Modelling and Analysing Variability in Product Families

Maurice H. ter Beek
ISTI–CNR, Pisa, Italy

joint work with
P. Asirelli
ISTI–CNR

A. Fantechi
University of Florence

S. Gnesi
ISTI–CNR

University of Seville, Spain

21 October 2011
Outline

1. Background and aim of our research activity
2. Running example: Coffee machine family
3. Logic $\mathcal{VACTL}$ interpreted over MTSs
4. Advanced variability management
5. Model checking families and products
6. Conclusions and future work
Variability management/modeling

Variability management

Key difference between SPLE and ‘conventional’ software engineering

Variability modeling

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

Key difference between SPLE and ‘conventional’ software engineering

Variability modeling

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

Variability management
Key difference between SPLE and ‘conventional’ software engineering

Variability modeling
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 our 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 derivation of products and formal verification over products and families alike

**Outcome**
- iFM'10, ACoTA@ASE’10, PLEASE@ICSE’11, FMOODS’11, SEW@FM’11, SPLC’11
- MTS: Modal Transition Systems (Larsen, Wasowski et alii)
- VACTL: variability and action-based CTL
- VMC: Variability Model Checker

**Comparison**
- FTS: Featured Transition Systems, LTL/fCTL, and SNIP/NuSMV (Classen, Heymans et alii @ ICSE’10/’11)
Aim of our 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 derivation of products and formal verification over products and families alike

**Outcome**
- iFM’10, ACoTA@ASE’10, PLEASE@ICSE’11, FMOODS’11, SEW@FM’11, SPLC’11
- MTS: Modal Transition Systems (Larsen, Wasowski et alii)
- VACTL: variability and action-based CTL
- VMC: Variability Model Checker

**Comparison**
- FTS: Featured Transition Systems, LTL/fCTL, and SNIP/NuSMV (Classen, Heymans et alii @ ICSE’10’/’11)
Aim of our 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 derivation of products and formal verification over products and families alike

Outcome

- iFM’10, ACoTA@ASE’10, PLEASE@ICSE’11, FMOODS’11, SEW@FM’11, SPLC’11
- MTS: Modal Transition Systems (Larsen, Wasowski et alii)
- .vaCTL: variability and action-based CTL
- VMC: Variability Model Checker

Comparison

- FTS: Featured Transition Systems, LTL/fCTL, and SNIP/NuSMV (Classen, Heymans et alii @ ICSE’10/’11)
Modal Transition Systems (MTSs)

Originally introduced by Larsen & Thomsen @ LICS 1988

MTSs are now an established model to formalize a product family’s
- underlying behavior, shared among all products, and
- variation points, differentiating between products

MTS is an LTS that distinguishes between may and must transitions (modeling optional or mandatory features, resp.)

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

We will model such advanced variability constraints by means of an associated set of logical formulae expressed in our variability and action-based branching-time temporal logic VA\(\text{CTL}\).
Modal Transition Systems (MTSs)

Originally introduced by Larsen & Thomsen @ LICS 1988

MTSs are now an established model to formalize a product family’s
- *underlying behavior*, shared among all products, and
- *variation points*, differentiating between products

MTS is an LTS that distinguishes between may and must transitions
(modeling *optional* or *mandatory* features, resp.)

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

We will model such advanced variability constraints by means of an
associated set of logical formulae expressed in our variability and
action-based branching-time temporal logic $\text{VACTL}$
Modal Transition Systems (MTSs)

Originally introduced by Larsen & Thomsen @ LICS 1988

MTSs are now an established model to formalize a product family’s
- *underlying behavior*, shared among all products, and
- *variation points*, differentiating between products

MTS is an LTS that distinguishes between may and must transitions
(modeling *optional* or *mandatory* features, resp.)

