A deontic logical framework for modelling product families

Research in progress

P. Asirelli, M.H. ter Beek, S. Gnesi, A. Fantechi

ISTI-CNR, Università di Firenze

VAMOS 2010
Linz, Austria

27 January 2010

Outline

Aim of our research activity

- To express in a single framework feature-based constraints over the products of a family and constraints over their behaviour
- To provide tools to support this framework with formal verification

In our search for a single logical framework in which to express both static and behavioural aspects of product families:

- we present a straightforward characterization of feature models by means of deontic logics
- we define a deontic extension of a behavioural logic, called DHML, that allows to express in a single framework both static constraints over the products of a family and constraints over their behaviour
- we give a semantic interpretation of DHML over MTSs, for which a verification framework based on model-checking techniques could be implemented

Aim of our research activity

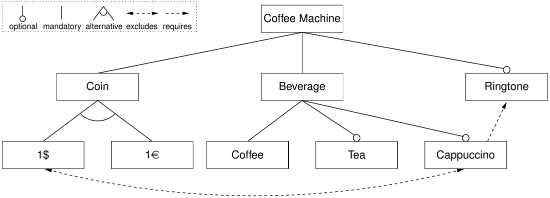
- To express in a single framework feature-based constraints over the products of a family and constraints over their behaviour
- To provide tools to support this framework with formal verification

In our search for a single logical framework in which to express both static and behavioural aspects of product families:

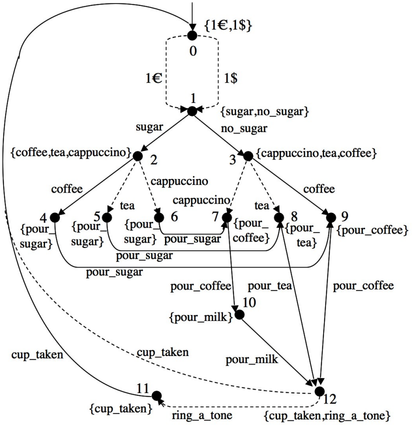
- we present a straightforward characterization of feature models by means of deontic logics
- we define a deontic extension of a behavioural logic, called DHML, that allows to express in a single framework both static constraints over the products of a family and constraints over their behaviour
- we give a semantic interpretation of DHML over MTSs, for which a verification framework based on model-checking techniques could be implemented

Running example: Coffee machine family

Feature model:



Modal Transition System:



— required transitions

- - - possible transitions

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

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

- $O(\alpha)$: action α is *obligatory* (required transition)
- $P(\alpha) = \neg O(\neg\alpha)$: action α is *permitted* (possible transition)
  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

- $O(\alpha)$: action α is *obligatory* (required transition)
- $P(\alpha) = \neg O(\neg\alpha)$: action α is *permitted* (possible transition)
  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 $(A \implies \neg P(B)) \wedge (B \implies \neg P(A))$

Example static properties of families

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

Example static properties of families

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

Example static properties of families

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

DHML: Deontic Hennesy-Milner Logic with until

DHML is a temporal logic based on the “Hennessy-Milner logic with until” [Larsen], augmented with the deontic O and P operators à la PDL logic [Castro & Maibaum] and the path operators E and A from CTL [Clarke et alii]

Syntax of DHML

$$\begin{aligned} \phi &::= \mathit{true} \mid p \mid \neg\phi \mid \phi \wedge \phi' \mid [\alpha]\phi \mid E\pi \mid A\pi \mid O(\alpha) \mid P(\alpha) \\ \pi &::= \phi\, U\, \phi' \end{aligned}$$

Informal meaning of remaining operators (p is a proposition)

- $[\alpha]\,\phi$: for all next states reachable with α, ϕ holds
- $E\,\pi$: there exists a path on which π holds
- $A\,\pi$: on each of the possible paths π holds
- $\phi\, U\, \phi'$: in the current or a future state ϕ' holds, while ϕ holds until that state

Usual abbreviations

$\mathit{false} = \neg\mathit{true}$, $\phi \vee \phi' = \neg(\neg\phi \wedge \neg\phi')$, $\phi \implies \phi' = \neg\phi \vee \phi'$, $\langle\alpha\rangle\phi = \neg[\alpha]\neg\phi$, $EF\phi = E\,(\mathit{tt}\; U\; \phi)$, $AG\phi = \neg EF\neg\phi$

DHML: Deontic Hennesy-Milner Logic with until

DHML is a temporal logic based on the “Hennessy-Milner logic with until” [Larsen], augmented with the deontic O and P operators à la PDL logic [Castro & Maibaum] and the path operators E and A from CTL [Clarke et alii]

Syntax of DHML

$$
\begin{array}{lll}
\phi & ::= & \mathit{true} \mid p \mid \neg\phi \mid \phi \wedge \phi' \mid [\alpha]\phi \mid E\pi \mid A\pi \mid O(\alpha) \mid P(\alpha) \\
\pi & ::= & \phi \; U \; \phi'
\end{array}
$$

