Box} q_{i+1}$*, for all* $i > 0$*, in the MTS.*

The set of all must paths from q_1 *is denoted by* $\Box$*-path*(q_1)*. A must path* σ *is denoted by* $\sigma^{\Box}$.

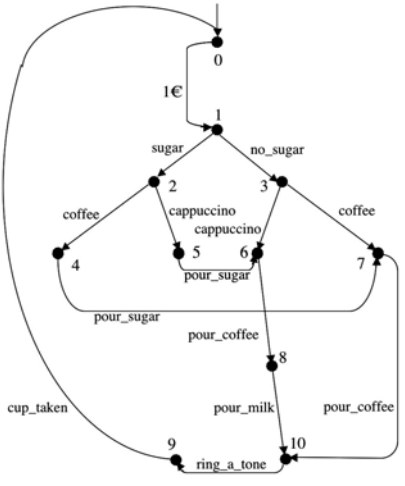
The MTS of Fig. 2(b), in which dashed arcs are used for may transitions and solid ones for must transitions, is another representation of the family of coffee machines described in Section 2. Note that an MTS is able to model the requirements concerning **optional** and **mandatory** characteristics through the use of may and must transitions. However, an MTS is not able to model that the actions 1€ and 1$ are exclusive (i.e. **alternative** features) nor that the action cappuccino cannot be executed in a European product (as results from the **excludes** relation between the features Cappuccino and 1$). This will be more clear later, after we define how to generate correct products from an MTS.

Definition 5. *Given an MTS* $F = (Q, A, \overline{q}, \rightarrow_{\Box}, \rightarrow_{\Diamond})$ *specifying a family, a set of products specified as a set of LTSs* $\{ P_i = (Q_i, A_i, \overline{q}_i, \rightarrow_i) \mid i > 0 \}$ *may be consistently derived by considering the transition relation* $\rightarrow_i$ *to be* $\rightarrow_{\Box} \cup R$, *with* $R \subseteq \rightarrow_{\Diamond}$, *and by pruning all states that are not reachable from* $\overline{q}$.

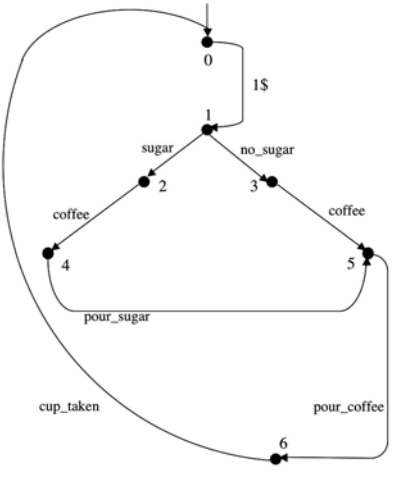
More precisely, we say that P_i is a product of F, *denoted by* $P_i \vdash F$, *if and only if* $\overline{q}_i \vdash \overline{q}$, *where* $q_i \vdash q$ *holds, for some* $q_i \in Q_i$ *and* $q \in Q$, *if and only if:*

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

Following Def. 5, starting from the MTS of Fig. 2(b), two consistent products can be derived: the coffee machines for the European and Canadian markets shown in Fig. 3. Note, however, that also the coffee machine described by the LTS of Fig. 2(a) can be consistently derived from this MTS. This is the demonstration of the fact that MTSs cannot model constraints in feature models regarding **alternative** features and the **excludes** relation. In fact, the product described by the LTS of Fig. 2(a) violates requirements 1 and 3 (cf. Section 2) by allowing the insertion of both 1€ and 1$ and at the same time offering cappuccino.



(a) LTS of a European coffee machine.



(b) LTS of an Canadian coffee machine.

Fig. 3. (a)-(b) Modelling coffee machines for the European and Canadian markets

6 A Logical Framework for Modelling Variability

In [3], we showed how certain MTSs can be completely characterised by deontic logic formulae and in [4] we presented a first attempt at a logical framework able to address both static and behavioural conformance of products of a family.

In this paper, we further develop that work and define a single logical framework in which to express both the evolution in time and the variability notions considered for product families. To this aim, we define the action-based and branching-time temporal logic MHML based on the "Hennessy–Milner logic with Until" defined in [14,24], but we interpret it over MTSs rather than LTSs. This leads to a deontic interpretation of the classical modal and temporal operators.

With respect to [4], we thus consider an action-based logic rather than an action- and state-based logic, and in Section 7 we will moreover provide model-checking algorithms to verify MHML formulae over MTSs.

