A logical framework to deal with variability
(research in progress)

M.H. ter Beek
joint work with
P. Asirelli, A. Fantechi and S. Gnesi

ISTI–CNR
Università di Firenze

XXL project meeting

Pisa, 21 June 2010

Outline

Product Line Engineering (PLE)

Paradigm

To develop a family of products using a common platform and mass customization

Aim

To lower production costs of the individual products by

- letting them share an overall reference model of the product family
- allowing them to differ w.r.t. particular characteristics to serve, e.g., different markets

Production process

Organized so as to maximize commonalities of the products and at the same time minimize the cost of variations

Product Line Engineering (PLE)

Paradigm

To develop a family of products using a common platform and mass customization

Aim

To lower production costs of the individual products by

- letting them share an overall reference model of the product family
- allowing them to differ w.r.t. particular characteristics to serve, e.g., different markets

Production process

Organized so as to maximize commonalities of the products and at the same time minimize the cost of variations

Product Line Engineering (PLE)

Paradigm

To develop a family of products using a common platform and mass customization

Aim

To lower production costs of the individual products by

- letting them share an overall reference model of the product family
- allowing them to differ w.r.t. particular characteristics to serve, e.g., different markets

Production process

Organized so as to maximize commonalities of the products and at the same time minimize the cost of variations

Variability in PLE

Feature modelling

Provide compact representations of all the products of a product family (product line) in terms of their features

Variability modelling

How to explicitly define the features or components of a product family that are **optional**, **alternative**, or **mandatory**

Managing variability with formal methods

Show that a certain product belongs to a product family or, instead, derive a product from a family by means of a proper selection of the features or components

Variability in PLE

Feature modelling

Provide compact representations of all the products of a product family (product line) in terms of their features

Variability modelling

How to explicitly define the features or components of a product family that are **optional**, **alternative**, or **mandatory**

Managing variability with formal methods

Show that a certain product belongs to a product family or, instead, derive a product from a family by means of a proper selection of the features or components

Variability in PLE

Feature modelling

Provide compact representations of all the products of a product family (product line) in terms of their features

Variability modelling

How to explicitly define the features or components of a product family that are **optional**, **alternative**, or **mandatory**

Managing variability with formal methods

Show that a certain product belongs to a product family or, instead, derive a product from a family by means of a proper selection of the features or components

Aim of our research activity

- To develop a logical framework that is able to deal with variability
- To provide tools to support this framework with formal verification

Current contributions

- We present a straightforward characterization of feature models by means of a deontic logic
- We define the action-based branching-time temporal logic MHML, allows expressing both constraints over the products of a family and constraints over their behaviour in a single logical framework
- We define the semantics of MHML over MTSs, leading to a novel deontic interpretation of classical modal and temporal operators
- We provide a global model-checking algorithm to verify MHML formulae over MTSs, i.e., a first step towards a verification framework based on model-checking techniques for MHML

Aim of our research activity

- To develop a logical framework that is able to deal with variability
- To provide tools to support this framework with formal verification

Current contributions

- We present a straightforward characterization of feature models by means of a deontic logic
- We define the action-based branching-time temporal logic MHML, allows expressing both constraints over the products of a family and constraints over their behaviour in a single logical framework
- We define the semantics of MHML over MTSs, leading to a novel deontic interpretation of classical modal and temporal operators
- We provide a global model-checking algorithm to verify MHML formulae over MTSs, i.e., a first step towards a verification framework based on model-checking techniques for MHML

Static & behavioural requirements of product families

Static requirements identify the **features** constituting different products and *behavioural requirements* the **admitted sequences of operations**

Static requirements of product families

- The only accepted coins are the 1 euro coin (1€), exclusively for the European products and the 1 dollar coin (1$), exclusively for the US products (1€ and 1$ are exclusive (**alternative**) features)
- A cappuccino is only offered by European products (**excludes** relation between features)

