

# 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

- 1 Background and aim of our research activity
- 2 Running example: Coffee machine family
- 3 Logic vACTL interpreted over MTSs
- 4 Advanced variability management
- 5 Model checking families and products
- 6 Conclusions and future work

## 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

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 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 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 VACTL

# Definition of MTS

$(Q, A, \bar{q}, \rightarrow_{\square}, \rightarrow_{\diamond})$  is an MTS with

- *underlying* LTS  $(Q, A, \bar{q}, \rightarrow_{\square} \cup \rightarrow_{\diamond})$
- *may* transition relation  $\rightarrow_{\diamond} \subseteq Q \times A \times Q$  (*possible* transitions)
- *must* transition relation  $\rightarrow_{\square} \subseteq Q \times A \times Q$  (*mandatory* transitions)

By definition, mandatory transitions must also be possible:  $\rightarrow_{\square} \subseteq \rightarrow_{\diamond}$

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

$\sigma^{\square} = q_1 a_1 q_2 a_2 q_3 \dots$  is a *must path* (from  $q_1$ ) if  $q_i \xrightarrow{a_i}_{\square} q_{i+1} \forall i > 0$

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

$(Q, A, \bar{q}, \rightarrow_{\square}, \rightarrow_{\diamond})$  is an MTS with

- *underlying* LTS  $(Q, A, \bar{q}, \rightarrow_{\square} \cup \rightarrow_{\diamond})$
- *may* transition relation  $\rightarrow_{\diamond} \subseteq Q \times A \times Q$  (*possible* transitions)
- *must* transition relation  $\rightarrow_{\square} \subseteq Q \times A \times Q$  (*mandatory* transitions)

By definition, mandatory transitions must also be possible:  $\rightarrow_{\square} \subseteq \rightarrow_{\diamond}$

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

$\sigma^{\square} = q_1 a_1 q_2 a_2 q_3 \dots$  is a *must path* (from  $q_1$ ) if  $q_i \xrightarrow{a_i}_{\square} q_{i+1} \forall i > 0$

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

$(Q, A, \bar{q}, \rightarrow_{\square}, \rightarrow_{\diamond})$  is an MTS with

- *underlying* LTS  $(Q, A, \bar{q}, \rightarrow_{\square} \cup \rightarrow_{\diamond})$
- *may* transition relation  $\rightarrow_{\diamond} \subseteq Q \times A \times Q$  (*possible* transitions)
- *must* transition relation  $\rightarrow_{\square} \subseteq Q \times A \times Q$  (*mandatory* transitions)

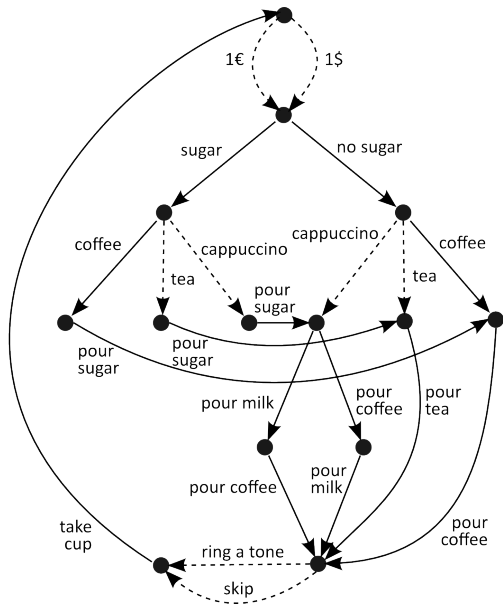
By definition, mandatory transitions must also be possible:  $\rightarrow_{\square} \subseteq \rightarrow_{\diamond}$

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

$\sigma^{\square} = q_1 a_1 q_2 a_2 q_3 \dots$  is a *must path* (from  $q_1$ ) if  $q_i \xrightarrow{a_i}_{\square} q_{i+1} \forall i > 0$