Informal meaning of remaining operators (p is a proposition)

- $[\alpha]\,\phi$: for all next states reachable with α, ϕ holds
- $E\,\pi$: there exists a path on which π holds
- $A\,\pi$: on each of the possible paths π holds
- $\phi\;U\;\phi'$: in the current or a future state ϕ' holds, while ϕ holds until that state

Usual abbreviations

$\mathit{false} = \neg\mathit{true}$, $\phi \vee \phi' = \neg(\neg\phi \wedge \neg\phi')$, $\phi \implies \phi' = \neg\phi \vee \phi'$, $\langle\alpha\rangle\phi = \neg[\alpha]\neg\phi$,
$EF\phi = E\,(tt\;U\;\phi)$, $AG\phi = \neg EF\neg\phi$

DHML: Deontic Hennesy-Milner Logic with until

DHML is a temporal logic based on the “Hennessy-Milner logic with until” [Larsen], augmented with the deontic O and P operators à la PDL logic [Castro & Maibaum] and the path operators E and A from CTL [Clarke et alii]

Syntax of DHML

$$\begin{array}{lcl} \phi & ::= & true \mid p \mid \neg\phi \mid \phi \wedge \phi' \mid [\alpha]\phi \mid E\pi \mid A\pi \mid O(\alpha) \mid P(\alpha) \\ \pi & ::= & \phi \; U \; \phi' \end{array}$$

Informal meaning of remaining operators (p is a proposition)

- $[\alpha]\,\phi$: for all next states reachable with α, ϕ holds
- $E\,\pi$: there exists a path on which π holds
- $A\,\pi$: on each of the possible paths π holds
- $\phi\; U\, \phi'$: in the current or a future state ϕ' holds, while ϕ holds until that state

Usual abbreviations

$false = \neg true$, $\phi \vee \phi' = \neg(\neg\phi \wedge \neg\phi')$, $\phi \implies \phi' = \neg\phi \vee \phi'$, $\langle\alpha\rangle\phi = \neg[\alpha]\neg\phi$,
$EF\phi = E\,(tt\; U\; \phi)$, $AG\phi = \neg EF\neg\phi$

DHML: Deontic Hennesy-Milner Logic with until

DHML is a temporal logic based on the "Hennessy-Milner logic with until" [Larsen], augmented with the deontic O and P operators à la PDL logic [Castro & Maibaum] and the path operators E and A from CTL [Clarke et alii]

Syntax of DHML

$$
\begin{array}{lcl}
\phi & ::= & \mathit{true} \mid p \mid \neg\phi \mid \phi \wedge \phi' \mid [\alpha]\phi \mid E\pi \mid A\pi \mid O(\alpha) \mid P(\alpha) \\
\pi & ::= & \phi \; U \; \phi'
\end{array}
$$

Informal meaning of remaining operators (p is a proposition)

- $[\alpha]\,\phi$: for all next states reachable with α, ϕ holds
- $E\,\pi$: there exists a path on which π holds
- $A\,\pi$: on each of the possible paths π holds
- $\phi\; U\, \phi'$: in the current or a future state ϕ' holds, while ϕ holds until that state

Usual abbreviations

$\mathit{false} = \neg\mathit{true}$, $\phi \vee \phi' = \neg(\neg\phi \wedge \neg\phi')$, $\phi \implies \phi' = \neg\phi \vee \phi'$, $\langle\alpha\rangle\phi = \neg[\alpha]\neg\phi$,
$EF\phi = E\,(tt\; U\; \phi)$, $AG\phi = \neg EF\neg\phi$

DHML: Semantics with MTS as interpretation structure

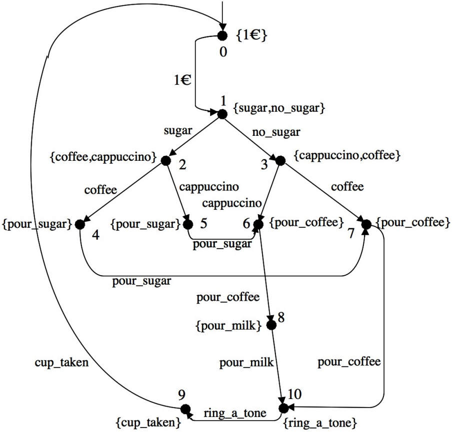
- $\rightarrow \subseteq S \times Act \times S$: transitions between states S are labelled with actions Act
- transitions are either required (—) or possible (- - -)
- $L : S \rightarrow 2^{AP}$: states are labelled with Atomic Propositions AP as well as with the events allowed in the states (i.e. $Act \subseteq AP$)
- $P \subseteq S \times Act$ denotes the actions which are permitted in a state: $P(s, \alpha)$ iff $\alpha \in L(s)$

