Demonstration of a model checker for the analysis of product variability

M.H. ter Beek, S. Gnesi & F. Mazzanti
ISTI–CNR, Pisa, Italy

Presenter: Ina Schaefer

SPLC 2012
Salvador, Bahia, Brazil
5 September 2012
Outline

1. Background and aim of the research activity
2. Running example: a family of coffee machines
3. Demo of the Variability Model Checker VMC
4. Discussion and future work
Paradigm

To develop a family of products (product line) 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

Product variants can be derived from a product family, thus allowing for reuse and differentiation

Production process

Maximize commonalities of product whilst minimizing cost of variations
(Software) Product Line Engineering

Paradigm
To develop a family of products (product line) 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

Product variants can be derived from a product family, thus allowing for reuse and differentiation

Production process
Maximize commonalities of product whilst minimizing cost of variations
(Software) Product Line Engineering

Paradigm
To develop a family of products (product line) 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

Product variants can be derived from a product family, thus allowing for reuse and differentiation

Production process
Maximize commonalities of product whilst minimizing cost of variations
Variability management/modeling

Feature modeling
Provide compact representations of all the products of a product family in terms of their *features* (pieces of functionality)

Variability modelling
How to explicitly define *optional*, *alternative*, *mandatory*, *required*, or *excluded* features of a product family as variation points

Managing variability with formal methods
Show that a certain product belongs to a product family or—instead—derive a product from a family by properly selecting features
Formally prove characteristics of products and families alike
Variability management/modeling

Feature modeling
Provide compact representations of all the products of a product family in terms of their *features* (pieces of functionality)

Variability modelling
How to explicitly define *optional*, *alternative*, *mandatory*, *required*, or *excluded* features of a product family as variation points

Managing variability with formal methods
Show that a certain product belongs to a product family or—instead—derive a product from a family by properly selecting features
Formally prove characteristics of products and families alike
**Variability management/modeling**

**Feature modeling**

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

**Variability modelling**

How to explicitly define *optional*, *alternative*, *mandatory*, *required*, or *excluded* features of a product family as variation points

**Managing variability with formal methods**

Show that a certain product belongs to a product family or—instead—derive a product from a family by properly selecting features

Formally prove characteristics of products and families alike
Aim of the research activity at large

**Aim**

- One formal framework to express both feature-based constraints over the products of a family and constraints over their behavior
- Tool support for product derivation and for the formal verification (by model checking) of properties over products and families alike

**Outcome:** iFM’10, ACOTA @ ASE’10, PLEASE @ ICSE’11, FMOODS’11, SEW-34 @ FM’11, SPLC’11, iFM’12 & FM’12

- **MTS:** Modal Transition Systems (Larsen et al.)
- **MHML:** CTL-like action- and state-based branching-time temporal logic (a.k.a. $\forall\mathrm{ACTL}$)
- **VMC:** Variability Model Checker

M.H. ter Beek et al. (ISTI–CNR, Pisa, Italy)
Aim of the research activity at large

**Aim**

- One formal framework to express both feature-based constraints over the products of a family and constraints over their behavior
- Tool support for product derivation and for the formal verification (by model checking) of properties over products and families alike

**Outcome:** iFM’10, ACOTA @ ASE’10, PLEASE @ ICSE’11, FMOODS’11, SEW-34 @ FM’11, SPLC’11, iFM’12 & FM’12

- **MTS:** Modal Transition Systems (Larsen et al.)
- **MHML:** CTL-like action- and state-based branching-time temporal logic (a.k.a. $\nu$ACTL)
- **VMC:** Variability Model Checker
Feature-based constraints

- The only accepted coins are 1€, exclusively for European products, and 1$, exclusively for Canadian products (alternative features).
- All products must offer coffee (mandatory feature); only European products may offer cappuccino (excludes relation among features).
- A ringtone must be rung in products offering cappuccino (requires relation among features), while it may be rung in other products (optional feature).