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

We will model such advanced variability constraints by means of an
associated set of logical formulae expressed in our variability and
action-based branching-time temporal logic $\texttt{VACTL}$
Definition of MTS

\((Q, A, \overline{q}, \rightarrow, \Rightarrow)\) is an MTS with

- **underlying LTS** \((Q, A, \overline{q}, \rightarrow, \Rightarrow)\)
- **may** transition relation \(\Rightarrow \subseteq Q \times A \times Q\) (*possible* transitions)
- **must** transition relation \(\rightarrow \subseteq Q \times A \times Q\) (*mandatory* transitions)

By definition, mandatory transitions must also be possible: \(\rightarrow \subseteq \Rightarrow\)

The set of all must paths from \(q_1\) is denoted by \(\Rightarrow\text{-path}\ (q_1)\)

\(\sigma^{\Rightarrow} = q_1a_1q_2a_2q_3\ldots\) is a must path (from \(q_1\)) if \(q_i \xrightarrow{a_i}^{\Rightarrow} q_{i+1}\ \forall i > 0\)

Subfamilies/products are obtained by preserving **all** must transitions, turning **some** may transitions into must transitions, and removing **some/all** remaining ones (resulting in MTS/LTS)
Definition of MTS

\[(Q, A, \bar{q}, \rightarrow, \rightarrow)\] is an MTS with

- **underlying** LTS \((Q, A, \bar{q}, \rightarrow)\)
- may transition relation \(\rightarrow \subseteq Q \times A \times Q\) (possible transitions)
- must transition relation \(\rightarrow \subseteq Q \times A \times Q\) (mandatory transitions)

By definition, mandatory transitions must also be possible: \(\rightarrow \subseteq \rightarrow\)

The set of all must paths from \(q_1\) is denoted by \(\square\)-path \((q_1)\)

\[\sigma^\square = q_1a_1q_2a_2q_3\cdots\] is a **must path** (from \(q_1\)) if \(q_i \xrightarrow{a_i} q_{i+1} \quad \forall \ i > 0\)

Subfamilies/products are obtained by preserving all must transitions, turning some may transitions into must transitions, and removing some/all remaining ones (resulting in MTS/LTS)
Definition of MTS

\((Q, A, \overline{q}, \rightarrow^{□}, \rightarrow^{◊})\) is an MTS with

- **underlying** LTS \((Q, A, \overline{q}, \rightarrow^{□} \cup \rightarrow^{◊})\)
- **may** transition relation \(\rightarrow^{◊} \subseteq Q \times A \times Q\) (possible transitions)
- **must** transition relation \(\rightarrow^{□} \subseteq Q \times A \times Q\) (mandatory transitions)

By definition, mandatory transitions must also be possible: \(\rightarrow^{□} \subseteq \rightarrow^{◊}\)

The set of all must paths from \(q_1\) is denoted by \(\square\text{-path}(q_1)\)

\(\sigma^{□} = q_1a_1q_2a_2q_3\cdots\) is a must path (from \(q_1\)) if \(q_i \xrightarrow{a_i}^{□} q_{i+1} \forall i > 0\)

Subfamilies/products are obtained by preserving all must transitions, turning some may transitions into must transitions, and removing some/all remaining ones (resulting in MTS/LTS)
Running example: Coffee machine family
Static & behavioral requirements of product families

Static requirements: features constituting different products

- Only accepted coins are 1€, exclusively for European products, and 1$, exclusively for Canadian products (alternative features)
- All products offer coffee (mandatory feature); cappuccino only offered by European products (excludes relation among features)
- A ringtone is rung in products that offer cappuccino (requires relation among features)

Behavioral requirements: admitted sequences of operations

- After coin insertion, user must press a button to choose whether (s)he wants sugar, after which (s)he may select a beverage
- Optionally, a ringtone is rung after delivering a beverage
- The machine returns to its idle state when the beverage is taken
Static & behavioral requirements of product families

Static requirements: features constituting different products

- Only accepted coins are 1€, exclusively for European products, and 1$, exclusively for Canadian products (alternative features)
- All products offer coffee (mandatory feature); cappuccino only offered by European products (excludes relation among features)
- A ringtone is rung in products that offer cappuccino (requires relation among features)

Behavioral requirements: admitted sequences of operations

- After coin insertion, user must press a button to choose whether (s)he wants sugar, after which (s)he may select a beverage
- Optionally, a ringtone is rung after delivering a beverage
- The machine returns to its idle state when the beverage is taken
Coffee machine family: Feature model