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

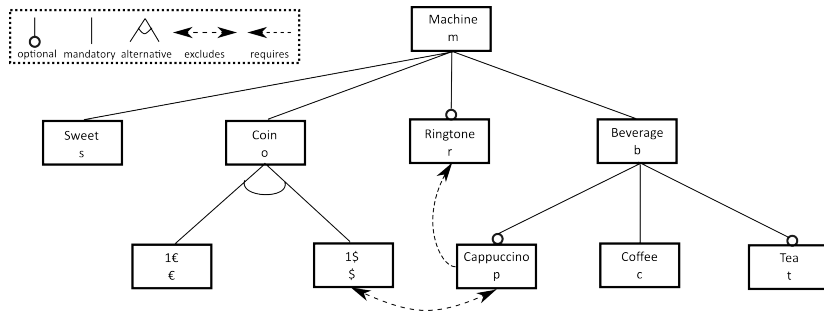
## Static requirements: features constituting different products

- 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

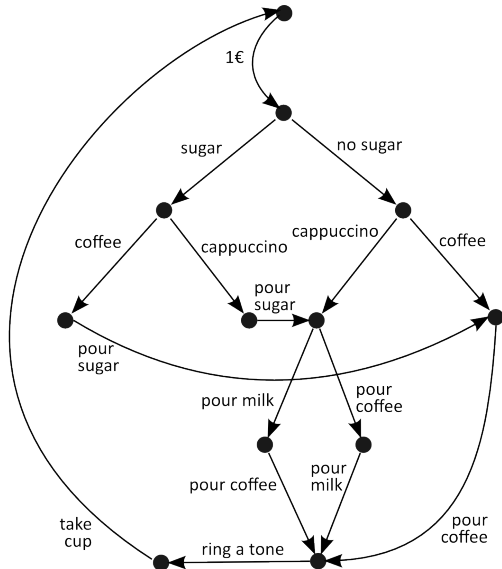
# Coffee machine family: Feature model



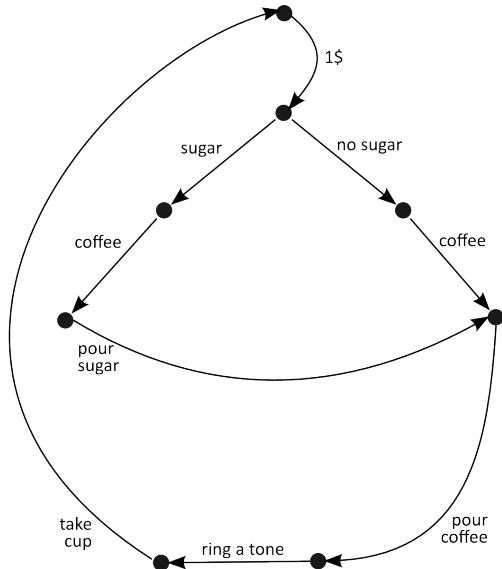
10 different valid products (coffee machines defined by features)

$\{\{m, s, o, b, c, \epsilon\}, \{m, s, o, b, c, \epsilon, r\}, \{m, s, o, b, c, \epsilon, t\},$   
 $\{m, s, o, b, c, \$\}, \{m, s, o, b, c, \$, r\}, \{m, s, o, b, c, \$, t\},$   
 $\{m, s, o, b, c, \epsilon, t, r\}, \{m, s, o, b, c, \epsilon, p, r\},$   
 $\{m, s, o, b, c, \$, t, r\}, \{m, s, o, b, c, \epsilon, p, r, t\}\}$

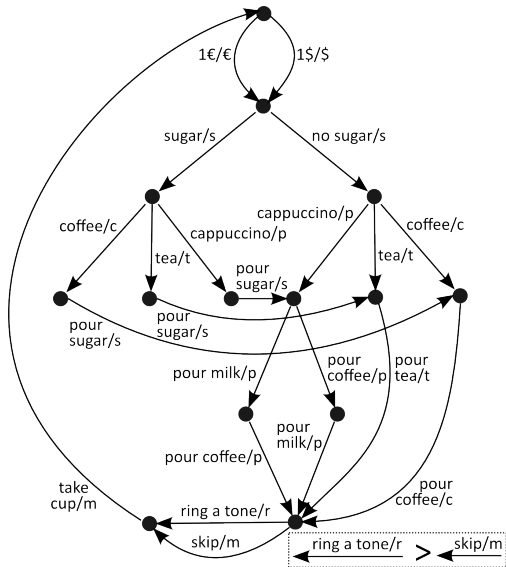
# A European coffee machine ( $\{m, s, o, b, c, \epsilon, p, r\}$ )



