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
1. Aim of our research activity
2. Running example
3. Deontic logic
4. Deontic characterization of feature models
5. Our DHML logic
6. Static and behavioural properties of families
7. Conclusions
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 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.

⇒ Deontic logic seems to be a natural candidate for expressing the conformance of products with respect to variability rules.
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.

⇒ 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 (¬), conjunction (∧), disjunction (∨) and implication (⇒), 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 \( \alpha \) is obligatory (required transition)
- \( P(\alpha) = \neg O(\neg \alpha) \): action \( \alpha \) is permitted (possible transition) if and only if its negation is not obligatory
A deontic logic consists of the standard operators of propositional logic, i.e. negation ($\neg$), conjunction ($\land$), disjunction ($\lor$) and implication ($\implies$), 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 $\alpha$ is *obligatory* (required transition)
- $P(\alpha) = \neg O(\neg \alpha)$: action $\alpha$ 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:

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

$$\Phi(A_1, A_2) = (O(A_1) \lor O(A_2)) \land \neg(P(A_1) \land P(A_2)) \text{ if } A_1, A_2 \text{ alternative,}$$

and otherwise:

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

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

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

- If $A$ excludes $B$, add the formula $(A \implies \neg P(B)) \land (B \implies \neg P(A))$.
Construction of deontic characterization of FM

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

$$A \implies \Phi(A_1, A_2),$$

where $\Phi(A_1, A_2)$ is defined as:

$$\Phi(A_1, A_2) = (O(A_1) \lor O(A_2)) \land \neg (P(A_1) \land P(A_2))$$

if $A_1, A_2$ alternative, and otherwise:

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

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

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

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

• If $A$ excludes $B$, add the formula $(A \implies \neg P(B)) \land (B \implies \neg P(A))$
• If $A$ is a feature and $A_1$ and $A_2$ are subfeatures, add the formula:
\[
A \implies \Phi(A_1, A_2), \quad \text{where } \Phi(A_1, A_2) \text{ is defined as:}
\]
\[
\Phi(A_1, A_2) = (O(A_1) \lor O(A_2)) \land \neg (P(A_1) \land P(A_2)) \text{ if } A_1, A_2 \text{ alternative,}
\]
and otherwise:
\[
\Phi(A_1, A_2) = \phi(A_1) \land \phi(A_2), \text{ in which } A_i, \text{ for } i \in \{1, 2\}, \text{ is defined as:}
\]
\[
\phi(A_i) = \begin{cases} 
P(A_i) & \text{if } A_i \text{ is optional} \\
O(A_i) & \text{if } A_i \text{ is mandatory}
\end{cases}
\]

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

• If $A$ excludes $B$, add the formula $(A \implies \neg P(B)) \land (B \implies \neg P(A))$
Example static properties of families

Characteristic formula of Coffee machine family

\[ O(\text{Coin}) \land O(\text{Beverage}) \land P(\text{Ringtone}) \land \]
\[ \text{Coin} \implies (O(1\text{\$}) \lor O(1\text{\€})) \land \neg (P(1\text{\$}) \land P(1\text{\€})) \]
\[ \text{Beverage} \implies O(\text{Coffee}) \land P(\text{Tea}) \land P(\text{Cappuccino}) \land \]
\[ \text{Cappuccino} \implies O(\text{Ringtone}) \]
\[ (1\text{\$} \implies \neg P(\text{Cappuccino})) \land (\text{Cappuccino} \implies \neg P(1\text{\$})) \]

Two example coffee machines

\[ \text{CM1} = \{ \text{Coin, 1\€, Beverage, Coffee} \} \]
\[ \text{CM2} = \{ \text{Coin, 1\€, Beverage, Coffee, Cappuccino} \} \]

CM1 in family, but CM2 not: Cappuccino \implies O(\text{Ringtone}) \text{ false}
Example static properties of families