10 different valid products (coffee machines defined by features)

\[
\begin{align*}
\{m, s, o, b, c, \varepsilon\}, & \quad \{m, s, o, b, c, \varepsilon, r\}, & \quad \{m, s, o, b, c, \varepsilon, t\}, \\
\{m, s, o, b, c, \$, \} & \quad \{m, s, o, b, c, \$, r\}, & \quad \{m, s, o, b, c, \$, t\}, \\
\{m, s, o, b, c, \varepsilon, t, r\} & \quad \{m, s, o, b, c, \varepsilon, p, r\}, & \quad \{m, s, o, b, c, \varepsilon, p, r, t\}
\end{align*}
\]
A European coffee machine ($\{m, s, o, b, c, \varepsilon, p, r\}$)
A Canadian coffee machine ($\{m, s, o, b, c, $, r\}$)
FTS associated to feature model of family
Product derivation: MTS versus FTS

**FTS** All and only products that are correct w.r.t. the requirements are derived (price: include a feature diagram in each FTS)

**MTS** Also correctly derived products may violate constraints of the type that MTSs cannot model (cf. LTS on next slide)
Product derivation: MTS versus FTS

**FTS**  All and only products that are correct w.r.t. the requirements are derived (price: include a feature diagram in each FTS)

**MTS**  Also correctly derived products may violate constraints of the type that MTSs cannot model (cf. LTS on next slide)
A correct but not valid product LTS of MTS

Diagram showing transitions between states like coffee, cappuccino, tea, pour sugar, pour milk, pour coffee, pour tea, and actions like take cup, ring a tone, and skip.
Definition of \( \text{vACTL} \) (slight revision of MHML from paper)

Variability and action-based branching-time temporal logic

A temporal logic based on the “Hennessy-Milner logic with until”, but augmented with deontic \( O \) (obligatory) and \( P \) (permitted) operators, CTL’s path operators \( E \) and \( A \) and ACTL’s action-based Until operator, both with and without a deontic interpretation.

Syntax of \( \text{vACTL} \)

\[
\phi ::= \text{true} \mid \neg \phi \mid \phi \land \phi' \mid \langle a \rangle \phi \mid [a] \phi \mid \langle a \rangle \Box \phi \mid [a] \Box \phi \mid \mathcal{E} \pi \mid \mathcal{A} \pi
\]

\[
\pi ::= \phi \{ \varphi \} \mathcal{U} \{ \varphi' \} \phi' \mid \phi \{ \varphi \} \mathcal{U} \Box \{ \varphi' \} \phi'
\]

Thus defines state formulae \( \phi \), path formulae \( \pi \) and action formulae \( \varphi \) (boolean compositions of actions) over set of atomic actions \( \{ a, b, \ldots \} \).

\( \langle a \rangle \Box \) and \( [a] \Box \) represent the classic deontic modalities \( O \) and \( P \), resp.
Definition of \( \text{vACTL} \) (slight revision of MHML from paper)

### Variability and action-based branching-time temporal logic

A temporal logic based on the “Hennessy-Milner logic with until”, but augmented with deontic \( O \) (obligatory) and \( P \) (permitted) operators, CTL’s path operators \( E \) and \( A \) and ACTL’s action-based Until operator, both with and without a deontic interpretation.

### Syntax of \( \text{vACTL} \)

\[
\phi ::= \text{true} \mid \neg \phi \mid \phi \land \phi' \mid \langle a \rangle \phi \mid [a] \phi \mid \langle a \rangle [a] \phi \mid [a] \phi \mid E \pi \mid A \pi
\]

\[
\pi ::= \phi \{ \varphi \} U \{ \varphi' \} \phi' \mid \phi \{ \varphi \} U [a] \{ \varphi' \} \phi'
\]

Thus defines state formulae \( \phi \), path formulae \( \pi \) and action formulae \( \varphi \) (boolean compositions of actions) over set of atomic actions \( \{ a, b, \ldots \} \).

\( \langle a \rangle [a] \) and \( [a] \) represent the classic deontic modalities \( O \) and \( P \), resp.
Definition of \textsc{VACTL} (slight revision of MHML from paper)

Variability and action-based branching-time temporal logic

A temporal logic based on the “Hennessy-Milner logic with until”, but augmented with deontic \emph{O} (\textit{obligatory}) and \emph{P} (\textit{permitted}) operators, CTL’s path operators \emph{E} and \emph{A} and ACTL’s action-based Until operator, both with and without a deontic interpretation.

