

From Featured Transition Systems to
Modal Transition Systems (with variability constraints)
and a Variability Model Checker

Maurice ter Beek and Stefania Gnesi

Formal Methods and Tools lab, ISTI-CNR, Pisa, Italy

Modeling and Analysing Variability in Product Families
UNIFI, 14/3/16

- 1 Background: Software Product Line Engineering (4/3/16)
- 2 Formal Methods and Analysis in Software Product Line Engineering
 - Featured Transition Systems
 - Modal Transition Systems with variability constraints
 - Transformation: from feature constraints to action constraints
- 3 v-CTL: variability-aware, action-based CTL
 - Preservation of formulae in v-CTL[□]/v-CTL^{Live}[□]
- 4 VMC: Variability Model Checker
 - Family-based and product-based verification with VMC
 - A value-passing modal process algebra

Formal methods and tools in SPLE

Computer-aided analysis of feature models

- Traditionally: focus on modelling/analysing structural constraints
- But: software systems often embedded/distributed/safety-critical
- Important: model/analyse also behaviour (e.g. quality assurance)

Goal: rigorously establish critical requirements of (software) systems

⇒ lift success stories from single product/system engineering to SPLE

Widely used behavioural SPL models with dedicated model checkers

- Modal Transition Systems (MTS) with variability constraints

Fantechi, Gnesi @ SPLC'08, Asirelli et al. @ iFM'10, SPLC'11, ter Beek et al. @ *JLAMP*, 2016

Variability Model Checker VMC

ter Beek et al. @ FM'12, SPLC'12, SPLat'14

- Featured Transition Systems (FTS)

Classen et al. @ ICSE'10, *IEEE TSE*, 2013, *Sci. Comput. Program.*, 2014

SNIP, ProVeLines, NuSMV extension

Classen et al. @ ICSE'11, *Int. J. Softw. Tools Technol. Transf.*, 2012, Cordy et al. @ SPLC'13

Recall (atomic propositions used for verification purposes):

De Nicola, Vaandrager © *J. ACM*, 1995

A **Doubly-Labelled Transition System** (L^2TS) is a sextuple $(Q, A, \bar{q}, \rightarrow, AP, L)$ with states Q , actions A , initial state \bar{q} , transitions $\rightarrow \subseteq Q \times A \times Q$, atomic propositions AP , and labeling function $L : S \rightarrow 2^{AP}$

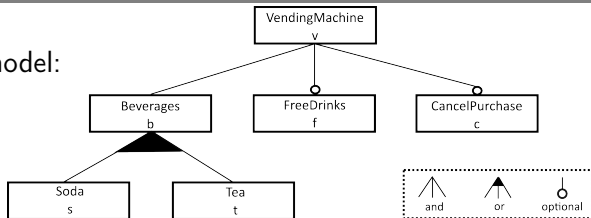
An FTS adds to this a feature model and feature expressions:

Classen et al. © ICSE'10, *IEEE TSE*, 2013, *Sci. Comput. Program.*, 2014

A **Featured Transition System** (FTS) is an octuple $(Q, A, \bar{q}, \rightarrow, AP, L, FD, \gamma)$ with *underlying* L^2TS $(Q, A, \bar{q}, \rightarrow, AP, L)$, feature diagram FD over a set \mathbb{F} of features, and total function $\gamma : \rightarrow \rightarrow \mathbb{B}(\mathbb{F})$ labelling each transition with a **feature expression**, i.e. a Boolean expression over the features

FTS of example SPL: a vending machine

Feature model:

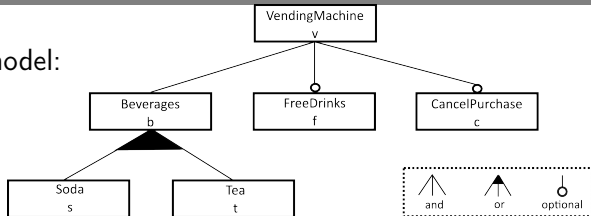


12 valid products

e.g. $\{v, b, s, t\}$, $\{v, b, s, c\}$

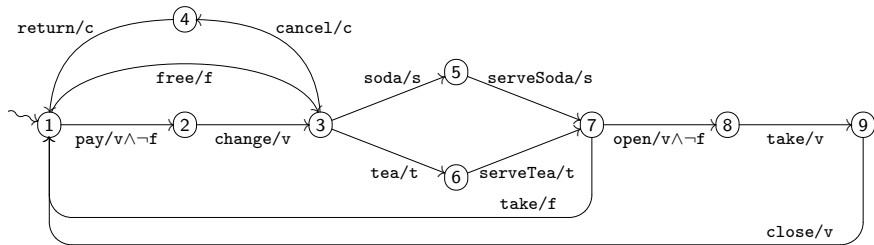
FTS of example SPL: a vending machine

Feature model:



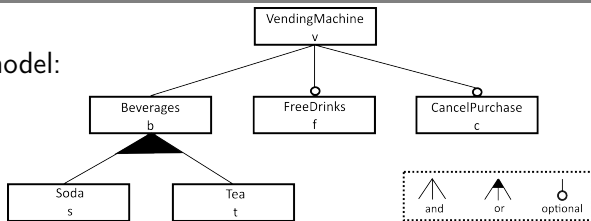
FTS of 12 valid products (LTSs)

e.g. $\{v, b, s, t\}$, $\{v, b, s, c\}$

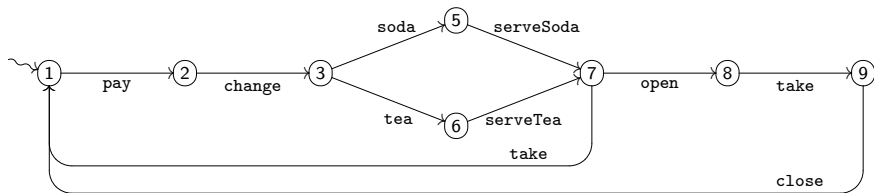


FTS of example SPL: a vending machine

Feature model:

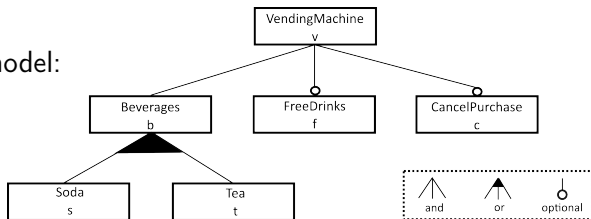


e.g. $\{v, b, s, t\}$, $\{v, b, s, c\}$

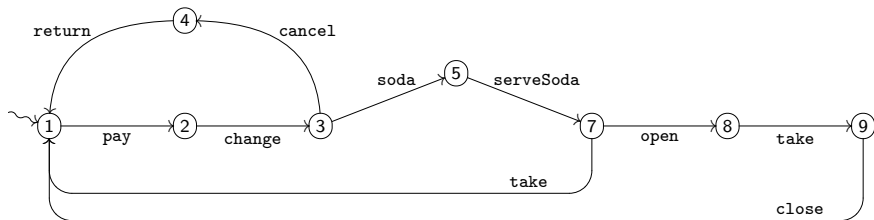


FTS of example SPL: a vending machine

Feature model:



e.g. $\{v, b, s, t\}$, $\{v, b, s, c\}$



Main ingredient: Modal Transition Systems (MTS)