Behavioural requirements of product families

- After inserting a coin, the user has to choose whether or not (s)he wants sugar, by pressing one of two buttons, after which (s)he may select a beverage
- The machine returns to its idle state when the beverage is taken

Static & behavioural requirements of product families

Static requirements identify the **features** constituting different products and *behavioural requirements* the **admitted sequences of operations**

Static requirements of product families

- The only accepted coins are the 1 euro coin (1€), exclusively for the European products and the 1 dollar coin (1$), exclusively for the US products (1€ and 1$ are exclusive (**alternative**) features)
- A cappuccino is only offered by European products (**excludes** relation between features)

Behavioural requirements of product families

- After inserting a coin, the user has to choose whether or not (s)he wants sugar, by pressing one of two buttons, after which (s)he may select a beverage
- The machine returns to its idle state when the beverage is taken

Static & behavioural requirements of product families

Static requirements identify the **features** constituting different products and *behavioural requirements* the **admitted sequences of operations**

Static requirements of product families

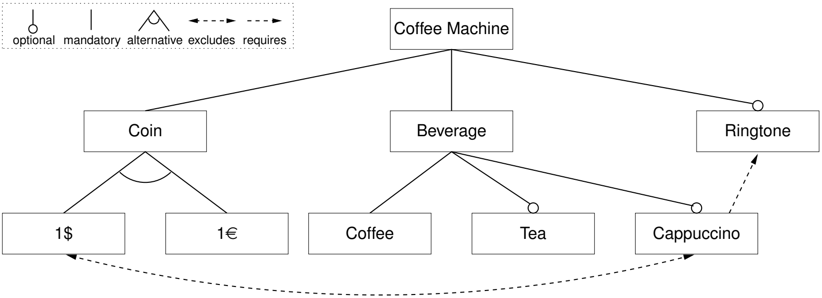
- The only accepted coins are the 1 euro coin (1€), exclusively for the European products and the 1 dollar coin (1$), exclusively for the US products (1€ and 1$ are exclusive (**alternative**) features)
- A cappuccino is only offered by European products (**excludes** relation between features)

Behavioural requirements of product families

- After inserting a coin, the user has to choose whether or not (s)he wants sugar, by pressing one of two buttons, after which (s)he may select a beverage
- The machine returns to its idle state when the beverage is taken

Running example: Coffee machine family

Feature model:



Deontic logic

- Deontic logic provides a natural way to formalize concepts like violation, obligation, permission and prohibition
- Deontic logic seems to be very useful to formalize product families specifications, since they allow one to capture the notions of **optional**, **mandatory** and **alternative** features
- Deontic logic seems to be very useful to formalize feature constraints such as **requires** and **excludes**.

$\Rightarrow$ Deontic logic seems to be a natural candidate for expressing the conformance of products with respect to variability rules

Deontic logic

- Deontic logic provides a natural way to formalize concepts like violation, obligation, permission and prohibition
- Deontic logic seems to be very useful to formalize product families specifications, since they allow one to capture the notions of **optional**, **mandatory** and **alternative** features
- Deontic logic seems to be very useful to formalize feature constraints such as **requires** and **excludes**.

$\Rightarrow$ Deontic logic seems to be a natural candidate for expressing the conformance of products with respect to variability rules

Deontic logic – continued

A deontic logic consists of the standard operators of propositional logic, i.e. negation ($\neg$), conjunction ($\wedge$), disjunction ($\vee$) and implication ($\Longrightarrow$), augmented with deontic operators (O and P in our case)

The most classic deontic operators, namely *it is obligatory that* (O) and *it is permitted that* (P) enjoy the duality property

Informal meaning of the deontic operators

- $O(a)$: action a is *obligatory*
- $P(a) = \neg O(\neg a)$: action a is *permitted*
  if and only if its negation is not obligatory

Deontic logic – continued

A deontic logic consists of the standard operators of propositional logic, i.e. negation ($\neg$), conjunction ($\wedge$), disjunction ($\vee$) and implication ($\Longrightarrow$), augmented with deontic operators (O and P in our case)