Syntax of \textsc{VACTL}

\[
\begin{align*}
\phi & ::= \text{true} \mid \neg \phi \mid \phi \land \phi' \mid \langle a \rangle \phi \mid [a] \phi \mid \langle a \rangle \Box \phi \mid [a] \Box \phi \mid E \pi \mid A \pi \\
\pi & ::= \phi \{ \varphi \} U \{ \varphi' \} \phi' \mid \phi \{ \varphi \} U \Box \{ \varphi' \} \phi'
\end{align*}
\]

Thus defines state formulae \(\phi\), path formulae \(\pi\) and action formulae \(\varphi\) (boolean compositions of actions) over set of atomic actions \\{a, b, \ldots\}

\(\langle a \rangle \Box\) and \([a] \Box\) represent the classic deontic modalities \emph{O} and \emph{P}, resp.
VACTL: Semantics over MTS

- \( q \models true \) always holds
- \( q \models \neg \phi \) iff not \( q \models \phi \)
- \( q \models \phi \land \phi' \) iff \( q \models \phi \) and \( q \models \phi' \)
- \( q \models \langle a \rangle \phi \) iff \( \exists q' \in Q \) such that \( q \xrightarrow{a} q' \), and \( q' \models \phi \)
- \( q \models [a] \phi \) iff \( \forall q' \in Q \) such that \( q \xrightarrow{a} q' \), we have \( q' \models \phi \)
- \( q \models \langle a \rangle \Box \phi \) iff \( \exists q' \in Q \) such that \( q \xrightarrow{a} q' \), and \( q' \models \phi \)
- \( q \models [a] \Box \phi \) iff \( \forall q' \in Q \) such that \( q \xrightarrow{a} q' \), we have \( q' \models \phi \)
- \( q \models E \pi \) iff \( \exists \sigma' \in \text{path}(q) \) such that \( \sigma' \models \pi \)
- \( q \models A \pi \) iff \( \forall \sigma' \in \text{path}(q) \) such that \( \sigma' \models \pi \)
- \( \sigma \models \phi \{ \varphi \} U \{ \varphi' \} \phi' \) iff \( \exists j \geq 1: \sigma(j) \models \phi', \sigma\{j\} \models \varphi', \) and \( \sigma(j+1) \models \phi' \), and \( \forall 1 \leq i < j: \sigma(i) \models \phi \) and \( \sigma\{i\} \models \varphi \)
- \( \sigma \models \phi \{ \varphi \} U^\Box \{ \varphi' \} \phi' \) iff \( \sigma \) is a must path \( \sigma^\Box \) and \( \sigma^\Box \models \phi \{ \varphi \} U \{ \varphi' \} \phi' \)

Abbreviations: \( EF \phi = E(true \{ true \} U \{ true \} \phi) \); \( EF^\Box \phi = E(true \{ true \} U^\Box \{ true \} \phi) \);
\( EF \{ \varphi \} true = E(true \{ true \} U \{ \varphi \} true) \); \( EF^\Box \{ \varphi \} true = E(true \{ true \} U^\Box \{ \varphi \} true) \);
\( AG \phi = \neg EF \neg \phi \); \( AG^\Box \phi = \neg EF^\Box \neg \phi \); etc.
**VACTL: Semantics over MTS**

- $q \models true$ always holds
- $q \models \neg \phi$ iff not $q \models \phi$
- $q \models \phi \land \phi'$ iff $q \models \phi$ and $q \models \phi'$
- $q \models \langle a \rangle \phi$ iff $\exists q' \in Q$ such that $q \xrightarrow{a} q'$, and $q' \models \phi$
- $q \models [a] \phi$ iff $\forall q' \in Q$ such that $q \xrightarrow{a} q'$, we have $q' \models \phi$
- $q \models \langle a \rangle \Box \phi$ iff $\exists q' \in Q$ such that $q \xrightarrow{a} q'$, and $q' \models \phi$
- $q \models [a] \Box \phi$ iff $\forall q' \in Q$ such that $q \xrightarrow{a} q'$, we have $q' \models \phi$
- $q \models E \pi$ iff $\exists \sigma' \in \text{path}(q)$ such that $\sigma' \models \pi$
- $q \models A \pi$ iff $\forall \sigma' \in \text{path}(q)$ such that $\sigma' \models \pi$
- $\sigma \models \phi \{ \varphi \} \ U \{ \varphi' \} \phi'$ iff $\exists j \geq 1: \sigma(j) \models \phi'$, $\sigma\{j\} \models \varphi'$, and $\sigma(j + 1) \models \phi'$, and $\forall 1 \leq i < j: \sigma(i) \models \phi$ and $\sigma\{i\} \models \varphi$
- $\sigma \models \phi \{ \varphi \} \ U \Box \{ \varphi' \} \phi'$ iff $\sigma$ is a must path $\sigma \Box$ and $\sigma \Box \models \phi \{ \varphi \} \ U \{ \varphi' \} \phi'$