6.1 Syntax of MHML

MHML extends the classical Hennessy–Milner logic with Until by taking into account the different type of transitions of an MTS and by incorporating the existential and universal state operators (quantifying over paths) from CTL [10]. As such, MHML is derived from the logics defined in [14,23,24] and it is an action-based variant of the logic proposed in [4].

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

Definition 6. *The syntax of MHML is:*

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

The semantics over MTSs makes MHML incorporate deontic interpretations of the classical modalities. In fact, the informal meaning of the nonstandard operators of MHML is as follows:

- $\langle a \rangle\,\phi$: a next state exists, reachable by a *must* transition executing action a, in which ϕ holds
- $[a]\,\phi$: in all next states, reachable by whatever 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
- $\phi\ U\ \phi'$: in the current state, or in a future state of a path, ϕ' holds, while ϕ holds in all preceding states of the path (but not necessarily in that state)
- $\phi\ U^{\Box}\ \phi'$: in the current state, or in a future state of a path, ϕ' holds, while ϕ holds in all preceding states of the path (but not necessarily in that state), and the path leading to that state is a *must* path

6.2 Semantics of MHML

The formal semantics of MHML is given through an interpretation over the MTSs defined in Section 5.1.

Definition 7. *Let $(Q, A, \overline{q}, \rightarrow_\Box, \rightarrow_\Diamond)$ be an MTS, let $q \in Q$ and let σ be a full path. Then the satisfaction relation $\models$ of MHML over MTSs is defined as follows:*

- $q \models \mathit{true}$ *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* $\exists\, q' \in Q : q \xrightarrow{a}_\Box q'$ *and* $q' \models \phi$
- $q \models [a]\, \phi$ *iff* $\forall\, q' \in Q : q \xrightarrow{a}_\Diamond q'$ *and* $q' \models \phi$
- $q \models E\,\pi$ *iff* $\exists\, \sigma' \in \mathit{path}(q) : \sigma' \models \pi$
- $q \models A\,\pi$ *iff* $\forall\, \sigma' \in \mathit{path}(q) : \sigma' \models \pi$
- $\sigma \models [\phi\ U\ \phi']$ *iff* $\exists\, j \geq 1 : \sigma(j) \models \phi'$ *and* $\forall\, 1 \leq i < j : \sigma(i) \models \phi$
- $\sigma \models [\phi\ U^\Box\ \phi']$ *iff* $\exists\, j \geq 1 : \sigma^\Box(j) \models \phi'$ *and* $\forall\, 1 \leq i < j : \sigma^\Box(i) \models \phi$

The classical duality rule of Hennessy–Milner logic, which states that $\langle a \rangle \phi$ abbreviates $\neg[a]\neg\phi$, 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 now be derived in the usual way: *false* abbreviates $\neg\,\mathit{true}$, $\phi \vee \phi'$ abbreviates $\neg(\neg\phi \wedge \neg\phi')$, $\phi \Longrightarrow \phi'$ abbreviates $\neg\phi \vee \phi'$. Moreover, $F\,\phi$ abbreviates $(\mathit{true}\ U\ \phi)$: there exists a future state in which ϕ holds. Likewise, $F^\Box\,\phi$ abbreviates $(\mathit{true}\ U^\Box\ \phi)$: there exists a future state of a must path in which ϕ holds. Finally, $AG\,\phi$ abbreviates $\neg EF\,\neg\phi$: in every state on every path, ϕ holds; and $AG^\Box\,\phi$ abbreviates $\neg EF^\Box\,\neg\phi$: in every state on every must path, ϕ holds.

An illustrative example of a well-formed formula in MHML is thus

$$[a]\,(P(b)\,\mathit{true} \wedge (\phi \implies \langle c \rangle\,\mathit{true})),$$

which states that after the execution of action a, the system is in a state in which executing action b is *permitted* (in the sense of a *may* transition) and, moreover, whenever formula ϕ holds, then executing action c is *obligatory* (in the sense of a *must* transition). Note that by defining the semantics of MHML over MTSs, we have indeed given a deontic interpretation to the classical box and diamond modalities of Hennessy–Milner logic. In fact, MHML can express both *permitted* and *obligatory* actions (features).