The most classic deontic operators, namely *it is obligatory that* (O) and *it is permitted that* (P) enjoy the duality property

Informal meaning of the deontic operators

- $O(a)$: action a is *obligatory*
- $P(a) = \neg O(\neg a)$: action a is *permitted*
  if and only if its negation is not obligatory

Construction of deontic characterization of FM

- If A is a feature and A_1 and A_2 are subfeatures, add the formula:

$$A \implies \Phi(A_1, A_2), \quad \text{where } \Phi(A_1, A_2) \text{ is defined as:}$$

$\Phi(A_1, A_2) = (O(A_1) \vee O(A_2)) \wedge \neg(P(A_1) \wedge P(A_2))$ if A_1, A_2 **alternative**, and otherwise:

$\Phi(A_1, A_2) = \phi(A_1) \wedge \phi(A_2)$, in which A_i, for $i \in \{1, 2\}$, is defined as:

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

- If A **requires** B, add the formula $A \implies O(B)$

- If A **excludes** B, add the formula $(A \implies \neg P(B)) \wedge (B \implies \neg P(A))$

Construction of deontic characterization of FM

- If A is a feature and A_1 and A_2 are subfeatures, add the formula:

$$A \implies \Phi(A_1, A_2), \quad \text{where } \Phi(A_1, A_2) \text{ is defined as:}$$

$\Phi(A_1, A_2) = (O(A_1) \vee O(A_2)) \wedge \neg(P(A_1) \wedge P(A_2))$ if A_1, A_2 **alternative**, and otherwise:

$\Phi(A_1, A_2) = \phi(A_1) \wedge \phi(A_2)$, in which A_i, for $i \in \{1, 2\}$, is defined as:

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

- If A **requires** B, add the formula $A \implies O(B)$

- If A **excludes** B, add the formula $(A \implies \neg P(B)) \wedge (B \implies \neg P(A))$

Construction of deontic characterization of FM

- If A is a feature and A_1 and A_2 are subfeatures, add the formula:

$$A \implies \Phi(A_1, A_2), \quad \text{where } \Phi(A_1, A_2) \text{ is defined as:}$$

$\Phi(A_1, A_2) = (O(A_1) \vee O(A_2)) \wedge \neg(P(A_1) \wedge P(A_2))$ if A_1, A_2 **alternative**, and otherwise:

$\Phi(A_1, A_2) = \phi(A_1) \wedge \phi(A_2)$, in which A_i, for $i \in \{1, 2\}$, is defined as:

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

- If A **requires** B, add the formula $A \implies O(B)$

- If A **excludes** B, add the formula $(A \implies \neg P(B)) \wedge (B \implies \neg P(A))$

Expressing feature models with deontic logic

Characteristic formula of Coffee machine family

$$O(\text{Coin}) \wedge O(\text{Beverage}) \wedge P(\text{Ringtone})$$

$$\wedge$$

$$\text{Coin} \implies (O(1\$) \vee O(1€)) \wedge \neg(P(1\$) \wedge P(1€))$$

$$\text{Beverage} \implies O(\text{Coffee}) \wedge P(\text{Tea}) \wedge P(\text{Cappuccino})$$

$$\wedge$$

$$\text{Cappuccino} \implies O(\text{Ringtone})$$

$$(1\$ \implies \neg P(\text{Cappuccino})) \wedge (\text{Cappuccino} \implies \neg P(1\$))$$

Two example coffee machines

$$\text{CM1} = \{\text{Coin}, 1€, \text{Beverage}, \text{Coffee}\}$$

$$\text{CM2} = \{\text{Coin}, 1€, \text{Beverage}, \text{Coffee}, \text{Cappuccino}\}$$