**Abbreviations:** $EF\phi = E(\text{true} \{ \text{true} \} \ U \{ \text{true} \} \phi)$; $EF\Box \phi = E(\text{true} \{ \text{true} \} \ U \Box \{ \text{true} \} \phi)$; $EF\{ \varphi \} \text{true} = E(\text{true} \{ \text{true} \} \ U \{ \varphi \} \text{true})$; $EF\Box \{ \varphi \} \text{true} = E(\text{true} \{ \text{true} \} \ U \Box \{ \varphi \} \text{true})$; $AG\phi = \neg EF \neg \phi$; $AG\Box \phi = \neg EF \Box \neg \phi$; etc.
VACTL can complement behavioral description of MTS by expressing constraints over possible products of a family that MTS cannot model

**Template ALT:** Features F1 and F2 are *alternative*

\[(\text{EF} \{F1\} \text{ true} ∨ \text{EF} \{F2\} \text{ true}) ∧ ¬(\text{EF} \{F1\} \text{ true} ∧ \text{EF} \{F2\} \text{ true})\]

**Template EXC:** Feature F1 *excludes* feature F2

\[\text{((EF } \{F1\} \text{ true} )⇒ (AG ¬\langle F2\rangle \text{ true})) ∧ \text{((EF } \{F2\} \text{ true} )⇒ (AG ¬\langle F1\rangle \text{ true}))\]

**Template REQ:** Feature F1 *requires* feature F2

\[\text{(EF } \{F1\} \text{ true} )⇒ \text{(EF} \{F2\} \text{ true})\]

Define no full temporal ordering among the related features.

Duty of behavioral LTS/MTS model, to be verified by VACTL formulae.
VACTL can complement behavioral description of MTS by expressing constraints over possible products of a family that MTS cannot model.

**Template ALT:** Features F1 and F2 are *alternative*

$$(EF \Box \{ F1 \} \text{ true} \lor EF \Box \{ F2 \} \text{ true}) \land \neg (EF \{ F1 \} \text{ true} \land EF \{ F2 \} \text{ true})$$

**Template EXC:** Feature F1 *excludes* feature F2

$$((EF \{ F1 \} \text{ true}) \implies (AG \neg \langle F2 \rangle \text{ true})) \land ((EF \{ F2 \} \text{ true}) \implies (AG \neg \langle F1 \rangle \text{ true}))$$

**Template REQ:** Feature F1 *requires* feature F2

$$(EF \{ F1 \} \text{ true}) \implies (EF \Box \{ F2 \} \text{ true})$$

Define no full temporal ordering among the related features.

Duty of behavioral LTS/MTS model, to be verified by VACTL formulae.
VACTL can complement behavioral description of MTS by expressing constraints over possible products of a family that MTS cannot model.

**Template ALT:** Features F1 and F2 are alternative

\[(EF \Box \{ F1 \} true \lor EF \Box \{ F2 \} true) \land \neg (EF \{ F1 \} true \land EF \{ F2 \} true)\]

**Template EXC:** Feature F1 excludes feature F2

\[(EF \{ F1 \} true) \implies (AG \neg \langle F2 \rangle true) \land ((EF \{ F2 \} true) \implies (AG \neg \langle F1 \rangle true))\]

**Template REQ:** Feature F1 requires feature F2

\[(EF \{ F1 \} true) \implies (EF \Box \{ F2 \} true)\]