The satisfaction relation of DHML is defined as follows:

- $s \models true$ always holds
- $s \models p$ iff $p \in L(s)$
- $s \models \neg\phi$ iff not $s \models \phi$
- $s \models \phi \wedge \phi'$ iff $s \models \phi$ and $s \models \phi'$
- $s \models [\alpha]\phi$ iff $s \xrightarrow{\alpha}_{\Diamond} s'$, for some $s' \in S$, implies $s' \models \phi$
- $s \models E\pi$ iff there exists a path σ starting in state s such that $\sigma \models \pi$
- $s \models A\pi$ iff $\sigma \models \pi$ for all paths σ starting in state s
- $s \models P(\alpha)$ iff $P(s, \alpha)$ holds
- $s \models O(\alpha)$ iff $P(s, \alpha)$ holds and $\exists s' : s \xrightarrow{\alpha}_{\Box} s'$
- $\sigma \models [\phi \; U \; \phi']$ iff there exists a state s_j, for some $j \geq 0$, on the path σ such that for all states s_k, with $j \leq k$, $s_k \models \phi'$ while for all states s_i, with $0 \leq i < j$, $s_i \models \phi$

MTS of a European Coffee Machine

A product is represented by a MTS with only required transitions:



Example behavioural properties of families

Behavioural properties of families

1. It is possible to get a coffee with 1€:

$$[1€]\ EF\ <coffee>\ true$$

2. It is always possible to ask for sugar:

$$AF\ <sugar>\ true$$

3. It is not possible to get a beverage without inserting a coin:

$$AG\,(\neg(coffee \vee tea \vee cappuccino)\ U\ (<1€>\ true \vee <1\$>\ true))$$

Example static and behavioural properties of families

Static and behavioural properties of families

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

$$((EF\ <1\$>\ true) \implies (AG\ \neg P(1€))) \wedge$$
$$((EF\ <1€>\ true) \implies (AG\ \neg P(1\$)))$$

2. a cappuccino is only offered by European products (**excludes** relation between features):

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

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

$$(EF\ <cappuccino>\ true) \implies (AF\ O(ring\_a\_tone))$$

Conclusions and open problems

Research in Progress—what we have done so far

1. defined a deontic characterization of a feature model (static requirements over a family)
2. defined behavioural deontic logic DHML to express the behavioural variability of a family

Research in Progress—what we are working on

- a model checker able to automatically verify DHML formulae over models described as MTSs, with possible constraints expressed in DHML itself
- exploit the relation between MTSs and L^2TSs to reuse the UMC model-checking engine (on-the-fly model checker designed for the efficient verification of UCTL logic over L^2TSs)
- compare the expressiveness of UCTL and DHML, which might lead to enhancements to the model-checking engine to cover DHML deontic operators

Research in Progress—what remains to be done

- how to express dependencies of variation points?
- how to identify properties that, proved on a family, are preserved by all its products?
- how does this scale to real problems and to incremental family construction?
- how to hide the logic and verification technicalities from the end user?
- what else???

Conclusions and open problems

Research in Progress—what we have done so far

1. defined a deontic characterization of a feature model (static requirements over a family)
2. defined behavioural deontic logic DHML to express the behavioural variability of a family

Research in Progress—what we are working on

- a model checker able to automatically verify DHML formulae over models described as MTSs, with possible constraints expressed in DHML itself
- exploit the relation between MTSs and L^2TSs to reuse the UMC model-checking engine (on-the-fly model checker designed for the efficient verification of UCTL logic over L^2TSs)
- compare the expressiveness of UCTL and DHML, which might lead to enhancements to the model-checking engine to cover DHML deontic operators

Research in Progress—what remains to be done

- how to express dependencies of variation points?
- how to identify properties that, proved on a family, are preserved by all its products?
- how does this scale to real problems and to incremental family construction?
- how to hide the logic and verification technicalities from the end user?
- what else???

Conclusions and open problems

Research in Progress—what we have done so far

1. defined a deontic characterization of a feature model (static requirements over a family)
2. defined behavioural deontic logic DHML to express the behavioural variability of a family

Research in Progress—what we are working on

- a model checker able to automatically verify DHML formulae over models described as MTSs, with possible constraints expressed in DHML itself
- exploit the relation between MTSs and L^2TSs to reuse the UMC model-checking engine (on-the-fly model checker designed for the efficient verification of UCTL logic over L^2TSs)
- compare the expressiveness of UCTL and DHML, which might lead to enhancements to the model-checking engine to cover DHML deontic operators

Research in Progress—what remains to be done

- how to express dependencies of variation points?
- how to identify properties that, proved on a family, are preserved by all its products?
- how does this scale to real problems and to incremental family construction?
- how to hide the logic and verification technicalities from the end user?
- what else???