CM1 in family, but CM2 not: Cappuccino $\implies$ $O(\text{Ringtone})$ false

Expressing feature models with deontic logic

Characteristic formula of Coffee machine family

$$
\begin{array}{l}
O(\text{Coin}) \wedge O(\text{Beverage}) \wedge P(\text{Ringtone}) \\
\wedge \\
\text{Coin} \implies (O(1\$) \vee O(1€)) \wedge \neg(P(1\$) \wedge P(1€)) \\
\text{Beverage} \implies O(\text{Coffee}) \wedge P(\text{Tea}) \wedge P(\text{Cappuccino}) \\
\wedge \\
\text{Cappuccino} \implies O(\text{Ringtone}) \\
(1\$ \implies \neg P(\text{Cappuccino})) \wedge (\text{Cappuccino} \implies \neg P(1\$))
\end{array}
$$

Two example coffee machines

$$
\begin{array}{lcl}
\text{CM1} & = & \{\text{Coin}, 1€, \text{Beverage}, \text{Coffee}\} \\
\text{CM2} & = & \{\text{Coin}, 1€, \text{Beverage}, \text{Coffee}, \text{Cappuccino}\}
\end{array}
$$

CM1 in family, but CM2 not: Cappuccino $\implies O(\text{Ringtone})$ false

Expressing feature models with deontic logic

Characteristic formula of Coffee machine family

$$
\begin{aligned}
& O(\text{Coin}) \wedge O(\text{Beverage}) \wedge P(\text{Ringtone}) \\
& \wedge \\
& \text{Coin} \implies (O(1\$) \vee O(1€)) \wedge \neg(P(1\$) \wedge P(1€)) \\
& \text{Beverage} \implies O(\text{Coffee}) \wedge P(\text{Tea}) \wedge P(\text{Cappuccino}) \\
& \wedge \\
& \text{Cappuccino} \implies O(\text{Ringtone}) \\
& (1\$ \implies \neg P(\text{Cappuccino})) \wedge (\text{Cappuccino} \implies \neg P(1\$))
\end{aligned}
$$

Two example coffee machines

$$
\begin{aligned}
\text{CM1} &= \{\text{Coin}, 1€, \text{Beverage}, \text{Coffee}\} \\
\text{CM2} &= \{\text{Coin}, 1€, \text{Beverage}, \text{Coffee}, \text{Cappuccino}\}
\end{aligned}
$$

CM1 in family, but CM2 not: Cappuccino $\implies$ O(Ringtone) false

Modal Transition System (MTS)

LTS

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 $q \xrightarrow{a} q'$. A *full path* is a path that cannot be extended any further.

MTS

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. $\rightarrow_{\Box} \subseteq \rightarrow_{\Diamond}$. A *must path* is a path of only must transitions.

Modal Transition System (MTS)

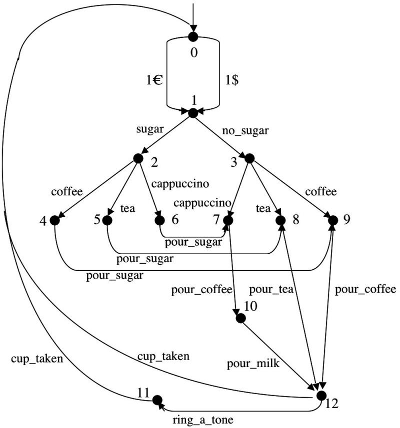
LTS

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 $q \xrightarrow{a} q'$. A *full path* is a path that cannot be extended any further.

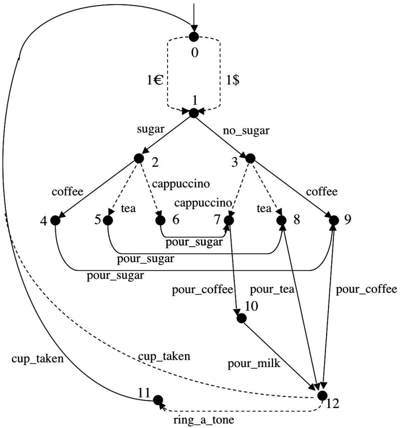
MTS

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. $\rightarrow_{\Box} \subseteq \rightarrow_{\Diamond}$. A *must path* is a path of only must transitions.