Define no full temporal ordering among the related features

Duty of behavioral LTS/MTS model, to be verified by VACTL formulae
VACTL can complement behavioral description of MTS by expressing constraints over possible products of a family that MTS cannot model

**Template ALT**: Features F1 and F2 are *alternative*

$$\left( EF \square \{F1\} \text{ true} \lor EF \square \{F2\} \text{ true} \right) \land \neg \left( EF \{F1\} \text{ true} \land EF \{F2\} \text{ true} \right)$$

**Template EXC**: Feature F1 *excludes* feature F2

$$\left( (EF \{F1\} \text{ true}) \Rightarrow (AG \neg \langle F2 \rangle \text{ true}) \right) \land \left( (EF \{F2\} \text{ true}) \Rightarrow (AG \neg \langle F1 \rangle \text{ true}) \right)$$

**Template REQ**: Feature F1 *requires* feature F2

$$(EF \{F1\} \text{ true}) \Rightarrow (EF \Box \{F2\} \text{ true})$$

Define no full temporal ordering among the related features

Duty of behavioral LTS/MTS model, to be verified by VACTL formulae
VACTL can complement behavioral description of MTS by expressing constraints over possible products of a family that MTS cannot model.

**Template ALT:** Features F1 and F2 are *alternative*

\[ (EF\{F1\} \text{ true} \lor EF\{F2\} \text{ true}) \land \neg (EF\{F1\} \text{ true} \land EF\{F2\} \text{ true}) \]

**Template EXC:** Feature F1 *excludes* feature F2

\[ ((EF\{F1\} \text{ true}) \implies (AG\neg\langle F2\rangle \text{ true})) \land ((EF\{F2\} \text{ true}) \implies (AG\neg\langle F1\rangle \text{ true})) \]

**Template REQ:** Feature F1 *requires* feature F2

\[ (EF\{F1\} \text{ true}) \implies (EF\{F2\} \text{ true}) \]

Define no full temporal ordering among the related features.

Duty of behavioral LTS/MTS model, to be verified by VACTL formulae.
Model checking families and products

Verify property expressed as logical formula \( \psi \) over model \( T \)

- Decide whether \( T \models \psi \), where \( \models \) is the logic’s satisfaction relation
- If \( T \not\models \psi \), then it is usually easy to generate a counterexample
- If \( T \) is finite, model checking thus reduces to a graph search

On the fly: Only a fragment of the overall state space might need to be generated and analysed to be able to produce the correct result

VMC: [http://fmtlab.isti.cnr.it/vmc/V5.1/](http://fmtlab.isti.cnr.it/vmc/V5.1/)

An on-the-fly model-checker for \( \forall \)CTL is defined as particularization of the FMC model checker for ACTL over a CCS-like input language

Recently implemented also our algorithm for product derivation

- Explore and verify product families (MTS)
- Generate all valid products, explore and verify products (LTS)
Model checking families and products

Verify property expressed as logical formula $\psi$ over model $T$
- Decide whether $T \models \psi$, where $\models$ is the logic’s satisfaction relation
- If $T \not\models \psi$, then it is usually easy to generate a counterexample
- If $T$ is finite, model checking thus reduces to a graph search

On the fly: Only a fragment of the overall state space might need to be generated and analysed to be able to produce the correct result

**VMC:** [http://fmtlab.isti.cnr.it/vmc/V5.1/](http://fmtlab.isti.cnr.it/vmc/V5.1/)

An on-the-fly model-checker for $\nu$ACTL is defined as particularization of the FMC model checker for ACTL over a CCS-like input language

Recently implemented also our algorithm for product derivation
- Explore and verify product families (MTS)
- Generate all valid products, explore and verify products (LTS)
Model checking families and products

Verify property expressed as logical formula $\psi$ over model $T$

- Decide whether $T \models \psi$, where $\models$ is the logic’s satisfaction relation
- If $T \not\models \psi$, then it is usually easy to generate a counterexample
- If $T$ is finite, model checking thus reduces to a graph search

On the fly: Only a fragment of the overall state space might need to be generated and analysed to be able to produce the correct result