Characteristic formula of Coffee machine family

\[ O(\text{Coin}) \land O(\text{Beverage}) \land P(\text{Ringtone}) \]
\[ \land \]
\[ \text{Coin} \implies (O(1\$) \lor O(1\euro)) \land \neg(P(1\$) \land P(1\euro)) \]
\[ \text{Beverage} \implies O(\text{Coffee}) \land P(\text{Tea}) \land P(\text{Cappuccino}) \]
\[ \land \]
\[ \text{Cappuccino} \implies O(\text{Ringtone}) \]
\[ (1\$ \implies \neg P(\text{Cappuccino})) \land (\text{Cappuccino} \implies \neg P(1\$)) \]

Two example coffee machines

\[ \text{CM1} = \{\text{Coin, 1\euro, Beverage, Coffee}\} \]
\[ \text{CM2} = \{\text{Coin, 1\euro, Beverage, Coffee, Cappuccino}\} \]

CM1 in family, but CM2 not: \( \text{Cappuccino} \implies O(\text{Ringtone}) \) false
Example static properties of families

Characteristic formula of Coffee machine family

\[ O(\text{Coin}) \land O(\text{Beverage}) \land P(\text{Ringtone}) \]
\[ \land \]
\[ \text{Coin} \implies (O(1\$) \lor O(1\€)) \land \neg (P(1\$) \land P(1\€)) \]
\[ \text{Beverage} \implies O(\text{Coffee}) \land P(\text{Tea}) \land P(\text{Cappuccino}) \]
\[ \land \]
\[ \text{Cappuccino} \implies O(\text{Ringtone}) \]
\[ (1\$ \implies \neg P(\text{Cappuccino})) \land (\text{Cappuccino} \implies \neg P(1\$)) \]

Two example coffee machines

\[ \text{CM1} = \{ \text{Coin, 1\€, Beverage, Coffee} \} \]
\[ \text{CM2} = \{ \text{Coin, 1\€, Beverage, Coffee, Cappuccino} \} \]

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

$$
\phi ::= \text{true} \mid p \mid \neg \phi \mid \phi \land \phi' \mid [\alpha] \phi \mid E\pi \mid A\pi \mid O(\alpha) \mid P(\alpha) \\
\pi ::= \phi \ U \phi'
$$

**Informal meaning of remaining operators ($p$ is a proposition)**

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

**Usual abbreviations**

$$
\text{false} = \neg \text{true}, \quad \phi \lor \phi' = \neg (\neg \phi \land \neg \phi'), \quad \phi \implies \phi' = \neg \phi \lor \phi', \quad \langle \alpha \rangle \phi = \neg [\alpha] \neg \phi, \\
EF\phi = E (tt U \phi), \quad AG\phi = \neg EF \neg \phi
$$
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**

$$
\phi ::= \text{true} \mid p \mid \neg \phi \mid \phi \land \phi' \mid [\alpha]\phi \mid E\pi \mid A\pi \mid O(\alpha) \mid P(\alpha)
$$

$$
\pi ::= \phi \,U\,\phi'
$$

**Informal meaning of remaining operators ($p$ is a proposition)**

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

**Usual abbreviations**

$$
false = \neg \text{true}, \quad \phi \lor \phi' = \neg (\neg \phi \land \neg \phi'), \quad \phi \implies \phi' = \neg \phi \lor \phi', \quad \langle \alpha \rangle \phi = \neg [\alpha] \neg \phi, \\
EF\phi = E (tt \, U \, \phi), \quad AG\phi = \neg EF \neg \phi
$$
DHML: Deontic Hennessy-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

$$\phi ::= \text{true} | \ p | \neg \phi | \phi \land \phi' | [\alpha] \phi | E\pi | A\pi | O(\alpha) | P(\alpha)$$

$$\pi ::= \phi \ U \phi'$$

Informal meaning of remaining operators ($p$ is a proposition)