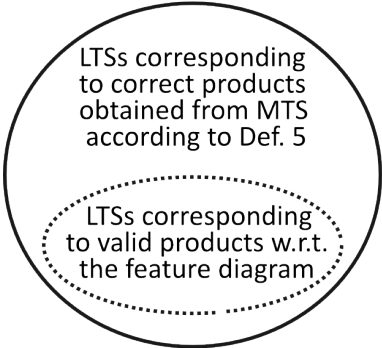
# A Canadian coffee machine ( $\{m, s, o, b, c, \$, r\}$ )



# FTS associated to feature model of family

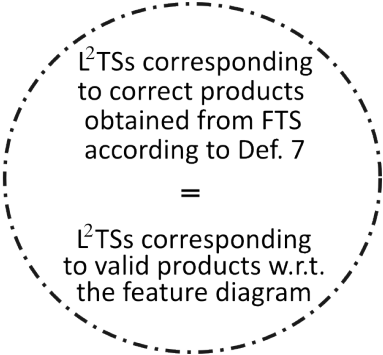


# Product derivation: MTS versus FTS



LTSs corresponding to correct products obtained from MTS according to Def. 5

LTSs corresponding to valid products w.r.t. the feature diagram



$L^2$ TSs corresponding to correct products obtained from FTS according to Def. 7

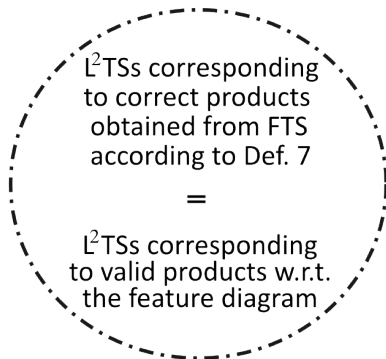
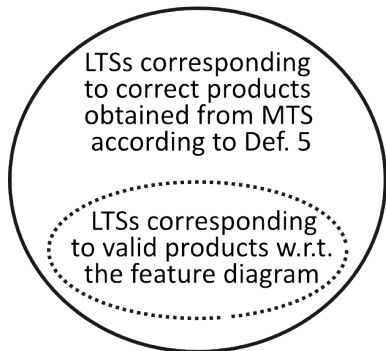
=

$L^2$ TSs corresponding to valid products w.r.t. the feature diagram

**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)

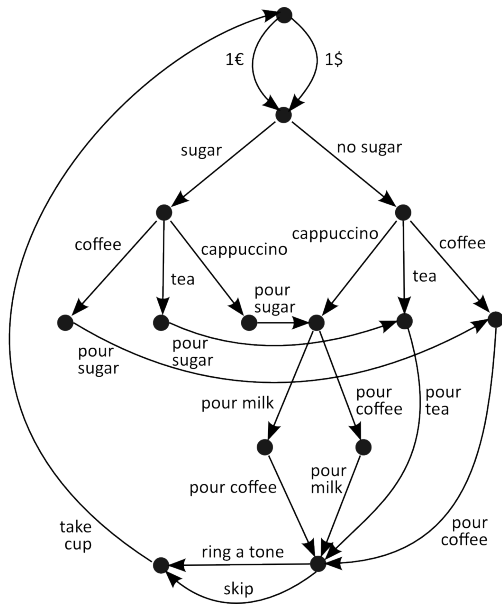
# Product derivation: MTS versus FTS



**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



# Definition of 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 VACTL

$$\begin{aligned}\phi &::= \text{true} \mid \neg \phi \mid \phi \wedge \phi' \mid \langle a \rangle \phi \mid [a] \phi \mid \langle a \rangle^\square \phi \mid [a]^\square \phi \mid E \pi \mid A \pi \\ \pi &::= \phi \{ \varphi \} U \{ \varphi' \} \phi' \mid \phi \{ \varphi \} U^\square \{ \varphi' \} \phi'\end{aligned}$$

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

$\langle a \rangle^\square$  and  $[a]^\square$  represent the classic deontic modalities  $O$  and  $P$ , resp.