- LTS distinguishing possible (**may**) and required (**must**) transitions

Larsen, Thomsen © LICS'88

- Recognized as a useful model to describe in a **compact** way the possible **behaviour** of all the products (LTS) of a product family

Fischbein, Uchitel, Braberman © ROSATEA'06, Fantechi, Gnesi © ESEC/FSE'07, SPLC'08

- MTS cannot model variability constraints regarding **alternative** features, nor regarding **requires/excludes** inter-feature relations, resulting in several variants and extensions

Larsen et al. © ESOP'07, Lauenroth et al. © ASE'09

- Our solution: add a set of **variability constraints** to the MTS to be able to decide which derivable products (LTS) are valid ones

ter Beek, Fantechi, Gnesi, Mazzanti © *JLAMP*, 2016

MTS with...

Recall:

A **Labelled Transition System** (LTS) is a quadruple $(Q, A, \bar{q}, \rightarrow)$ with states Q , actions A , initial state \bar{q} and transitions $\rightarrow \subseteq Q \times A \times Q$

Next we define MTS with variability constraints:

A **Modal Transition System** (MTS) is a quintuple $(Q, A, \bar{q}, \rightarrow_{\square}, \rightarrow_{\diamond})$ such that $(Q, A, \bar{q}, \rightarrow_{\square} \cup \rightarrow_{\diamond})$ is an LTS, called its underlying LTS

An MTS has two distinct transition relations

1. **may** transition relation $\rightarrow_{\diamond} \subseteq Q \times A \times Q$: **possible** transitions
2. **must** transition relation $\rightarrow_{\square} \subseteq Q \times A \times Q$: **required** transitions

By definition, any required transition is also possible: $\rightarrow_{\square} \subseteq \rightarrow_{\diamond}$
(denote $--\rightarrow \equiv \rightarrow_{\diamond} \setminus \rightarrow_{\square}$: **optional** transitions)

... variability constraints (accepted by VMC)

Variability constraints of form **ALT**ernative, **EXC**ludes, **REQ**uires, etc.

a_1 **ALT** \cdots **ALT** a_n : precisely one among the $n \geq 2$ actions a_1, \dots, a_n is reachable in \mathcal{L} (i.e. is the label of a reachable transition)

b_1 **OR** \cdots **OR** b_n , where b_i is either a_i or $\neg a_i$: at least one among the conditions on $n \geq 2$ actions b_1, \dots, b_n holds, i.e. $b_i = a_i$ is reachable in \mathcal{L} or $b_i = \neg a_i$ is not reachable in \mathcal{L}

a_1 **EXC** a_2 : at most one of the actions a_1 and a_2 is reachable in \mathcal{L}

a_1 **REQ** a_2 : action a_2 is reachable in \mathcal{L} whenever a_1 is reachable in \mathcal{L}

a_1 **REQ** (a_2 **ALT** \cdots **ALT** a_n) : precisely one among the $n \geq 2$ actions a_2, \dots, a_n is reachable in \mathcal{L} if a_1 is reachable in \mathcal{L}

a_1 **REQ** (a_2 **OR** \cdots **OR** a_n) : at least one among the $n \geq 2$ actions a_2, \dots, a_n is reachable in \mathcal{L} if a_1 is reachable in \mathcal{L}

Derive products (implemented in VMC)

A **product LTS** is obtained from a family MTS in the following way:

1. include **all** (reachable) must transitions and
2. include **subset** of the (reachable) optional transitions, remove rest
3. satisfy assumptions of **coherence** and **consistency**
4. satisfy **variability constraints**

⇒ Each selection gives rise to a different variant

Let $(Q, A, \bar{q}, \delta^\diamond, \delta^\square)$ be a coherent MTS, i.e. $\exists \xrightarrow{a} \implies \nexists \xrightarrow{a}$

The set $\{\mathcal{P}_i = (Q_i, A, \bar{q}, \delta_i) \mid i > 0\}$ of **product LTS** is obtained by considering each pair of $Q_i \subseteq Q$ and $\delta_i \subseteq \delta^\diamond \cup \delta^\square$ to be defined s.t.

1. every $q \in Q_i$ is reachable in \mathcal{P}_i from \bar{q} via transitions from δ_i
2. there exists no $(q, a, q') \in \delta^\square \setminus \delta_i$ such that $q \in Q_i$
3. LTS is consistent: both $\xrightarrow{a} \rightsquigarrow \xrightarrow{a}$ and $\cancel{\xrightarrow{a}}$ not allowed

Derive products (implemented in VMC)

A **product LTS** is obtained from a family MTS in the following way:

1. include **all** (reachable) must transitions and
2. include **subset** of the (reachable) optional transitions, remove rest
3. satisfy assumptions of **coherence** and **consistency**
4. satisfy **variability constraints**

⇒ Each selection gives rise to a different variant

Let $(Q, A, \bar{q}, \delta^\diamond, \delta^\square)$ be a coherent MTS, i.e. $\exists \xrightarrow{a} \implies \nexists \xrightarrow{a}$

The set $\{\mathcal{P}_i = (Q_i, A, \bar{q}, \delta_i) \mid i > 0\}$ of **product LTS** is obtained by considering each pair of $Q_i \subseteq Q$ and $\delta_i \subseteq \delta^\diamond \cup \delta^\square$ to be defined s.t.