We could of course extend the semantics of MHML formulae to an interpretation over LTSs rather than over MTSs. In that case, since LTSs consist of only *must* transitions, all modalities would need to be interpreted as in the classical Hennessy–Milner logic; this would mean that the weaker version of the diamond operator $P(a)\,\phi$ in MHML would collapse onto the classical diamond operator $\langle a \rangle\,\phi$ of Hennessy–Milner logic.

6.3 Expressing Static *and* Behavioural Requirements

MHML is able to complement the behavioural description of an MTS by expressing constraints over possible products of a family, modelling in this way the static requirements that cannot be expressed in an MTS.

To begin with we consider the following formalisations of the static requirements 1 and 3 (cf. Section 2).

Property A. The actions of inserting 1€ or 1$ are **alternative** features:

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

Property B. The action cappuccino cannot be executed in Canadian coffee machines (**excludes** relation between features):

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

These formulae combine static requirements represented by the pure deontic formulae of Section 4.1, through their deontic interpretation in MHML, with behavioural relations among actions expressible by the temporal part of MHML.

Recall that the deontic obligation operator is mapped onto MHML's diamond modality and the deontic permission operator is represented by MHML's weaker version of the classical diamond modality. It is important to note, however, that the classical duality property among the O and P operators of deontic logic is not preserved by this mapping.

To continue, we consider the following formalisation of the static part of requirement 4 (cf. Section 2).

Property C. A ringtone must be rung whenever a cappuccino is delivered:

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

This is an example of a **requires** relation between features. Note that such a static relation between features does not imply any ordering among the related features; e.g., a coffee machine that performs a ringtone before producing a cappuccino cannot be excluded as a 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 orderings.

Subsequently, we consider the following formalisation of a further requirement that is particularly interesting for the user of a coffee machine:

Property D. Once the user has selected a coffee, a coffee is eventually delivered:

$$AG\,[coffee]\,AF^{\Box}\,\langle pour\_\, coffee\rangle\, true$$

7 Model-Checking Algorithms for MHML

The problem model checking aims to solve can be stated as: Given a desired property, expressed as a formula ψ in a certain logic, and a model M, in the form of a transition system, one wants to decide whether $M \models \psi$ holds, where

$\models$ is the logic's satisfaction relation. If $M \not\models \psi$, then it is usually easy to generate a counterexample. If M is finite, model checking reduces to a graph search.

Based on the model-checking parameters M and ψ, different strategies can be pursued when designing a model-checking algorithm. The following global model-checking algorithm extends classical algorithms for the Hennessy–Milner logic and for CTL to MHML [10,11,31]. Actually, the only variation is the distinction of the transition relation ($\rightarrow_\Diamond$ or $\rightarrow_\Box$) used in the different cases.

Algorithm 1. A global model-checking algorithm for MHML.

for all $q \in Q$ **do** $L(q) := \{true\}$
for $i = 1$ **to** *length*(ψ) **do**
  for all subformulae ϕ of ψ **such that** *length*(ϕ) $= i$ **do**
    if $\phi = true$ **then**
      {nothing to do}
    else if $\phi = \neg\,\phi_1$ **then**
      for all $q \in Q$ **do**
        if $\phi_1 \notin L(q)$ **then**
          $L(q) := L(q) \cup \{\phi\}$
    else if $\phi = \phi_1 \wedge \phi_2$ **then**
      for all $q \in Q$ **do**
        if $\phi_1 \in L(q)$ **and** $\phi_2 \in L(q)$ **then**
          $L(q) := L(q) \cup \{\phi\}$
    else if $\phi = [a]\phi_1$ **then**
      for all $q \in Q$ **do**
        if $\forall\, q' \colon q \xrightarrow{a}_\Diamond q'$, $\phi_1 \in L(q')$ **then**
          $L(q) := L(q) \cup \{\phi\}$
    else if $\phi = \langle a \rangle \phi_1$ **then**
      for all $q \in Q$ **do**
        if $\exists\, q' \colon q \xrightarrow{a}_\Box q'$, $\phi_1 \in L(q')$ **then**
          $L(q) := L(q) \cup \{\phi\}$
    else if $\phi = P(a)\,\phi_1$ **then**
      for all $q \in Q$ **do**
        if $\exists\, q' \colon q \xrightarrow{a}_\Diamond q'$, $\phi_1 \in L(q')$ **then**
          $L(q) := L(q) \cup \{\phi\}$
    else if $\phi = E\,(\phi_1\ U^\Box\ \phi_2)$ **then**
      $T := \{\, q \mid \phi_2 \in L(q) \,\}$
      for all $q \in T$ **do**
        $L(q) := L(q) \cup \{E\,(\phi_1\ U^\Box\ \phi_2)\}$
        while $T \neq \varnothing$ **do**
          choose $q \in T$
          $T := T \setminus \{q\}$
          for all p **such that** $p \rightarrow_\Box q$ **do**
            if $E(\phi_1\, U^\Box \phi_2) \notin L(p)$ **and** $\phi_1 \in L(p)$ **then**
              $L(p) := L(p) \cup \{E\,(\phi_1\ U^\Box\ \phi_2)\}$
              $T := T \cup \{p\}$