## 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 VACTL

$$\begin{aligned}\phi &::= \text{true} \mid \neg\phi \mid \phi \wedge \phi' \mid \langle a \rangle \phi \mid [a] \phi \mid \langle a \rangle^\square \phi \mid [a]^\square \phi \mid E\pi \mid A\pi \\ \pi &::= \phi \{ \varphi \} U \{ \varphi' \} \phi' \mid \phi \{ \varphi \} U^\square \{ \varphi' \} \phi'\end{aligned}$$

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

$\langle a \rangle^\square$  and  $[a]^\square$  represent the classic deontic modalities  $O$  and  $P$ , resp.

# Definition of 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 VACTL

$$\begin{aligned}\phi &::= \text{true} \mid \neg\phi \mid \phi \wedge \phi' \mid \langle a \rangle \phi \mid [a] \phi \mid \langle a \rangle^\square \phi \mid [a]^\square \phi \mid E\pi \mid A\pi \\ \pi &::= \phi \{ \varphi \} U \{ \varphi' \} \phi' \mid \phi \{ \varphi \} U^\square \{ \varphi' \} \phi'\end{aligned}$$

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

$\langle a \rangle^\square$  and  $[a]^\square$  represent the classic deontic modalities  $O$  and  $P$ , resp.

# VACTL: Semantics over MTS

- $q \models \text{true}$  always holds
- $q \models \neg \phi$  iff not  $q \models \phi$
- $q \models \phi \wedge \phi'$  iff  $q \models \phi$  and  $q \models \phi'$
- $q \models \langle a \rangle \phi$  iff  $\exists q' \in Q$  such that  $q \xrightarrow{a}_{\diamond} q'$ , and  $q' \models \phi$
- $q \models [a] \phi$  iff  $\forall q' \in Q$  such that  $q \xrightarrow{a}_{\diamond} q'$ , we have  $q' \models \phi$
- $q \models \langle a \rangle^{\square} \phi$  iff  $\exists q' \in Q$  such that  $q \xrightarrow{a}_{\square} q'$ , and  $q' \models \phi$
- $q \models [a]^{\square} \phi$  iff  $\forall q' \in Q$  such that  $q \xrightarrow{a}_{\square} q'$ , 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  $\sigma\{i\} \models \varphi$
- $\sigma \models \phi \{ \varphi \} U^{\square} \{ \varphi' \} \phi'$  iff  $\sigma$  is a must path  $\sigma^{\square}$  and  $\sigma^{\square} \models \phi \{ \varphi \} U \{ \varphi' \} \phi'$

Abbreviations:  $EF\phi = E(\text{true} \{ \text{true} \} U \{ \text{true} \} \phi)$ ;  $EF^{\square}\phi = E(\text{true} \{ \text{true} \} U^{\square} \{ \text{true} \} \phi)$ ;  
 $EF\{\varphi\} \text{true} = E(\text{true} \{ \text{true} \} U \{\varphi\} \text{true})$ ;  $EF^{\square}\{\varphi\} \text{true} = E(\text{true} \{ \text{true} \} U^{\square} \{\varphi\} \text{true})$ ;  
 $AG\phi = \neg EF\neg\phi$ ;  $AG^{\square}\phi = \neg EF^{\square}\neg\phi$ ; etc.