1. every $q \in Q_i$ is reachable in \mathcal{P}_i from \bar{q} via transitions from δ_i
2. there exists no $(q, a, q') \in \delta^\square \setminus \delta_i$ such that $q \in Q_i$
3. LTS is consistent: both $\xrightarrow{a} \rightsquigarrow \xrightarrow{a}$ and ~~\xrightarrow{a}~~ not allowed

VMC: Variability Model Checker

Input: specification of an MTS in process-algebraic terms,
together with a set of variability constraints

VMC offers two kinds of behavioural variability analyses (more later):

1. The actual set of all valid product behaviour can explicitly be generated and the resulting LTS can all be verified against one and the same logic property (expressed in Action-based CTL)

De Nicola, Vaandrager © J. ACM, 1995

2. A logic property (expressed in **variability-aware** ACTL, seen before) can directly be verified against the MTS, relying on the fact that under certain syntactic conditions validity over the MTS implies validity of the same property for all its products

ter Beek, Fantechi, Gnesi, Mazzanti © JLAMP, 2016

VMC (v6.3, released in November 2015) is freely usable online:

<http://fmt.isti.cnr.it/vmc/>

VMC: Variability Model Checker

Input: specification of an MTS in process-algebraic terms,
together with a set of variability constraints

VMC offers two kinds of behavioural variability analyses (more later):

1. The actual set of all valid product behaviour can explicitly be generated and the resulting LTS can all be verified against one and the same logic property (expressed in Action-based CTL)

De Nicola, Vaandrager © *J. ACM*, 1995

2. A logic property (expressed in **variability-aware** ACTL, seen before) can directly be verified against the MTS, relying on the fact that under certain syntactic conditions validity over the MTS implies validity of the same property for all its products

ter Beek, Fantechi, Gnesi, Mazzanti © *JLAMP*, 2016

VMC (v6.3, released in November 2015) is freely usable online:

<http://fmt.isti.cnr.it/vmc/>

VMC: Variability Model Checker

Input: specification of an MTS in process-algebraic terms,
together with a set of variability constraints

VMC offers two kinds of behavioural variability analyses (more later):

1. The actual set of all valid product behaviour can explicitly be generated and the resulting LTS can all be verified against one and the same logic property (expressed in Action-based CTL)
2. A logic property (expressed in **variability-aware** ACTL, seen before) can directly be verified against the MTS, relying on the fact that under certain syntactic conditions validity over the MTS implies validity of the same property for all its products

De Nicola, Vaandrager © *J. ACM*, 1995

ter Beek, Fantechi, Gnesi, Mazzanti © *JLAMP*, 2016

VMC (v6.3, released in November 2015) is freely usable online:

<http://fmt.isti.cnr.it/vmc/>

VMC: Variability Model Checker

Input: specification of an MTS in process-algebraic terms,
together with a set of variability constraints

VMC offers two kinds of behavioural variability analyses (more later):

1. The actual set of all valid product behaviour can explicitly be generated and the resulting LTS can all be verified against one and the same logic property (expressed in Action-based CTL)
2. A logic property (expressed in **variability-aware** ACTL, seen before) can directly be verified against the MTS, relying on the fact that under certain syntactic conditions validity over the MTS implies validity of the same property for all its products

De Nicola, Vaandrager © *J. ACM*, 1995

ter Beek, Fantechi, Gnesi, Mazzanti © *JLAMP*, 2016

VMC (v6.3, released in November 2015) is freely usable online:

<http://fmt.isti.cnr.it/vmc/>

Crux: from feature constraints to action constraints

From feature model: A requires C



From FTS to MTS (naive): a REQ c



From FTS to MTS (solution):

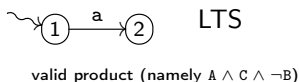
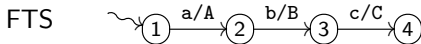
1. new action \forall feature (to handle more complex feature expressions)
2. dummy transition \forall action (to verify constraints, **ignored when model checking**)



! Consistency guarantees that whenever a c-labelled may transition from the initial state is preserved in this LTS, then also any other **reachable** c-labelled may transition must be preserved

Crux: from feature constraints to action constraints

From feature model: A requires C



From FTS to MTS (naive): a REQ c



From FTS to MTS (solution):

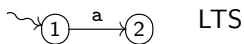
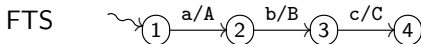
1. new action \forall feature (to handle more complex feature expressions)
2. dummy transition \forall action (to verify constraints, ignored when model checking)



! Consistency guarantees that whenever a c-labelled may transition from the initial state is preserved in this LTS, then also any other **reachable** c-labelled may transition must be preserved

Crux: from feature constraints to action constraints

From feature model: A requires C



valid product (namely $A \wedge C \wedge \neg B$)

From FTS to MTS (naive): a REQ c



~~valid product~~ (violates a REQ c)

From FTS to MTS (solution):

1. new action \forall feature (to handle more complex feature expressions)
2. dummy transition \forall action (to verify constraints, **ignored when model checking**)

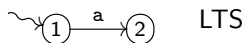
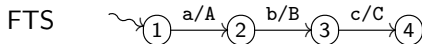


valid product (satisfies a REQ c)

! Consistency guarantees that whenever a c-labelled may transition from the initial state is preserved in this LTS, then also any other **reachable** c-labelled may transition must be preserved

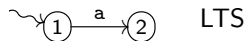
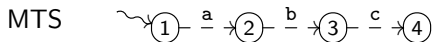
Crux: from feature constraints to action constraints

From feature model: A requires C



valid product (namely $A \wedge C \wedge \neg B$)

From FTS to MTS (naive): a REQ c



~~valid product~~ (violates a REQ c)

From FTS to MTS (solution):

1. new action \forall feature (to handle more complex feature expressions)
2. dummy transition \forall action (to verify constraints, ignored when model checking)

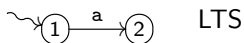
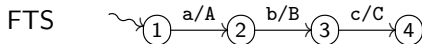


valid product (satisfies a REQ c)

! Consistency guarantees that whenever a c-labelled may transition from the initial state is preserved in this LTS, then also any other **reachable** c-labelled may transition must be preserved

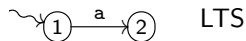
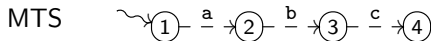
Crux: from feature constraints to action constraints

From feature model: A requires C



valid product (namely $A \wedge C \wedge \neg B$)

From FTS to MTS (naive): a REQ c



~~valid product~~ (violates a REQ c)

From FTS to MTS (solution):

1. new action \forall feature (to handle more complex feature expressions)
2. dummy transition \forall action (to verify constraints, ignored when model checking)

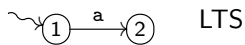
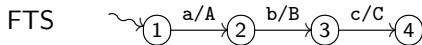


valid product (satisfies a REQ c)

! Consistency guarantees that whenever a c-labelled may transition from the initial state is preserved in this LTS, then also any other *reachable* c-labelled may transition must be preserved

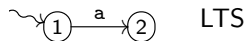
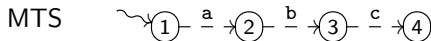
Crux: from feature constraints to action constraints

From feature model: A requires C



valid product (namely $A \wedge C \wedge \neg B$)

From FTS to MTS (naive): a REQ c



~~valid product~~ (violates a REQ c)

From FTS to MTS (solution):

1. new action \forall feature (to handle more complex feature expressions)
2. dummy transition \forall action (to verify constraints, **ignored when model checking**)