**VMC:** [http://fmtlab.isti.cnr.it/vmc/V5.1/](http://fmtlab.isti.cnr.it/vmc/V5.1/)

An on-the-fly model-checker for $\mathsf{vACTL}$ is defined as particularization of the FMC model checker for $\mathsf{ACTL}$ over a CCS-like input language

Recently implemented also our algorithm for product derivation

- Explore and verify product families (MTS)
- Generate all valid products, explore and verify products (LTS)
Model checking temporal orderings

Property D: Always once a coffee has been selected, a coffee is eventually delivered

\[ AG [\text{coffee}] \ AF \{ \text{pour coffee} \} \ true \]

Property E: A coffee machine may never deliver a coffee before a coin has been inserted

\[ A [true \ \{ \neg \text{pour coffee} \} \ U \{ 1$ \lor 1\€ \} \ true] \]

<table>
<thead>
<tr>
<th>Property</th>
<th>MTS family</th>
<th>European</th>
<th>Canadian</th>
<th>LTS product</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: ALT</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>B: EXC</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>C: REQ</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>D</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>E</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>
Property D: Always once a coffee has been selected, a coffee is eventually delivered

\[ AG[\text{coffee}] \land AF\square \{\text{pour coffee}\} \land true \]

Property E: A coffee machine may never deliver a coffee before a coin has been inserted

\[ A[true \{\neg \text{pour coffee}\} \land U\square \{1\text{\$} \lor 1\text{\€}\} \land true] \]

<table>
<thead>
<tr>
<th>Property</th>
<th>MTS family</th>
<th>European</th>
<th>Canadian</th>
<th>LTS product</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: ALT</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>B: EXC</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>C: REQ</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>D</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>E</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>
Property D: Always once a coffee has been selected, a coffee is eventually delivered

\[\text{AG [coffee]} \ \text{AF} \square \ \{\text{pour coffee}\} \ \text{true}\]

Property E: A coffee machine may never deliver a coffee before a coin has been inserted

\[\text{A [true } \neg \text{pour coffee} \text{ U} \square \ \{1\$ \lor 1\€\} \ \text{true}]\]

<table>
<thead>
<tr>
<th>Property</th>
<th>MTS family</th>
<th>European</th>
<th>Canadian</th>
<th>LTS product</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: ALT</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>B: EXC</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>C: REQ</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>D</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>E</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>
MTS specification of coffee machine family in VMC

VMC v5.0b

Commands Menu
- New Model ...
- Edit Current Model
- Explore the MTS
- Generate All Products
- View Family MTS
- Welcome
- Quit

MTS specification:

T0 = may(euro).T1 + may(dollar).T1
T1 = must(sugar).T2 + must(no_sugar).T3
T2 = must(coffee).T4 + may(tea).T5 + may(cappuccino).T6
T3 = may(cappuccino).T7 + may(tea).T8 + must(coffee).T9
T4 = must(pour_sugar).T9
T5 = must(pour_sugar).T8
T6 = must(pour_sugar).T7
T7 = must(pour_milk).T10 + must(pour_coffee).T11
T8 = must(pour_tea).T12
T9 = must(pour_coffee).T12
T10 = must(pour_coffee).T12
T11 = must(pour_milk).T12
T12 = may(no_ring).T13 + may(ring_a_tone).T13
T13 = must(cup_taken).T0

net SYS = T0

Constraints {
  euro ALT dollar
  dollar EXC cappuccino
  cappuccino REQ ring_a_tone
}

SoC/L/Products

AF <ring_a_tone> true
Result of ‘a coffee machine always eventually rings a tone’