# VACTL: Semantics over MTS

- $q \models \text{true}$  always holds
- $q \models \neg \phi$  iff not  $q \models \phi$
- $q \models \phi \wedge \phi'$  iff  $q \models \phi$  and  $q \models \phi'$
- $q \models \langle a \rangle \phi$  iff  $\exists q' \in Q$  such that  $q \xrightarrow{a}_{\diamond} q'$ , and  $q' \models \phi$
- $q \models [a] \phi$  iff  $\forall q' \in Q$  such that  $q \xrightarrow{a}_{\diamond} q'$ , we have  $q' \models \phi$
- $q \models \langle a \rangle^{\square} \phi$  iff  $\exists q' \in Q$  such that  $q \xrightarrow{a}_{\square} q'$ , and  $q' \models \phi$
- $q \models [a]^{\square} \phi$  iff  $\forall q' \in Q$  such that  $q \xrightarrow{a}_{\square} q'$ , 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  $\sigma\{i\} \models \varphi$
- $\sigma \models \phi \{ \varphi \} U^{\square} \{ \varphi' \} \phi'$  iff  $\sigma$  is a must path  $\sigma^{\square}$  and  $\sigma^{\square} \models \phi \{ \varphi \} U \{ \varphi' \} \phi'$

Abbreviations:  $EF\phi = E(\text{true} \{ \text{true} \} U \{ \text{true} \} \phi)$ ;  $EF^{\square}\phi = E(\text{true} \{ \text{true} \} U^{\square} \{ \text{true} \} \phi)$ ;  
 $EF\{\varphi\} \text{true} = E(\text{true} \{ \text{true} \} U \{\varphi\} \text{true})$ ;  $EF^{\square}\{\varphi\} \text{true} = E(\text{true} \{ \text{true} \} U^{\square} \{\varphi\} \text{true})$ ;  
 $AG\phi = \neg EF \neg \phi$ ;  $AG^{\square}\phi = \neg EF^{\square} \neg \phi$ ; etc.

# Advanced variability management

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*

$$(EF^{\square} \{F1\} \text{ true} \vee EF^{\square} \{F2\} \text{ true}) \wedge \neg(EF \{F1\} \text{ true} \wedge EF \{F2\} \text{ true})$$

Template EXC: Feature F1 *excludes* feature F2

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

Template REQ: Feature F1 *requires* feature F2

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

Define no full temporal ordering among the related features

Duty of behavioral LTS/MTS model, to be verified by VACTL formulae

# Advanced variability management

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*

$$(EF^{\square} \{F1\} true \vee EF^{\square} \{F2\} true) \wedge \neg (EF \{F1\} true \wedge EF \{F2\} true)$$

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

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

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

$$(EF \{F1\} true) \implies (EF^{\square} \{F2\} true)$$

Define no full temporal ordering among the related features

Duty of behavioral LTS/MTS model, to be verified by VACTL formulae

# Advanced variability management

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*

$$(EF^{\square} \{F1\} true \vee EF^{\square} \{F2\} true) \wedge \neg(EF \{F1\} true \wedge EF \{F2\} true)$$

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

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

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

$$(EF \{F1\} true) \implies (EF^{\square} \{F2\} true)$$

Define no full temporal ordering among the related features

Duty of behavioral LTS/MTS model, to be verified by VACTL formulae

# Advanced variability management

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*

$$(EF^{\square} \{F1\} \text{ true} \vee EF^{\square} \{F2\} \text{ true}) \wedge \neg(EF \{F1\} \text{ true} \wedge EF \{F2\} \text{ true})$$

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

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

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

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

Define no full temporal ordering among the related features

Duty of behavioral LTS/MTS model, to be verified by VACTL formulae

# Advanced variability management

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*

$$(EF^{\square} \{F1\} true \vee EF^{\square} \{F2\} true) \wedge \neg(EF \{F1\} true \wedge EF \{F2\} true)$$

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

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

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

$$(EF \{F1\} true) \implies (EF^{\square} \{F2\} true)$$

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/>

An on-the-fly model-checker for vACTL 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/>

An on-the-fly model-checker for vACTL 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/>

An on-the-fly model-checker for vACTL 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 temporal orderings

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

$$AG [coffee] AF^{\square} \{pour\ coffee\} \ true$$

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

$$A [true \ \{\neg\ pour\ coffee\} U^{\square} \{1\$ \vee 1\epsilon\} \ true]$$

Property	MTS family	European	Canadian	LTS product
A: ALT	<i>false</i>	<i>true</i>	<i>true</i>	<i>false</i>
B: EXC	<i>false</i>	<i>true</i>	<i>true</i>	<i>false</i>
C: REQ	<i>false</i>	<i>true</i>	<i>true</i>	<i>true</i>
D	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>
E	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>