Behavioral constraints

- After coin insertion, user must press a button to choose whether (s)he wants sugar, after which (s)he may select a beverage.
- The optional ringtone is rung after delivering a beverage.
Feature-based constraints

- The only accepted coins are 1€, exclusively for European products, and 1$, exclusively for Canadian products (alternative features).
- All products must offer coffee (mandatory feature); only European products may offer cappuccino (excludes relation among features).
- A ringtone must be rung in products offering cappuccino (requires relation among features), while it may be rung in other products (optional feature).

Behavioral constraints

- After coin insertion, user must press a button to choose whether (s)he wants sugar, after which (s)he may select a beverage.
- The optional ringtone is rung after delivering a beverage.
Coffee machine family: Feature model

10 different valid products (coffee machines defined by features)

\[
\{m, s, o, b, c, \epsilon\}, \{m, s, o, b, c, \epsilon, r\}, \{m, s, o, b, c, \epsilon, t\}, \\
\{m, s, o, b, c, \$, r\}, \{m, s, o, b, c, \$, t\}, \\
\{m, s, o, b, c, \epsilon, t, r\}, \{m, s, o, b, c, \epsilon, p, r\}, \\
\{m, s, o, b, c, \$, t, r\}, \{m, s, o, b, c, \epsilon, p, r, t\}\]
Coffee machine family: Feature model

10 different valid products (coffee machines defined by features)

\[
\{\{m, s, o, b, c, \€\}\}, \{m, s, o, b, c, \€, r\}, \{m, s, o, b, c, \€, t\}, \\
\{m, s, o, b, c, $\}, \{m, s, o, b, c, $, r\}, \{m, s, o, b, c, $, t\}, \\
\{m, s, o, b, c, \€, t, r\}, \{m, s, o, b, c, \€, p, r\}, \\
\{m, s, o, b, c, $, t, r\}, \{m, s, o, b, c, \€, p, r, t\}\}
\]
Modal Transition Systems (MTSs)

Use for behavioral modeling of SPLs recognized by Uchitel et al.

An MTS is an LTS distinguishing **optional** (may) and **mandatory** (must) transitions to formalize a product family’s

- *underlying behavior*, shared among all products, and
- *variation points*, differentiating between products

A product (LTS) is derived by including all (reachable) must transitions and a subset of the (reachable) may transitions

MTS however cannot model variability constraints regarding **alternative** features nor regarding the **requires** and **excludes** inter-feature relations

A solution: add a set of variability constraints (hiding the logic!) to the MTS to define which derivable products must be considered valid ones (we defined an algorithm to derive only—and possibly all—valid ones)
Modal Transition Systems (MTSs)

Use for behavioral modeling of SPLs recognized by Uchitel et al.

An MTS is an LTS distinguishing **optional** (may) and **mandatory** (must) transitions to formalize a product family’s

- *underlying behavior*, shared among all products, and
- *variation points*, differentiating between products

A product (LTS) is derived by including all (reachable) must transitions and a subset of the (reachable) may transitions

MTS however cannot model variability constraints regarding **alternative** features nor regarding the **requires** and **excludes** inter-feature relations

A solution: add a set of variability constraints (hiding the logic!) to the MTS to define which derivable products must be considered valid ones (we defined an algorithm to derive only—and possibly all—valid ones)
Modal Transition Systems (MTSs)

Use for behavioral modeling of SPLs recognized by Uchitel et al.

An MTS is an LTS distinguishing optional (may) and mandatory (must) transitions to formalize a product family’s

- underlying behavior, shared among all products, and
- variation points, differentiating between products

A product (LTS) is derived by including all (reachable) must transitions and a subset of the (reachable) may transitions

MTS however cannot model variability constraints regarding alternative features nor regarding the requires and excludes inter-feature relations

A solution: add a set of variability constraints (hiding the logic!) to the MTS to define which derivable products must be considered valid ones (we defined an algorithm to derive only—and possibly all—valid ones)
Coffee machine family: a possible MTS model
A valid European coffee machine ($\{m, s, o, b, c, €, p, r\}$)
A valid Canadian coffee machine ($\{m, s, o, b, c, $, r\}$)
A correct but not valid product LTS of above MTS
VMC: Variability Model Checker