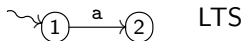
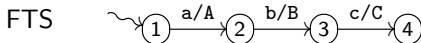


valid product (satisfies a REQ c)

! Consistency guarantees that whenever a c-labelled may transition from the initial state is preserved in this LTS, then also any other **reachable** c-labelled may transition must be preserved

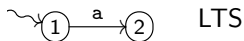
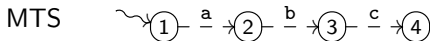
Cruz: from feature constraints to action constraints

From feature model: A requires C



valid product (namely $A \wedge C \wedge \neg B$)

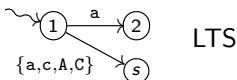
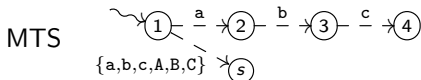
From FTS to MTS (naive): a REQ c



~~valid product~~ (violates a REQ c)

From FTS to MTS (solution):

1. new action \forall feature (to handle more complex feature expressions)
2. dummy transition \forall action (to verify constraints, **ignored when model checking**)

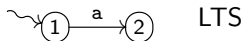
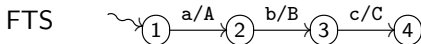


valid product (satisfies a REQ c)

! Consistency guarantees that whenever a c-labelled may transition from the initial state is preserved in this LTS, then also any other **reachable** c-labelled may transition must be preserved

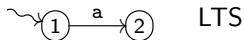
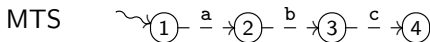
Crux: from feature constraints to action constraints

From feature model: A requires C



valid product (namely $A \wedge C \wedge \neg B$)

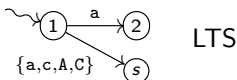
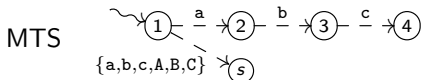
From FTS to MTS (naive): a REQ c



~~valid product~~ (violates a REQ c)

From FTS to MTS (solution):

1. new action \forall feature (to handle more complex feature expressions)
2. dummy transition \forall action (to verify constraints, **ignored when model checking**)

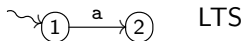
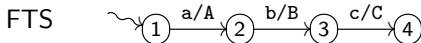


valid product (satisfies a REQ c)

! Consistency guarantees that whenever a c-labelled may transition from the initial state is preserved in this LTS, then also any other **reachable** c-labelled may transition must be preserved

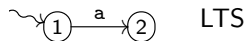
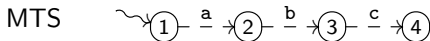
Crux: from feature constraints to action constraints

From feature model: A requires C



valid product (namely $A \wedge C \wedge \neg B$)

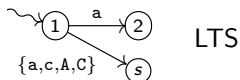
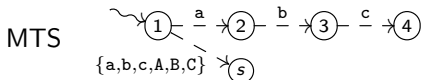
From FTS to MTS (naive): a REQ c



~~valid product~~ (violates a REQ c)

From FTS to MTS (solution):

1. new action \forall feature (to handle more complex feature expressions)
2. dummy transition \forall action (to verify constraints, **ignored when model checking**)



valid product (satisfies a REQ c)

! Consistency guarantees that whenever a c-labelled may transition from the initial state is preserved in this LTS, then also any other **reachable** c-labelled may transition must be preserved

Model Transformation (1/4)

Step 1: definition of valid products in terms of features

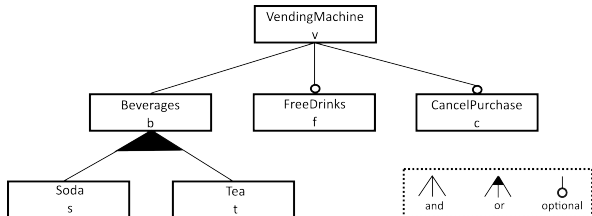
Translate feature model in a set of variability constraints on features

Model Transformation (1/4)

Step 1: definition of valid products in terms of features

Translate feature model in a set of variability constraints on features

Constraints {
Soda OR Tea
}



Model Transformation (2/4)

Step 2: definition of valid products in terms of actions

Define logic formula ' $a \leftrightarrow \phi$ ' for each transition $\xrightarrow{a/\phi}$ in FTS (feature expressions not translatable in VMC format are transformed in CNF, which is the reason for which the n -ary OR construct $b_1 \text{ OR } \dots \text{ OR } b_n$ can now contain either $b_i = a_i$ or its negation $b_i = \sim a_i$)

```
Constraints {  
    free IFF FreeDrinks  
    pay ALT FreeDrinks  
    cancel IFF CancelPurchase  
    soda IFF Soda  
    tea IFF Tea  
    takeFree IFF FreeDrinks  
    open ALT FreeDrinks  
}
```

Model Transformation (2/4)

Step 2: definition of valid products in terms of actions

Define logic formula ' $a \leftrightarrow \phi$ ' for each transition $\xrightarrow{a/\phi}$ in FTS (feature expressions not translatable in VMC format are transformed in CNF, which is the reason for which the n -ary OR construct $b_1 \text{ OR } \dots \text{ OR } b_n$ can now contain either $b_i = a_i$ or its negation $b_i = \sim a_i$)

```
Constraints {  
    free IFF FreeDrinks  
    pay ALT FreeDrinks  
    cancel IFF CancelPurchase  
    soda IFF Soda  
    tea IFF Tea  
    takeFree IFF FreeDrinks  
    open ALT FreeDrinks  
}
```

Model Transformation (3/4)

Step 3: definition of valid products in MTS and variability constraints

1. Define FTS process algebraically, using 'a(may)' for each 'a/ ϕ '
2. Add dummy transition for each 'may' action / non-mandatory feature
3. Create a new initial process with special action behaviour leading to FTS encoding, whereas signature leads to dummy transitions

```
Behaviour = behaviour.T1
```

```
T1 = pay(may).T2 + free(may).T3
```

```
T2 = change.T3
```

```
...
```

```
T9 = close.T1
```

```
Signature = signature.(
```

```
  free(may).nil + pay(may).nil + ... + open(may).nil +
```

```
  FreeDrinks(may).nil + ... + Tea(may).nil
```

```
)
```

```
VMCmodel = Behaviour + Signature
```

Model Transformation (3/4)

Step 3: definition of valid products in MTS and variability constraints

1. Define FTS process algebraically, using 'a(may)' for each 'a/ ϕ '
2. Add dummy transition for each 'may' action / non-mandatory feature
3. Create a new initial process with special action behaviour leading to FTS encoding, whereas signature leads to dummy transitions

```
Behaviour = behaviour.T1
```

```
T1 = pay(may).T2 + free(may).T3
```

```
T2 = change.T3
```

```
...
```

```
T9 = close.T1
```

```
Signature = signature.(
```

```
  free(may).nil + pay(may).nil + ... + open(may).nil +
```

```
  FreeDrinks(may).nil + ... + Tea(may).nil
```

```
)
```

```
VMCmodel = Behaviour + Signature
```