Modelling the family of coffee machines



(a) LTS modelling the family



(b) MTS modelling the family

MTS can thus model **optional** and **mandatory** features, but neither **alternative** nor **excludes** features

MHML: Hennesy–Milner Logic with Until

- Allows expressing both constraints over the products of a family and constraints over their behaviour in a single logical framework
- Interpreted over MTSs rather than LTSs, leading to **novel deontic interpretation** of the classical modal and temporal operators

Syntax of MHML

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

Informal meaning of nonstandard operators

- $\langle a \rangle\, \phi$: a next state exists, reachable by **must** transition executing a, where ϕ holds
- $[\alpha]\, \phi$: in all next states, reachable by whatever transition executing a, ϕ holds
- $\phi\; U\, \phi'$: in the current or in a future state of a path, ϕ' holds, while ϕ holds in all preceding states of the path
- $\phi\; U^{\Box}\, \phi'$: in the current or in a future state of a path, ϕ' holds, while ϕ holds in all preceding states of the path, and the path leading to that state is a **must** path

MHML: Hennesy–Milner Logic with Until

- Allows expressing both constraints over the products of a family and constraints over their behaviour in a single logical framework
- Interpreted over MTSs rather than LTSs, leading to **novel deontic interpretation** of the classical modal and temporal operators

Syntax of MHML

$$\begin{array}{lcl} \phi & ::= & \mathit{true} \mid \neg\,\phi \mid \phi \wedge \phi' \mid \langle a \rangle\,\phi \mid [a]\,\phi \mid E\,\pi \mid A\,\pi \\ \pi & ::= & \phi\; U\, \phi' \mid \phi\; U^{\Box}\, \phi' \end{array}$$

Informal meaning of nonstandard operators

- $\langle a \rangle\,\phi$: a next state exists, reachable by **must** transition executing a, where ϕ holds
- $[\alpha]\,\phi$: in all next states, reachable by whatever transition executing a, ϕ holds
- $\phi\; U\, \phi'$: in the current or in a future state of a path, ϕ' holds, while ϕ holds in all preceding states of the path
- $\phi\; U^{\Box}\, \phi'$: in the current or in a future state of a path, ϕ' holds, while ϕ holds in all preceding states of the path, and the path leading to that state is a **must** path

MHML: Hennesy–Milner Logic with Until

- Allows expressing both constraints over the products of a family and constraints over their behaviour in a single logical framework
- Interpreted over MTSs rather than LTSs, leading to **novel deontic interpretation** of the classical modal and temporal operators

Syntax of MHML

$$\begin{array}{lcl} \phi & ::= & \mathit{true} \mid \neg\,\phi \mid \phi \wedge \phi' \mid \langle a \rangle\,\phi \mid [a]\,\phi \mid E\,\pi \mid A\,\pi \\ \pi & ::= & \phi\; U\,\phi' \mid \phi\; U^{\Box}\,\phi' \end{array}$$

Informal meaning of nonstandard operators

- $\langle a \rangle\,\phi$: a next state exists, reachable by **must** transition executing a, where ϕ holds
- $[\alpha]\,\phi$: in all next states, reachable by whatever transition executing a, ϕ holds
- $\phi\; U\,\phi'$: in the current or in a future state of a path, ϕ' holds, while ϕ holds in all preceding states of the path
- $\phi\; U^{\Box}\,\phi'$: in the current or in a future state of a path, ϕ' holds, while ϕ holds in all preceding states of the path, and the path leading to that state is a **must** path

MHML: Semantics over MTSs

The satisfaction relation $\models$ of MHML over MTSs is defined as follows (q is a state and σ is a full path)