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

Usual abbreviations

$$\text{false} = \neg \text{true}, \ \phi \lor \phi' = \neg (\neg \phi \land \neg \phi'), \ \phi \rightarrow \phi' = \neg \phi \lor \phi', \ \langle \alpha \rangle \phi = \neg [\alpha] \neg \phi, \ EF \phi = E (tt \ U \phi), \ AG \phi = \neg EF \neg \phi$$
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

$$\phi ::= \text{true} \mid p \mid \neg \phi \mid \phi \land \phi' \mid [\alpha] \phi \mid E\pi \mid A\pi \mid O(\alpha) \mid P(\alpha)$$

$$\pi ::= \phi \text{ U } \phi'$$

Informal meaning of remaining operators ($p$ is a proposition)

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

Usual abbreviations

$$\text{false} = \neg \text{true}, \quad \phi \lor \phi' = \neg (\neg \phi \land \neg \phi'), \quad \phi \implies \phi' = \neg \phi \lor \phi', \quad \langle \alpha \rangle \phi = \neg [\alpha] \neg \phi,$$

$$\text{EF} \phi = E (\text{tt U } \phi), \quad \text{AG} \phi = \neg \text{EF} \neg \phi$$
DHML: Semantics with MTS as interpretation structure

- \( \rightarrow \subseteq S \times \text{Act} \times S \): transitions between states \( S \) are labelled with actions \( \text{Act} \)
- Transitions are either required (—) or possible (---)
- \( L : S \rightarrow 2^{\text{AP}} \): states are labelled with Atomic Propositions \( \text{AP} \) as well as with the events allowed in the states (i.e. \( \text{Act} \subseteq \text{AP} \))
- \( P \subseteq S \times \text{Act} \) 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:

- \( s \models \text{true} \) always holds
- \( s \models p \) iff \( p \in L(s) \)
- \( s \models \neg \phi \) iff not \( s \models \phi \)
- \( s \models \phi \land \phi' \) iff \( s \models \phi \) and \( s \models \phi' \)
- \( s \models [\alpha] \phi \) iff \( s \xrightarrow{\alpha} s' \), for some \( s' \in S \), implies \( s' \models \phi \)
- \( s \models E \pi \) iff there exists a path \( \sigma \) starting in state \( s \) such that \( \sigma \models \pi \)
- \( s \models A \pi \) iff \( \sigma \models \pi \) for all paths \( \sigma \) starting in state \( s \)
- \( s \models P(\alpha) \) iff \( P(s, \alpha) \) holds
- \( s \models O(\alpha) \) iff \( P(s, \alpha) \) holds and \( \exists s' : s \xrightarrow{\alpha} \square s' \)
- \( \sigma \models [\phi \ U \ \phi'] \) iff there exists a state \( s_j \), for some \( j \geq 0 \), on the path \( \sigma \) such that for all states \( s_k \), with \( j \leq k \), \( s_k \models \phi' \) while for all states \( s_i \), with \( 0 \leq i < j \), \( s_i \models \phi \)
A product is represented by a MTS with only required transitions:
Example 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:
   \[ AG(\neg(coffee \lor tea \lor cappuccino) \ U (\langle 1€ \rangle \ true \lor \langle 1$ \rangle \ true)) \]
Example static and behavioural properties of families

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

\[(\text{EF} < 1\$ > \text{true}) \implies (\text{AG} \neg P(1\€))) \land \]
\[\text{(EF} < 1\€ > \text{true}) \implies (\text{AG} \neg P(1\$)))\]

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

\[(\text{EF} < \text{cappuccino} > \text{true}) \implies (\text{AG} \neg P(1\$))) \land \]
\[\text{(EF} < 1\$ > \text{true}) \implies (\text{AG} \neg P(\text{cappuccino})))\]

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

\[(\text{EF} < \text{cappuccino} > \text{true}) \implies (\text{AF} \ O(\text{ring_a_tone}))\]
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 L₂TSs to reuse the UMC model-checking engine (on-the-fly model checker designed for the efficient verification of UCTL logic over L₂TSs).
- 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 $L^2$TSs to reuse the UMC model-checking engine (on-the-fly model checker designed for the efficient verification of UCTL logic over $L^2$TSs)
- 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 $L^2$TSs to reuse the UMC model-checking engine (on-the-fly model checker designed for the efficient verification of UCTL logic over $L^2$TSs)
- 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???