Model Transformation (3/4)

Step 3: definition of valid products in MTS and variability constraints

1. Define FTS process algebraically, using 'a(may)' for each 'a/ ϕ '
2. Add dummy transition for each 'may' action / non-mandatory feature
3. Create a new initial process with special action behaviour leading to FTS encoding, whereas signature leads to dummy transitions

```
Behaviour = behaviour.T1
```

```
T1 = pay(may).T2 + free(may).T3
```

```
T2 = change.T3
```

```
...
```

```
T9 = close.T1
```

```
Signature = signature.(
```

```
  free(may).nil + pay(may).nil + ... + open(may).nil +
```

```
  FreeDrinks(may).nil + ... + Tea(may).nil
```

```
)
```

```
VMCmodel = Behaviour + Signature
```

Model Transformation (3/4)

Step 3: definition of valid products in MTS and variability constraints

1. Define FTS process algebraically, using 'a(may)' for each 'a/ ϕ '
2. Add dummy transition for each 'may' action / non-mandatory feature
3. Create a new initial process with special action behaviour leading to FTS encoding, whereas signature leads to dummy transitions

```
Behaviour = behaviour.T1
```

```
T1 = pay(may).T2 + free(may).T3
```

```
T2 = change.T3
```

```
...
```

```
T9 = close.T1
```

```
Signature = signature.(
```

```
  free(may).nil + pay(may).nil + ... + open(may).nil +
```

```
  FreeDrinks(may).nil + ... + Tea(may).nil
```

```
)
```

```
VMCmodel = Behaviour + Signature
```

Model Transformation (3/4)

Step 3: definition of valid products in MTS and variability constraints

1. Define FTS process algebraically, using 'a(may)' for each 'a/ ϕ '
2. Add dummy transition for each 'may' action / non-mandatory feature
3. Create a new initial process with special action behaviour leading to FTS encoding, whereas signature leads to dummy transitions

```
Behaviour = behaviour.T1
```

```
T1 = pay(may).T2 + free(may).T3
```

```
T2 = change.T3
```

```
...
```

```
T9 = close.T1
```

```
Signature = signature.(
```

```
  free(may).nil + pay(may).nil + ... + open(may).nil +
```

```
  FreeDrinks(may).nil + ... + Tea(may).nil
```

```
)
```

```
VMCmodel = Behaviour + Signature
```

Model Transformation (4/4)

Step 4: definition of live action sets and introduction of must transitions

We perform two optimisations for model-checking purposes

1. the explicit definition of additional live action sets
2. the transformation of may transitions into must transitions

These help VMC to understand a model's live states and thereby take full advantage of the specificities of variability-aware ACTL

```
Constraints {  
  free OR pay  
  cancel OR soda OR tea  
  takeFree OR open  
}
```



Model Transformation (4/4)

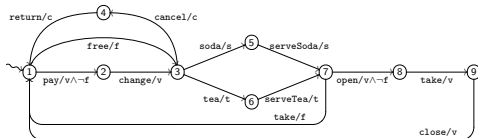
Step 4: definition of live action sets and introduction of must transitions

We perform two optimisations for model-checking purposes

1. the explicit definition of additional live action sets
2. the transformation of may transitions into must transitions

These help VMC to understand a model's live states and thereby take full advantage of the specificities of variability-aware ACTL

```
Constraints {  
  free OR pay  
  cancel OR soda OR tea  
  takeFree OR open  
}
```



Soundness of model transformation

Given FTS S and MTS S'

Let $\llbracket S \rrbracket$ denote set of valid product configurations for S

Let $\text{FTS}(S)$ and $\text{MTS}(S')$ denote the set of LTS products of S and S'

Theorem

Let S be FTS and S' be MTS obtained by the model transformation

- 1. \exists bijection between $\llbracket S \rrbracket$ and $\text{MTS}(S')$ such that each p in $\llbracket S \rrbracket$ is associated to an LTS that contains a (dummy) transition with label F for each feature F in p and no transitions labelled with a feature not in p*
- 2. $\text{FTS}(S)$ and the set of LTS obtained by omitting the dummy transitions from the LTS in $\text{MTS}(S')$ are equal*

Proof.

Sketch in SEFM'15 paper. Full proof in forthcoming journal paper. \square

Recall v-ACTL: variability-aware, action-based CTL

Action formulae ψ , state formulae ϕ , path formulae π

$\psi ::= \text{true} \mid a \mid \cancel{a(e)} \mid \neg\psi \mid \psi \wedge \psi$

ter Beek, Gnesi, Mazzanti © PSI'14

$\phi ::= \text{true} \mid \neg\phi \mid \phi \wedge \phi \mid \langle\chi\rangle\phi \mid [\chi]\phi \mid E\pi \mid A\pi \mid \mu Y.\phi(Y) \mid \nu Y.\phi(Y)$

$\pi ::= [\phi \{ \chi \} U \{ \chi' \} \phi'] \mid [\phi \{ \chi \} U \phi'] \mid [\phi \{ \chi \} W \{ \chi' \} \phi'] \mid [\phi \{ \chi \} W \phi'] \mid X \{ \chi \} \phi \mid F\phi \mid F \{ \chi \} \phi \mid G\phi$

$\langle\chi\rangle^\square \phi$: a next state exists, reachable by a **must** transition executing an action satisfying χ , in which ϕ holds

$[\chi]^\square \phi$: in all next states reachable by a **must** transition executing an action satisfying χ , ϕ holds

$F^\square \{ \chi \} \phi$: there exists a future state, reached by an action satisfying χ , in which ϕ holds and all transitions until that state are **must** transitions

Preservation of formulae in v-ACTL[□]/v-ACTLive[□]

v-ACTL[□]/v-ACTLive[□]:

$$\begin{aligned} \phi ::= & \text{false} \mid \text{true} \mid \phi \wedge \phi \mid \phi \vee \phi \mid [\psi]\phi \mid \langle \psi \rangle^{\square} \phi \mid \\ & EF^{\square} \phi \mid EF^{\square} \{\psi\} \phi \mid AF^{\square} \phi \mid AF^{\square} \{\psi\} \phi \mid AG \phi \mid \\ & AF \phi \mid AF \{\psi\} \phi \end{aligned}$$

any formula that is true for MTS, is also true for all products (LTSs)

v-ACTL⁻:

$$\begin{aligned} \chi ::= & \text{false} \mid \text{true} \mid \chi \wedge \chi \mid \chi \vee \chi \mid \langle \psi \rangle \chi \mid \\ & EF \chi \mid EF \{\psi\} \chi \end{aligned}$$

any formula that is false for MTS, is also false for all products (LTSs)