# Model checking temporal orderings

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

$$AG [coffee] AF^{\square} \{pour\ coffee\} \ true$$

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

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

Property	MTS family	European	Canadian	LTS product
A: ALT	<i>false</i>	<i>true</i>	<i>true</i>	<i>false</i>
B: EXC	<i>false</i>	<i>true</i>	<i>true</i>	<i>false</i>
C: REQ	<i>false</i>	<i>true</i>	<i>true</i>	<i>true</i>
D	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>
E	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>

# Model checking temporal orderings

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

$$AG [coffee] AF^{\square} \{pour\ coffee\} \ true$$

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

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

Property	MTS family	European	Canadian	LTS product
A: ALT	<i>false</i>	<i>true</i>	<i>true</i>	<i>false</i>
B: EXC	<i>false</i>	<i>true</i>	<i>true</i>	<i>false</i>
C: REQ	<i>false</i>	<i>true</i>	<i>true</i>	<i>true</i>
D	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>
E	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>

# MTS specification of coffee machine family in VMC

The screenshot shows the VMC v5.0b web interface. The browser address bar displays `http://fmtlab.isti.cnr.it/vmc/V5.0/vmc.html`. The interface includes a sidebar with a "Commands Menu" containing buttons for "New Model ...", "Edit Current Model", "Explore the MTS", "Generate All Products", "View Family MTS", "Welcome", and "Quit". The main content area displays a list of constraints for a coffee machine family, followed by a "SoCL/Products" section with a text input field and two buttons: "Check The Formula" and "Explain the Result".

VMC v5.0b

Commands Menu

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

```
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
}
```

SoCL/Products

AF <ring\_a\_tone> true

Check The Formula   Explain the Result

# Result of 'a coffee machine always eventually rings a tone'

The screenshot shows a web browser window with the URL `http://fmtlab.isti.cnr.it/vmc/V5.0/vmc.html`. The page title is "VMC v5.0b". On the left, there is a "Commands Menu" with buttons for "New Model ...", "Edit Current Model", "Explore the MTS", "Generate All Products", "View Family MTS", "Welcome", and "Quit". Below the menu is a decorative image of a coffee machine. The main content area is titled "Evaluation of formula 'AF true' on products" and contains a list of products with their corresponding formula evaluation results:

<a href="#">product11.txt</a>	Formula evaluates	FALSE
<a href="#">product12.txt</a>	Formula evaluates	TRUE
<a href="#">product13.txt</a>	Formula evaluates	TRUE
<a href="#">product20.txt</a>	Formula evaluates	FALSE
<a href="#">product21.txt</a>	Formula evaluates	TRUE
<a href="#">product22.txt</a>	Formula evaluates	TRUE
<a href="#">product26.txt</a>	Formula evaluates	FALSE
<a href="#">product27.txt</a>	Formula evaluates	TRUE
<a href="#">product28.txt</a>	Formula evaluates	TRUE
<a href="#">product33.txt</a>	Formula evaluates	TRUE
<a href="#">product34.txt</a>	Formula evaluates	TRUE
<a href="#">product39.txt</a>	Formula evaluates	TRUE
<a href="#">product40.txt</a>	Formula evaluates	TRUE
<a href="#">product42.txt</a>	Formula evaluates	TRUE
<a href="#">product43.txt</a>	Formula evaluates	TRUE
<a href="#">product44.txt</a>	Formula evaluates	FALSE
<a href="#">product45.txt</a>	Formula evaluates	TRUE
<a href="#">product46.txt</a>	Formula evaluates	TRUE
<a href="#">product48.txt</a>	Formula evaluates	TRUE
<a href="#">product49.txt</a>	Formula evaluates	TRUE
<a href="#">product53.txt</a>	Formula evaluates	FALSE

Below the list, the text "SoCL/Products" is displayed. In a white box, the formula `AF <ring_a_tone> true` is shown. To the right of this box are two buttons: "Check The Formula" and "Explain the Result".

# VMC: Variability Model Checker

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 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 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)

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