Algorithm 1 stores in $L(q)$ all subformulae of ψ that are true in q, initially associating *true* to every state and then evaluating subformulae of increasing size on the MTS. Evaluating Until formulae requires another visit of the MTS. The algorithm's complexity is $O(|\psi| \times |Q| \times (|Q| + |\rightarrow_\Diamond|))$. A more efficient version uses for Until a depth-first search: its complexity is linear w.r.t. the state space size.

Note that we consider only one of the existential Until operators; other combinations of existential/universal quantification and Until operators can be dealt with similarly. In particular, the procedure for the classical Until operator U can be obtained from that for $U^\Box$ by allowing a may transition in its inner for-loop.

Verifying properties A–D on Figs. 2–3 with Algorithm 1 leads to Table 1.

Finally, note the potential for inconsistency: An MTS of a family might not allow any products that satisfy all constraints on features expressed by MHML formulae, i.e. all MHML formulae complementing the behavioural description of an MTS would be false in that MTS. This clearly advocates the usefulness of our model-checking algorithm on MTSs.

Table 1. Results of verifying properties A–D on Figs. 2–3 with Algorithm 1

Property	Fig. 2(a)	Fig. 2(b)	Fig. 3(a)	Fig. 3(b)
A	*false*	*false*	*true*	*true*
B	*false*	*true*	*true*	*true*
C	*false*	*true*	*true*	*true*
D	*true*	*true*	*true*	*true*

8 Towards the Derivation of Correct Products from a Family Description

In Section 5.1, we sketched an algorithm for deriving LTS descriptions of *correct* products from the MTS description of a product family. These products are correct in the sense that they respect the family's requirements as modelled by the MTS, such as, e.g., the presence of features that are **optional** or **mandatory** but also their behavioural ordering in time. We subsequently presented a relation between LTSs and MTSs to formalise when an LTS *is a product of* a family (specified as an MTS).

In [19], the authors present an algorithm for checking conformance of LTS models against MTS ones according to a given branching relation, i.e. for checking conformance of the behaviour of a product against its product family's behaviour. It is a fixed-point algorithm that starts with the Cartesian product of the states and iteratively eliminates those pairs that are not valid according to the given relation. They have implemented their algorithm in a tool that allows one to check whether a given LTS conforms to a given MTS according to a number of different branching relations.

Both algorithms allow the derivation of products (specified as LTSs) that are *correct with respect to* the MTS model of a family of products. As we have seen in the previous section, this means that these products might be *incorrect with respect to* the static constraints that cannot be expressed in MTSs, such as, e.g., the presence of features that are **alternative** or part of an **excludes** relation. Since these constraints can be formulated in MHML, we envision an algorithm for deriving correct LTS descriptions of products from an MTS description of a product family *and* an associated set of MHML formulae expressing further constraints for this family. The idea is to prune optional (may) transitions in the MTS in a counterexample-guided way, i.e. based on model-checking techniques.

We informally present our idea by considering as example Property A of Section 6.3, i.e. 1€ and 1$ are **alternative** features:

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

Model checking this formula over the MTS of Fig. 2(b) returns as counterexample two paths through this MTS, one starting with the 1$ action and one starting with the 1€ action. Both these actions are optional, which means that two correct products (cf. Fig. 3) can be derived from this MTS: one by pruning the 1$ action

and one by pruning the 1∈ action instead. At this point, model checking should be repeated in order to see whether other counterexamples remain (which is not the case for our example).

Based on the principle illustrated by means of this example, an algorithm can be devised in which the conjunction of the constraints is repeatedly model checked, first over the MTS description of the product family and consequently over the resulting (set of) pruned MTSs. These intermediate MTSs are obtained by pruning may transitions in a counterexample-guided way until the formula (conjunction of constraints) is found to be true.