Preservation of formulae in $v\text{-ACTL}^\square/v\text{-ACTLive}^\square$

$v\text{-ACTL}^\square/v\text{-ACTLive}^\square$:

$$\begin{aligned} \phi ::= & \textit{false} \mid \textit{true} \mid \phi \wedge \phi \mid \phi \vee \phi \mid [\psi]\phi \mid \langle \psi \rangle^\square \phi \mid \\ & EF^\square \phi \mid EF^\square \{\psi\} \phi \mid AF^\square \phi \mid AF^\square \{\psi\} \phi \mid AG \phi \mid \\ & AF \phi \mid AF \{\psi\} \phi \end{aligned}$$

any formula that is true for MTS, is also true for all products (LTSs)

$v\text{-ACTL}^-$:

$$\begin{aligned} \chi ::= & \textit{false} \mid \textit{true} \mid \chi \wedge \chi \mid \chi \vee \chi \mid \langle \psi \rangle \chi \mid \\ & EF \chi \mid EF \{\psi\} \chi \end{aligned}$$

any formula that is false for MTS, is also false for all products (LTSs)

Live states use SPL-specific information

$$S \models \phi \Rightarrow S_p \models \phi \quad \forall \text{product LTS } S_p \text{ of MTS } S$$

Recall: all (reachable) must transitions are preserved ($\langle \rangle^\square$, $[\]^\square$, F^\square)

Live action sets define live states (not occur as final in any product)



Assume
a OR b



LTS

In any product in which p occurs,
 p has at least one outgoing transition

$\Rightarrow p$ is a live state, since a OR b gives rise to a live action set $\{a, b\}$

Live states use SPL-specific information

$$S \models \phi \Rightarrow S_p \models \phi \quad \forall \text{ product LTS } S_p \text{ of MTS } S$$

Recall: all (reachable) must transitions are preserved ($\langle \rangle^\square$, $[\]^\square$, F^\square)

Live action sets define live states (not occur as final in any product)



Assume
a OR b



LTS



In any product in which p occurs,
 p has at least one outgoing transition

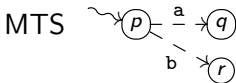
$\Rightarrow p$ is a live state, since a OR b gives rise to a live action set $\{a, b\}$

Live states use SPL-specific information

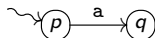
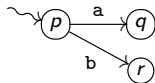
$$S \models \phi \Rightarrow S_p \models \phi \quad \forall \text{product LTS } S_p \text{ of MTS } S$$

Recall: all (reachable) must transitions are preserved ($\langle \rangle^\square$, $[\]^\square$, F^\square)

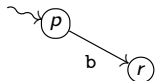
Live action sets define **live states** (not occur as final in any product)



Assume
a OR b



LTS



In any product in which p occurs,
 p has at least one outgoing transition

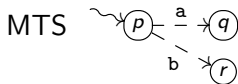
$\Rightarrow p$ is a live state, since a OR b gives rise to a live action set $\{a, b\}$

Live states use SPL-specific information

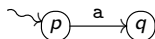
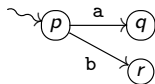
$$S \models \phi \Rightarrow S_p \models \phi \quad \forall \text{product LTS } S_p \text{ of MTS } S$$

Recall: all (reachable) must transitions are preserved ($\langle \rangle^\square$, $[\]^\square$, F^\square)

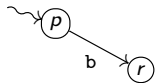
Live action sets define **live states** (not occur as final in any product)



Assume
a OR b



LTS



In any product in which p occurs,
 p has at least one outgoing transition

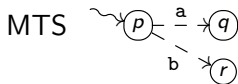
$\Rightarrow p$ is a live state, since a OR b gives rise to a live action set $\{a, b\}$

Live states use SPL-specific information

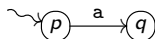
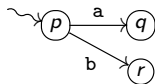
$$S \models \phi \Rightarrow S_p \models \phi \quad \forall \text{product LTS } S_p \text{ of MTS } S$$

Recall: all (reachable) must transitions are preserved ($\langle \rangle^\square$, $[\]^\square$, F^\square)

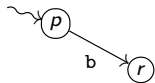
Live action sets define **live states** (not occur as final in any product)



Assume
a OR b



LTS



In any product in which p occurs,
 p has at least one outgoing transition

$\Rightarrow p$ is a live state, since a OR b gives rise to a live action set $\{a, b\}$

Recall VMC: Variability Model Checker

ter Beek, Mazzanti, Sulova @ FM'12, ter Beek, Gnesi, Mazzanti @ SPLC'12

VMC builds on optimization of UMC (input: UML state machines)

ter Beek, Fantechi, Gnesi, Mazzanti @ *Sci. Comput. Program.*, 2011

VMC: bounded, on-the-fly model checking, providing **explanations**

↘ Biere, Cimatti, Clarke, Zhu @ TACAS'99

VMC accepts as input a specification in (value-passing) modal process algebra, possibly with additional variability constraints

- interactively explore the model (MTS)
- derive and explore (all) the model's valid variants (LTSs)
- visualize the model/variants graphically as MTS/LTSs
- verify v-ACTL properties over MTSs/LTSs
- interactively explain why a property is (not) satisfied

Model checking of v-ACTL formulae over MTS can be achieved in a complexity that is **linear** w.r.t. the state space size

Vending Machine: family-based verification

VMC notifies whenever preservation of a verification result is applicable

VMC v6.1

● ● ● ● ●

Edit Model

View Current Model


Explore the MTS

Draw Family MTS

Generate Products

Welcome

Quit



Kandisky 1908

The Formula:

```
[behaviour] not E[true {not tea} U {serveTea}true]
```

is **TRUE**

The formula holds for ALL the MTS products

(states generated= 11, computations fragments generated= 10, evaluation time= 0.000644000 sec.)