<table>
<thead>
<tr>
<th>Product</th>
<th>Formula evaluates</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>product11.txt</td>
<td>Formula evaluates</td>
<td>FALSE</td>
</tr>
<tr>
<td>product12.txt</td>
<td>Formula evaluates</td>
<td>TRUE</td>
</tr>
<tr>
<td>product13.txt</td>
<td>Formula evaluates</td>
<td>TRUE</td>
</tr>
<tr>
<td>product20.txt</td>
<td>Formula evaluates</td>
<td>FALSE</td>
</tr>
<tr>
<td>product21.txt</td>
<td>Formula evaluates</td>
<td>TRUE</td>
</tr>
<tr>
<td>product22.txt</td>
<td>Formula evaluates</td>
<td>TRUE</td>
</tr>
<tr>
<td>product26.txt</td>
<td>Formula evaluates</td>
<td>FALSE</td>
</tr>
<tr>
<td>product27.txt</td>
<td>Formula evaluates</td>
<td>TRUE</td>
</tr>
<tr>
<td>product28.txt</td>
<td>Formula evaluates</td>
<td>TRUE</td>
</tr>
<tr>
<td>product33.txt</td>
<td>Formula evaluates</td>
<td>TRUE</td>
</tr>
<tr>
<td>product34.txt</td>
<td>Formula evaluates</td>
<td>TRUE</td>
</tr>
<tr>
<td>product39.txt</td>
<td>Formula evaluates</td>
<td>TRUE</td>
</tr>
<tr>
<td>product40.txt</td>
<td>Formula evaluates</td>
<td>TRUE</td>
</tr>
<tr>
<td>product42.txt</td>
<td>Formula evaluates</td>
<td>TRUE</td>
</tr>
<tr>
<td>product43.txt</td>
<td>Formula evaluates</td>
<td>TRUE</td>
</tr>
<tr>
<td>product44.txt</td>
<td>Formula evaluates</td>
<td>FALSE</td>
</tr>
<tr>
<td>product45.txt</td>
<td>Formula evaluates</td>
<td>TRUE</td>
</tr>
<tr>
<td>product46.txt</td>
<td>Formula evaluates</td>
<td>TRUE</td>
</tr>
<tr>
<td>product48.txt</td>
<td>Formula evaluates</td>
<td>TRUE</td>
</tr>
<tr>
<td>product49.txt</td>
<td>Formula evaluates</td>
<td>TRUE</td>
</tr>
<tr>
<td>product53.txt</td>
<td>Formula evaluates</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

SoCL/Products

AF <ring_a_tone> true
VMC can be used to automatically derive, from an MTS description of a product family and an associated set of VACTL formulae expressing further constraints for this family, all its valid products (i.e. a set of LTS descriptions of products, each one correct w.r.t. all VACTL constraints).

VMC can also be used to experiment with products, using VACTL to verify further (temporal) constraints, etc.

Future work required before possible application in industry:

- A high-level language hiding all semantic details (investigate relation between features and actions)
- A predefined taxonomy for properties specified in VACTL (like specification patterns repository for LTL, (A)CTL, etc.)
- Scale to large, industrial-size product families (with many variation points and many features)
VMC: Variability Model Checker

VMC can be used to automatically derive, from an MTS description of a product family **and** an associated set of $\nu\text{ACTL}$ formulae expressing further constraints for this family, **all** its valid products (i.e. a set of LTS descriptions of products, each one correct w.r.t. all $\nu\text{ACTL}$ constraints)

VMC can also be used to experiment with products, using $\nu\text{ACTL}$ to verify further (temporal) constraints, etc.

**Future work required before possible application in industry**

- A high-level language hiding all semantic details (investigate relation between features and actions)
- A predefined taxonomy for properties specified in $\nu\text{ACTL}$ (like specification patterns repository for LTL, (A)CTL, etc.)
- Scale to large, industrial-size product families (with many variation points and many features)
VMC: Variability Model Checker

VMC can be used to automatically derive, from an MTS description of a product family and an associated set of \( \nuACTL \) formulae expressing further constraints for this family, all its valid products (i.e. a set of LTS descriptions of products, each one correct w.r.t. all \( \nuACTL \) constraints).

VMC can also be used to experiment with products, using \( \nuACTL \) to verify further (temporal) constraints, etc.

Future work required before possible application in industry

- A high-level language hiding all semantic details (investigate relation between features and actions)
- A predefined taxonomy for properties specified in \( \nuACTL \) (like specification patterns repository for LTL, (A)CTL, etc.)
- Scale to large, industrial-size product families (with many variation points and many features)
Variability Modelling of Software-Intensive Systems (VaMoS’12)
6th International Workshop
Leipzig, Germany, January 25–27, 2012

⇒ http://www.vamos-workshop.net

Submission: October 30, 2011
Notification: November 27, 2011

with keynotes by:

- Paulo Borba, Centro de Informatica, Universidade Federal de Pernambuco, BR
- Charles Krueger, BigLever Software, Inc., USA