A tool for modeling and analysis of behavioral variability in SPLs

Given a textual encoding of an MTS and a set of variability constraints:
- interactively explore the MTS
- derive and explore (all) the family’s valid products (LTSs)
- visualize the family/products graphically as MTS/LTSs
- verify branching-time temporal logic properties over family/products
- interactively explain why a product does (not) satisfy a property

Freely usable online: [http://fmtlab.isti.cnr.it/vmc/](http://fmtlab.isti.cnr.it/vmc/)

Core is a command-line version of the model checker and of a product generation procedure, both stand-alone executables written in Ada, so:
- easy to compile for Windows | Linux | Solaris | Mac OS X
- wrapped with a set of CGI scripts handled by a web server:
  - easy to build a graphical html-oriented GUI
  - easy to integrate with graph drawing tools
A tool for modeling and analysis of behavioral variability in SPLs

Given a textual encoding of an MTS and a set of variability constraints:

- interactively explore the MTS
- derive and explore (all) the family’s valid products (LTSs)
- visualize the family/products graphically as MTS/LTSs
- verify branching-time temporal logic properties over family/products
- interactively explain why a product does (not) satisfy a property

Freely usable online: http://fmtlab.isti.cnr.it/vmc/

Core is a command-line version of the model checker and of a product generation procedure, both stand-alone executables written in Ada, so:

- easy to compile for Windows | Linux | Solaris | Mac OS X
- wrapped with a set of CGI scripts handled by a web server:
  - easy to build a graphical html-oriented GUI
  - easy to integrate with graph drawing tools
VMC in relation to other model checkers

VMC itself is a product of a family of model checkers

- developed and highly optimized at ISTI–CNR during last decades
- on-the-fly: in general no need to generate/explore full state space
- capable of verifying properties specified in a CTL-like action- and state-based branching-time temporal logic

products: FMC (input: CCS/CSP/LOTOS-like process algebras), UMC (input: UML state machines) & CMC (input: WS orchestration calculus)

Detailed comparison with MTSA, SNIP & SPLVERIFIER in paper

MTSA: also LTS-based, but no specific features to analyze variability
SNIP: no graphical interface, not highly optimized, but built-in support for feature diagrams and model-checking algorithms tailored for SPLE
SPLVERIFIER: focuses on features, but addressing the detection of feature interactions instead of variability
VMC in relation to other model checkers

VMC itself is a product of a family of model checkers
- developed and highly optimized at ISTI–CNR during last decades
- on-the-fly: in general no need to generate/explore full state space
- capable of verifying properties specified in a CTL-like action- and state-based branching-time temporal logic

products: FMC (input: CCS/CSP/LOTOS-like process algebras), UMC (input: UML state machines) & CMC (input: WS orchestration calculus)

Detailed comparison with MTSA, SNIP & SPL

MTSA: also LTS-based, but no specific features to analyze variability

SNIP: no graphical interface, not highly optimized, but built-in support for feature diagrams and model-checking algorithms tailored for SPLE

SPL: focuses on features, but addressing the detection of feature interactions instead of variability
VMC’s web interface

Welcome to VMC

Documentation:
Sample code:
Download:

Requirements:
Any browser with frames, javascript, DHTML, SVG support and popup enabled.
E.g. Firefox, Chrome, Safari, Opera are OK
compatibility with Internet Explorer not tested.