ACT-UCTL-SoCL-VACTL

```
[behaviour] not E [true {not tea} U {serveTea} true]
```

It is not possible that serveTea occurs without being preceded by tea


Vending Machine: product-based verification

VMC lists for each product the action labels of all may transitions that have been preserved (as must transitions) in that product

VMC v6.1

● ● ● ● ●

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



Kandinsky 1908

Evaluation of the formula "[behaviour] [pay] AF {takePaid} true" on all family products

product_1+Soda+open+pay+soda	Formula evaluates	TRUE
product_10+CancelPurchase+FreeDrinks+Tea+cancel+free+takeFree+tea	Formula evaluates	TRUE
product_11+FreeDrinks+Soda+Tea+free+soda+takeFree+tea	Formula evaluates	TRUE
product_12+CancelPurchase+FreeDrinks+Soda+Tea+cancel+free+soda+takeFree+tea	Formula evaluates	TRUE
product_2+CancelPurchase+Soda+cancel+open+pay+soda	Formula evaluates	FALSE
product_3+Tea+open+pay+tea	Formula evaluates	TRUE
product_4+CancelPurchase+Tea+cancel+open+pay+tea	Formula evaluates	FALSE
product_5+Soda+Tea+open+pay+soda+tea	Formula evaluates	TRUE
product_6+CancelPurchase+Soda+Tea+cancel+open+pay+soda+tea	Formula evaluates	FALSE
product_7+FreeDrinks+Soda+free+soda+takeFree	Formula evaluates	TRUE
product_8+CancelPurchase+FreeDrinks+Soda+cancel+free+soda+takeFree	Formula evaluates	TRUE
product_9+FreeDrinks+Tea+free+takeFree+tea	Formula evaluates	TRUE

Logic Formula for all Products

```
[behaviour] [pay] AF {takePaid} true
```

Whenever pay occurs, eventually takePaid occurs

Specification of the family of coffee machines in VMC

VMC v6.2



Edit Model

View Current Model

Explore the MTS

Draw Family MTS

Modelcheck MTS ...

Generate Products

Welcome

Quit



Kandisky 1908

```
1 T1 = euro(may).T2 + dollar(may).T2
2 T2 = sugar.T3 + no_sugar.T4
3 T3 = tea(may).T5 + coffee(may).T6 + cappuccino(may).T7
4 T4 = cappuccino(may).T8 + coffee(may).T9 + tea(may).T10
5 T5 = pour_sugar.T10
6 T6 = pour_sugar.T9
7 T7 = pour_sugar.T8
8 T8 = pour_milk.T9
9 T9 = pour_espresso(may).T11 + pour_regular(may).T11
10 T10 = pour_tea.T11
11 T11 = take_cup.T1
12
13 net SYS = T1
14
15 Constraints {
16   euro ALT dollar
17   cappuccino OR coffee OR tea
18   dollar EXC tea
19   cappuccino REQ coffee
20   coffee REQ (pour_espresso ALT pour_regular)
21 }
22 |
```

Products of the family of coffee machines derived by VMC

VMC v6.2



Edit Model

View Current Model

Explore the MTS

Draw Family MTS

Modelcheck MTS ...

Generate Products

Modelcheck Products

Welcome

Quit



Kandisky 1908

Valid Products of the Family

[product_01+euro+tea](#)

[product_02+coffee+euro+pour_espresso](#)

[product_03+coffee+euro+pour_regular](#)

[product_04+coffee+euro+pour_espresso+tea](#)

[product_05+coffee+euro+pour_regular+tea](#)

[product_06+cappuccino+coffee+euro+pour_espresso](#)

[product_07+cappuccino+coffee+euro+pour_regular](#)

[product_08+cappuccino+coffee+euro+pour_espresso+tea](#)

[product_09+cappuccino+coffee+euro+pour_regular+tea](#)

[product_10+coffee+dollar+pour_espresso](#)

[product_11+coffee+dollar+pour_regular](#)

[product_12+cappuccino+coffee+dollar+pour_espresso](#)

[product_13+cappuccino+coffee+dollar+pour_regular](#)

A product of the family of coffee machines derived by VMC

FMC v6.1



Edit Model

View Current Model

Explore the Model

Draw Abstract L2TS

Draw Abstract Traces

Modelcheck L2TS ...

Welcome

Quit




Kandisky 1908

```
1 -----
2 -- product _06+cappuccino+coffee+euro+pour_espresso
3 -----
4
5 T1 = euro().T2 + nil
6 T2 = sugar().T3 + no_sugar().T4
7 T3 = nil + coffee().T6 + cappuccino().T7
8 T4 = cappuccino().T8 + coffee().T9 + nil
9 T5 = pour_sugar().T10
10 T6 = pour_sugar().T9
11 T7 = pour_sugar().T8
12 T8 = pour_milk().T9
13 T9 = pour_espresso().T11 + nil
14 T10 = pour_tea().T11
15 T11 = take_cup().T1
16
17 net SYS = (T1)
18 |
```

A formula verified by VMC over the family of coffee machines

VMC v6.2
● ● ● ● ●

Edit Model
View Current Model
Explore the MTS
Draw Family MTS
Generate Products
Welcome
Quit



Kandinsky 1908

The Formula:
 $AG [sugar] AF \{pour_sugar\} true$

is TRUE

The formula holds for ALL the valid MTS products
(states generated= 11, computations fragments generated= 29, evaluation time= 0.000452000 sec.)

ACTL-UCTL-SocL-vACTL
AG [sugar] AF {pour_sugar} true

Check The Formula Explain the Result

It is always the case that whenever sugar is chosen, eventually sugar is poured

A formula verified by VMC over the family of coffee machines

VMC v6.2

● ● ● ● ●

Edit Model

View Current Model


Explore the MTS

Draw Family MTS

Generate Products

Welcome

Quit



Kandisky 1908

The Formula:

$AG ((not <sugar and not may> true) or <no_sugar and not may> true)$

is **TRUE**

Even if the formula is TRUE for the MTS, its validity is not necessarily preserved by the MTS products

(states generated= 11, computations fragments generated= 47, evaluation time= 0.000890000 sec.)

ACTL-UCTL-SocL-vACTL

$AG ((not <sugar>\# true) or (<no_sugar>\# true))$

Check The Formula

Explain the Result

*It is always the case that whenever sugar can be chosen,
also no sugar can be chosen*

A formula verified by VMC over all products of the family

VMC v6.2



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



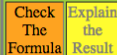
Kandinsky 1908

Evaluation of the formula "[dollar] EF {cappuccino} true" on all family products

product_01+euro+tea	Formula evaluates	TRUE
product_02+coffee+euro+pour_espresso	Formula evaluates	TRUE
product_03+coffee+euro+pour_regular	Formula evaluates	TRUE
product_04+coffee+euro+pour_espresso+tea	Formula evaluates	TRUE
product_05+coffee+euro+pour_regular+tea	Formula evaluates	TRUE
product_06+cappuccino+coffee+euro+pour_espresso	Formula evaluates	TRUE
product_07+cappuccino+coffee+euro+pour_regular	Formula evaluates	TRUE
product_08+cappuccino+coffee+euro+pour_espresso+tea	Formula evaluates	TRUE
product_09+cappuccino+coffee+euro+pour_regular+tea	Formula evaluates	TRUE
product_10+coffee+dollar+pour_espresso	Formula evaluates	FALSE
product_11+coffee+dollar+pour_regular	Formula evaluates	FALSE
product_12+cappuccino+coffee+dollar+pour_espresso	Formula evaluates	TRUE
product_13+cappuccino+coffee+dollar+pour_regular	Formula evaluates	TRUE