The precise definition of such an algorithm is left for future work, and requires a careful study of the different possible types of constraints and of the effectiveness of the counterexample-guided pruning. After an initial analysis, it seems that local model checking is a more effective strategy to base such an algorithm on, due to its ability to generate counterexample paths early on, without the need of an extensive exploration of the state space. The resulting algorithm would thus allow one to generate all LTSs that satisfy both the family's requirements as modelled by the MTS and its associated set of MHML formulae, i.e. all products that are *correct with respect to* the MTS model of a family of products.

9 Conclusions and Future Work

In this paper we have continued the line of research we initiated in [3,4] with the following contributions:

1. We have introduced the action-based branching-time temporal logic MHML, which allows the expression of both constraints over the products of a family and constraints over their behaviour in a single logical framework.
2. We have set a first step towards a modelling and verification framework based on model-checking techniques for MHML, by providing a model-checking algorithm to verify MHML formulae over MTSs and by sketching a way to derive correct products from a family description. Both an analysis of the complexity of the model-checking algorithm for MHML and the actual implementation of a model-checking environment for MHML are left as future work.

Such an actual implementation of a model-checking environment for MHML could be an extension of existing model-checking tools, like UMC [6,7,28] or MTSA [15]. UMC is an on-the-fly model checker for UCTL (UML-oriented CTL) formulae over a set of communicating UML state machines. MTSA, built on top of the LTS Analyser LTSA, is a tool that supports the construction, analysis and elaboration of MTSs. To this aim, we can make use of the fact that model checking MTSs is not more complex than model checking LTSs, as checking an MTS can be reduced to two times checking an LTS [19].

The added value of the logical framework we have introduced in this paper can thus be summarised as allowing the possibility to reason, in a single framework, on static *and* dynamic aspects of products of a family. Moreover, from a theoretical point of view, we provide a novel *deontic* interpretation of the classical modal and temporal operators.

Finally, there are a number of aspects of our line of research that require a deeper understanding:

- how to identify classes of properties that, once proved over family descriptions, are preserved by all products of the family;
- how to evaluate quantitative properties, like the number of different possible products of a family;
- how the approach scales to real-world situations in PLE;
- how to hide the complexity of the proposed modelling and verification framework from end users.

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 edn., vol. 8, pp. 147–264. Kluwer, Dordrecht (2002)
3. Asirelli, P., ter Beek, M.H., Fantechi, A., Gnesi, S.: Deontic Logics for modelling Behavioural Variability. In: Benavides, D., Metzger, A., Eisenecker, U. (eds.) Proceedings Variability Modelling of Software-intensive Systems (VaMoS 2009). ICB Research Report, vol. 29, pp. 71–76. Universität 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 Variability Modelling of Software-intensive Systems (VaMoS 2010). ICB Research Report, vol. 37, pp. 37–44. Universität Duisburg, Essen (2010)
5. Batory, D.: Feature Models, Grammars and Propositional Formulas. In: Obbink, H., Pohl, K. (eds.) SPLC 2005. LNCS, vol. 3714, pp. 7–20. Springer, Heidelberg (2005)
6. ter Beek, M.H., Fantechi, A., Gnesi, S., Mazzanti, F.: An action/state-based model-checking approach for the analysis of communication protocols for service-oriented applications. In: Leue, S., Merino, P. (eds.) FMICS 2007. LNCS, vol. 4916, pp. 133–148. Springer, Heidelberg (2008)
7. ter Beek, M.H., Fantechi, A., Gnesi, S., Mazzanti, F.: A state/event-based model-checking approach for the analysis of abstract system properties. Sci. Comput. Program. (to appear 2010)
8. Castro, P.F., Maibaum, T.S.E.: A Complete and Compact Propositional Deontic Logic. In: Jones, C.B., Liu, Z., Woodcock, J. (eds.) ICTAC 2007. LNCS, vol. 4711, pp. 109–123. Springer, Heidelberg (2007)
9. Castro, P.F., Maibaum, T.S.E.: A Tableaux System for Deontic Action Logic. In: van der Meyden, R., van der Torre, L. (eds.) DEON 2008. LNCS (LNAI), vol. 5076, pp. 34–48. Springer, Heidelberg (2008)
10. 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)
11. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT, Cambridge (1999)
12. Classen, A., Heymans, P., Schobbens, P.-Y., Legay, A., Raskin, J.-F.: Model Checking Lots of Systems: Efficient Verification of Temporal Properties in Software Product Lines. In: Proceedings 32nd ACM/IEEE International Conference on Software Engineering, vol. 1, pp. 335–344. ACM, New York (2010)
13. Czarnecki, K., Eisenecker, U.W.: Generative Programming: Methods, Tools, and Applications. Addison-Wesley, Boston (2000)
14. De Nicola, R., Vaandrager, F.W.: Three Logics for Branching Bisimulation. J. ACM 42(2), 458–487 (1995)