Authors:
Franco Mazzanti (http://fmt.isti.cnr.it/~mazzanti), Aldi Sulova

Credits:
Graphics generated with GraphViz (http://www.graphviz.org/)
Graph minimization with MCRL2-ltsconvert

The running example is available by selecting coffeemodel6.txt
Family of coffee machines specified in VMC

```
1  T1 = euro(optional).T2 + dollar(optional).T2
2  T2 = sugar.T3 + no_sugar.T4
3  T3 = coffee.T5 + sugared_cappuccino(optional).T6 + sugared_tea(optional).T7
4  T4 = coffee.T8 + unsugared_cappuccino(optional).T9 + unsugared_tea(optional).T10
5  T5 = pour_sugar.T8
6  T6 = pour_sugar.T9
7  T7 = pour_sugar.T10
8  T8 = pour_coffee.T13
9  T9 = pour_coffee.T11 + pour_milk.T12
10 T10 = pour_tea.T13
11 T11 = pour_milk.T13
12 T12 = pour_coffee.T13
13 T13 = ring_a_tone(optional).T14 + no_ring(optional).T14
14 T14 = take_cup.T1
15
16  net SYS = T1
17
18  Constraints {
19    euro ALT dollar
20    unsugared_cappuccino IFF sugared_cappuccino
21    unsugared_tea IFF sugared_tea
22    dollar EXC unsugared_cappuccino
23    unsugared_cappuccino REQ ring_a_tone
24    ring_a_tone ALT no_ring
25    }
26
27```

Permitted variability constraints ALTernative, EXCludes, REquires, and IFF (shorthand for bilateral REqs) hide the logic formalization from user
View the graph in **DOT** format or as a **PDF** picture or as plain **SVG** data.

The above graph shows the MTS family model evolutions. Dotted edges denote "may" transitions, full edges denote "must" transitions.
MTS model of coffee machine family actually permits a user to buy a cappuccino with a dollar, something which is forbidden for its products by the variability constraint \texttt{dollar EXC unsugared\_cappuccino}
The formula expresses that every path through the MTS that starts with the insertion of a dollar, may eventually lead to an unsugared cappuccino.
Outcome of a property explained by VMC

The formula:
\[[\text{dollar}] \quad \text{EF}\ <\text{unsugared\_cappuccino}\ >\ \text{true}\]
\[\text{is FOUND\_TRUE in State } C1\]

This happens because:
- \(C1 \rightarrow C2\) \{\text{euro}(\text{optional})\}
- \(C1 \rightarrow C2\) \{\text{dollar}(\text{optional})\}

And the evolutions which satisfy the action formula \text{dollar}
also satisfy the subformula:
\[\text{EF}\ <\text{unsugared\_cappuccino}\ >\ \text{true}\]

In particular:
In state \(C2\) the subformula:
\[\text{EF}\ <\text{unsugared\_cappuccino}\ >\ \text{true}\] is Satisfied.

The formula:
\[\text{EF}\ <\text{unsugared\_cappuccino}\ >\ \text{true}\]
\[\text{is FOUND\_TRUE in State } C2\]

This happens because:
- \(C2 \rightarrow C4\) \{\text{no\_sugar}\} /\ast \ldots /\ast /

And the subformula:
\[\text{unsugared\_cappuccino}\ >\ \text{true}\]
is Satisfied in State \(C4\)

The formula:
\[<\text{unsugared\_cappuccino}\ >\ \text{true}\]
\[\text{is FOUND\_TRUE in State } C4\]

This happens because:
- \(C4 \rightarrow C11\) \{\text{unsugared\_cappuccino}(\text{optional})\}

The transition label satisfies the action expression \text{unsugared\_cappuccino}

And in State \(C11\) the subformula:
\[\text{true}\] is Satisfied.

Logic Formula for Family MTS
\[[\text{dollar}] \quad \text{EF}\ <\text{unsugared\_cappuccino}\ >\ \text{true}\]
Products of family of coffee machines derived by VMC

VMC indeed generates all 10 valid products/LTSs that are derivable from the family/MTS if the set of variability constraints is considered.
Outcomes of a property verified over products with VMC

As required, no valid product (i.e. coffee machine) can deliver an (unsugared) cappuccino upon the insertion of a dollar!
Specification of one of the products derived by VMC

Clicking on a product, VMC opens a window with its textual encoding