Semantics of MHML

- $q \models \mathit{true}$ 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 $q \xrightarrow{a}_{\Diamond} q'$, for some $q' \in Q$, implies $q' \models \phi$
- $q \models E \, \pi$ 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 $\forall \, 1 \leq i < j : \sigma^{\Box}(i) \models \phi$

MHML: Deontic interpretation

Abbreviations

false $= \neg$*true*, $\phi \vee \phi' = \neg(\neg\phi \wedge \neg\phi')$, $\phi \implies \phi' = \neg\phi \vee \phi'$, $F\phi =$ (*true* $U\ \phi$), $F^{\Box}\phi =$ (*true* $U^{\Box}\ \phi$), $AG\phi = \neg EF\neg\phi$, $AG^{\Box}\phi = \neg EF^{\Box}\neg\phi$

Note

- Classical duality rules of Hennessy–Milner logic ($\langle a\rangle\phi = \neg[a]\neg\phi$) and of deontic logics ($P(a) = \neg O(\neg a)$) do not hold for MHML
- In fact, $\neg[a]\neg\phi$ corresponds to a weaker version of the classical diamond operator, namely:

$q \models P(a)\,\phi$ iff a next state **may** exist, reachable by executing action a, in which ϕ holds

MHML can express both **permitted** ($P(\cdot)$) and **obligatory** ($\langle\cdot\rangle$)

MHML: Deontic interpretation

Abbreviations

$\mathit{false} = \neg \mathit{true}$, $\phi \vee \phi' = \neg(\neg\phi \wedge \neg\phi')$, $\phi \implies \phi' = \neg\phi \vee \phi'$, $F\phi = (\mathit{true}\ U\ \phi)$, $F^{\Box}\phi = (\mathit{true}\ U^{\Box}\phi)$, $AG\phi = \neg EF\neg\phi$, $AG^{\Box}\phi = \neg EF^{\Box}\neg\phi$

Note

- Classical duality rules of Hennessy–Milner logic ($\langle a\rangle\phi = \neg[a]\neg\phi$) and of deontic logics ($P(a) = \neg O(\neg a)$) do not hold for MHML
- In fact, $\neg[a]\neg\phi$ corresponds to a weaker version of the classical diamond operator, namely:

$q \models P(a)\,\phi$ iff a next state **may** exist, reachable by executing action a, in which ϕ holds

MHML can express both **permitted** ($P(\cdot)$) and **obligatory** ($\langle\cdot\rangle$)

Example static and behavioural properties of families

A actions 1€ and 1\$ are exclusive (**alternative** features):

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

B action cappuccino cannot be executed in American coffee machines (**excludes** relation between features):

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

C a ringtone is rung whenever a cappuccino is delivered (**requires** relation between features):

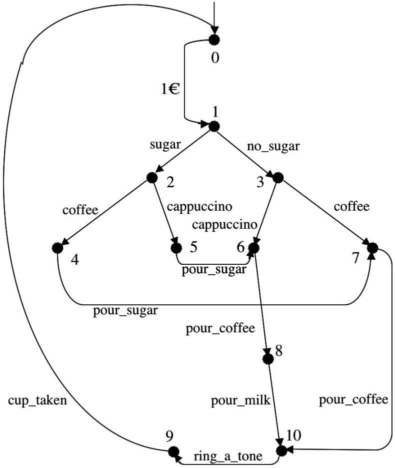
$$(EF\,\langle cappuccino\rangle\, true) \implies (AF\,\langle ring\_a\_tone\rangle\, true)$$

D once user has selected a coffee, a coffee is eventually delivered:

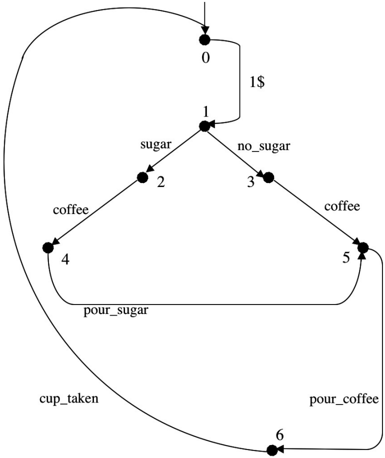
$$AG\,[coffee]\,AF^{\square}\,\langle pour\_coffee\rangle\, true$$