15. D'Ippolito, N., Fischbein, D., Chechik, M., Uchitel, S.: MTSA: The Modal Transition System Analyser. In: Proceedings 23rd IEEE/ACM International Conference on Automated Software Engineering, pp. 475–476. IEEE, Washington (2008)
16. Fantechi, A., Gnesi, S.: A Behavioural Model for Product Families. In: Crnkovic, I., Bertolino, A. (eds.) Proceedings 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT symposium on Foundations of Software Engineering, pp. 521–524. ACM, New York (2007)
17. Fantechi, A., Gnesi, S.: Formal modelling for Product Families Engineering. In: Proceedings 12th International Software Product Line Conference, pp. 193–202. IEEE, Washington (2008)
18. 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.) SPLC 2004. LNCS, vol. 3154, pp. 255–265. Springer, Heidelberg (2004)
19. Fischbein, D., Uchitel, S., Braberman, V.A.: A Foundation for Behavioural Conformance in Software Product Line Architectures. In: Hierons, R.M., Muccini, H. (eds.) Proceedings ISSTA 2006 Workshop on Role of Software Architecture for Testing and Analysis, pp. 39–48. ACM, New York (2006)
20. Gruler, A., Leucker, M., Scheidemann, K.D.: Modelling and Model Checking Software Product Lines. In: Barthe, G., de Boer, F.S. (eds.) FMOODS 2008. LNCS, vol. 5051, pp. 113–131. Springer, Heidelberg (2008)
21. Gruler, A., Leucker, M., Scheidemann, K.D.: Calculating and Modelling Common Parts of Software Product Lines. In: Proceedings 12th International Software Product Line Conference, pp. 203–212. IEEE, Washington (2008)
22. Kang, K., Choen, S., Hess, J., Novak, W., Peterson, S.: Feature Oriented Domain Analysis (FODA) Feasibility Study. Technical Report SEI-90-TR-21, Carnegie Mellon University (1990)
23. Larsen, K.G.: Modal Specifications. In: Sifakis, J. (ed.) Automatic Verification Methods for Finite State Systems. LNCS, vol. 407, pp. 232–246. Springer, Heidelberg (1989)
24. Larsen, K.G.: Proof Systems for Satisfiability in Hennessy-Milner Logic with Recursion. Theor. Comput. Sci. 72(2-3), 265–288 (1990)
25. Larsen, K.G., Nyman, U., Wąsowski, A.: Modal I/O Automata for Interface and Product Line Theories. In: De Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 64–79. Springer, Heidelberg (2007)
26. Larsen, K.G., Thomsen, B.: A Modal Process Logic. In: Proceedings 3rd Annual Symposium on Logic in Computer Science, pp. 203–210. IEEE, Washington (1988)
27. Mannion, M., Camara, J.: Theorem Proving for Product Line Model Verification. In: van der Linden, F.J. (ed.) PFE 2003. LNCS, vol. 3014, pp. 211–224. Springer, Heidelberg (2004)
28. Mazzanti, F.: UMC model checker v3.6 (2009), `http://fmt.isti.cnr.it/umc`
29. Meyer, J.-J.C., Wieringa, R.J. (eds.): Deontic Logic in Computer Science: Normative System Specification. John Wiley & Sons, Chichester (1994)
30. Meyer, M.H., Lehnerd, A.P.: The Power of Product Platforms: Building Value and Cost Leadership. The Free Press, New York (1997)
31. Müller-Olm, M., Schmidt, D.A., Steffen, B.: Model-Checking: A Tutorial Introduction. In: Cortesi, A., Filé, G. (eds.) SAS 1999. LNCS, vol. 1694, pp. 330–354. Springer, Heidelberg (1999)
32. Pohl, K., Böckle, G., van der Linden, F.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer, Heidelberg (2005)
33. Schmidt, H., Fecher, H.: Comparing disjunctive modal transition systems with an one-selecting variant. J. Logic Algebraic Program. 77(1-2), 20–39 (2008)