```
-- product71-euro-sugared_cappuccino-unsugared_cappuccino-ring_a_tone

T1 = euro.T2
T2 = sugar.T3 + no_sugar.T4
T3 = coffee.T5 + sugared_cappuccino.T6
T4 = coffee.T8 + unsugared_cappuccino.T9
T5 = pour_sugar.T8
T6 = pour_sugar.T9
T7 = pour_coffee.T11
T9 = pour_coffee.T12 + pour_milk.T13
T11 = ring_a_tone.T14
T12 = pour_milk.T11
T13 = pour_coffee.T11
T14 = take_cup.T1

net SYS = T1
```
Product Model Evolutions Chart (LTS)

View the graph in DOT format or as a PDF pdf picture or as plain SVG data.

The above graph shows the LTS product model evolutions, which by definition contains only full edges.
Outcome of a property verified over a product with VMC

This formula expresses that in this particular LTS, there exists both a path to a sugared cappuccino and a path to an unsugared cappuccino.
The formula:
(BF <sugared_cappuccino> true) and
BF <unsugared_cappuccino> true
is FOUND_TRUE in State C1

This happens because the subformula:
BF <sugared_cappuccino> true
is Satisfied in State C1

And because the subformula:
BF <unsugared_cappuccino> true
is Satisfied in State C1

The formula:
BF <sugared_cappuccino> true
is FOUND_TRUE in State C1

This happens because
C1 -> C2 {euro} /* ... */
C2 -> C3 {sugar} /* ... */
and the subformula:
<sugared_cappuccino> true
is Satisfied in State C3

The formula:
BF <unsugared_cappuccino> true
is FOUND_TRUE in State C1

This happens because
C1 -> C2 {euro} /* ... */
C2 -> C4 {no_sugar} /* ... */
and the subformula:
<unsugared_cappuccino> true
is Satisfied in State C4

Logic Formula for Product LTS
(BF <sugared_cappuccino> true) and BF <unsugared_cappuccino> true
Discussion and future work

VMC can also be used to specify and analyze only specific subsets of a product family’s valid products by applying restrictions via constraints.

Add constraint `coffee EXC dollar` to the family specification

Only European coffee machines will be derived as valid products, which can then be analyzed both as a subset and individually.

Future work required before possible application in industry

- A high-level language hiding all semantic details (investigate the relation between features and actions)
- A predefined taxonomy for exemplary logical properties (e.g. the specification patterns repository for LTL, (A)CTL, etc.)
- Scale to large, industrial-size product families (with many variation points and many features)
Discussion and future work

VMC can also be used to specify and analyze only specific subsets of a product family’s valid products by applying restrictions via constraints.

Add constraint coffee EXC dollar to the family specification

Only European coffee machines will be derived as valid products, which can then be analyzed both as a subset and individually.

Future work required before possible application in industry

- A high-level language hiding all semantic details (investigate the relation between features and actions)
- A predefined taxonomy for exemplary logical properties (e.g. the specification patterns repository for LTL, (A)CTL, etc.)
- Scale to large, industrial-size product families (with many variation points and many features)
Discussion and future work

VMC can also be used to specify and analyze only specific subsets of a product family’s valid products by applying restrictions via constraints.

Add constraint \texttt{coffee EXC dollar} to the family specification.

Only European coffee machines will be derived as valid products, which can then be analyzed both as a subset and individually.

Future work required before possible application in industry:

- A high-level language hiding all semantic details
  (investigate the relation between features and actions)
- A predefined taxonomy for exemplary logical properties
  (e.g. the specification patterns repository for LTL, (A)CTL, etc.)
- Scale to large, industrial-size product families
  (with many variation points and many features)
Publicity: consider submitting to VaMoS 2013 in Pisa

Variability Modelling of Software-Intensive Systems (VaMoS’13)
7th International Workshop

Pisa, Italy, January 23–25, 2013

⇒ http://www.vamos-workshop.net

Submission deadline: November 4, 2012

PC chairs:
- Philippe Collet (Université Nice Sophia Antipolis, France)
- Klaus Schmid (Stiftung Universität Hildesheim, Germany)

Organized by the FMT lab at the CNR research area in Pisa