Global model-checking algorithm for MHML

for all $q \in Q$ **do**
  $L(q) := \{true\}$
for $i = 1$ **to** $length(\psi)$ **do**
  for all subformulae ϕ of ψ **such that** $length(\phi) = i$ **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 $\forall\, q' : q \xrightarrow{a}_{\Diamond} q',\ \phi_1 \in L(q')$ **then**
          $L(q) := L(q) \cup \{\phi\}$
    else if $\phi = \langle a\rangle\phi_1$ **then**
      for all $q \in Q$ **do**
        if $\exists\, q' : q \xrightarrow{a}_{\Box} q',\ \phi_1 \in L(q')$ **then**
          $L(q) := L(q) \cup \{\phi\}$
    else if $\phi = P(a)\,\phi_1$ **then**
      for all $q \in Q$ **do**
        if $\exists\, q' : q \xrightarrow{a}_{\Diamond} q',\ \phi_1 \in L(q')$ **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**
              $L(p) := L(p) \cup \{E\,(\phi_1\ U^{\Box}\ \phi_2)\}$
              $T := T \cup \{p\}$



(c) LTS of a European Coffee Machine



(d) LTS of an American Coffee Machine

Model-checking results

Property	Fig. (a)	Fig. (b)	Fig. (c)	Fig. (d)
A	*false*	*false*	*true*	*true*
B	*false*	*true*	*true*	*true*
C	*false*	*true*	*true*	*true*
D	*true*	*true*	*true*	*true*

Table: Results of verifying properties A–D on Figs.(a)–(d)

Deriving correct products from a product family

Behavioural properties of families

- Idea: prune optional (may) transitions in the MTS in a counter-example-guided way, i.e., based on model-checking techniques
- An algorithm can be devised in which the conjunction of the constraints is repeatedly model checked, first over the MTS of the product family and then 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

A vision for future research

- Our aim is to develop rigorous modelling techniques as well as analysis and verification tools that can be used for the systematic, large-scale provision and market segmentation of **software services**
- We foresee flexible design techniques with which **software service line organizations** can develop novel classes of service-oriented applications that can easily be adapted to customer requirements as well as to changes in the context in which, and while, they execute
- By superposing variability mechanisms on current languages for service design, based on policies and strategies defined by service providers, we envision the possibility to identify **variability points** that can be triggered at run-time to increase adaptability and optimize the (re)use of resources

- The resulting design techniques and support tools will be able to assist organizations to plan, optimize, and control the quality of software service provision, both at design- and at run-time

A vision for future research

- Our aim is to develop rigorous modelling techniques as well as analysis and verification tools that can be used for the systematic, large-scale provision and market segmentation of **software services**
- We foresee flexible design techniques with which **software service line organizations** can develop novel classes of service-oriented applications that can easily be adapted to customer requirements as well as to changes in the context in which, and while, they execute
- By superposing variability mechanisms on current languages for service design, based on policies and strategies defined by service providers, we envision the possibility to identify **variability points** that can be triggered at run-time to increase adaptability and optimize the (re)use of resources

- The resulting design techniques and support tools will be able to assist organizations to plan, optimize, and control the quality of software service provision, both at design- and at run-time

Concrete proposal: Modelling Variability, Evolvability, and Adaptability in Service Computing

First focus on the definition of the formal modelling framework, with a threefold objective:

1. Extend (semi)formal existing notations and languages for SOC with notions of variability through which increased levels of flexibility and adaptability can be achieved in software service provision
2. Define a rigorous semantics of variability over behavioural models of services that can support a number of design- and run-time analysis techniques
3. Develop analysis and verification techniques that remain effective over specifications with variability points, including situations in which the variability is triggered at run-time