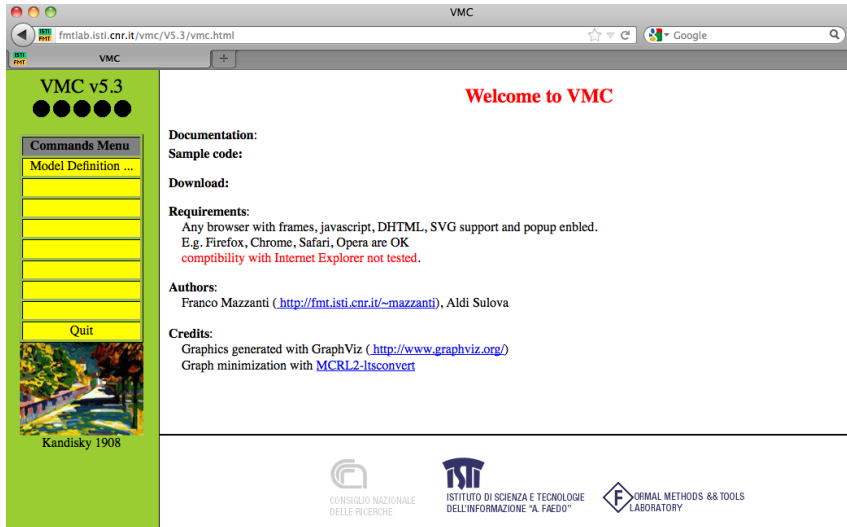
Logic Formula for all Products

[dollar] EF {cappuccino} true



*Upon the insertion of a dollar,
it might be the case that eventually a cappuccino can be chosen*

Slightly outdated demo: VMC's (old) web interface



The screenshot shows a web browser window titled "VMC" with the address bar containing "fmtlab.isti.cnr.it/vmc/V5.3/vmc.html". The page has a green sidebar on the left with the title "VMC v5.3" and five black circles below it. The sidebar contains a "Commands Menu" with a "Model Definition ..." button and several empty yellow buttons, followed by a "Quit" button and a small image of a landscape painting labeled "Kandisky 1908". The main content area is white and features the heading "Welcome to VMC" in red. Below this, there are sections for "Documentation:", "Sample code:", "Download:", "Requirements:", "Authors:", and "Credits:". The "Requirements:" section lists browser compatibility. The "Authors:" section lists Franco Mazzanti and Aldi Sulova. The "Credits:" section mentions GraphViz and MCRL2-Itscnvert.

VMC v5.3

Commands Menu

Model Definition ...

Quit

Kandisky 1908

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:

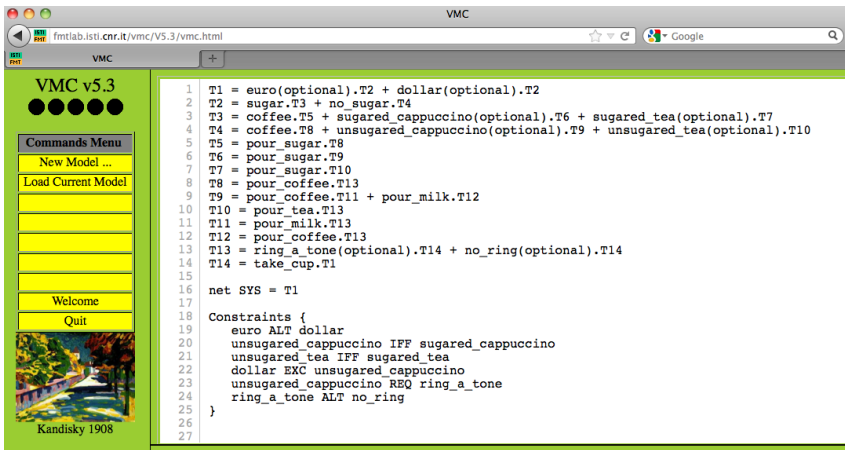
Graphics generated with GraphViz (<http://www.graphviz.org/>)
Graph minimization with [MCRL2-Itscnvert](#)

CONSIGLIO NAZIONALE DELLE RICERCHE

ISTI
ISTITUTO DI SCIENZA E TECNOLOGIE DELL'INFORMAZIONE "A. FAEDO"

FORMAL METHODS && TOOLS LABORATORY

Family of coffee machines specified in VMC



The screenshot shows the VMC v5.3 web interface. On the left is a sidebar with a 'Commands Menu' containing 'New Model ...', 'Load Current Model', and 'Welcome'. Below the menu is a 'Quit' button and a small image of a bridge with the caption 'Kandisky 1908'. The main area displays a list of transition rules (T1 to T14) and constraints. The rules are:

```
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
```

The constraints are:

```
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 logic formalization from user

Family/MTS of coffee machines visualized by VMC

VMC v5.3

Commands Menu

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

Kandisky 1919

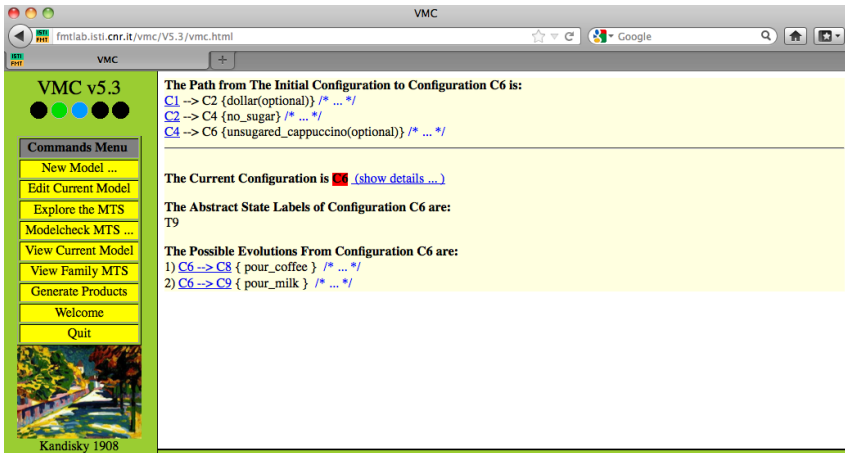
Family Model Evolutions Chart (MTS)

Zoom Out Zoom In

View the graph in [DOT](#) format or as a [PDF](#) pdf picture or as plain [SYG](#) data.

The above graph shows the MTS family model evolutions. Dotted edges denote "may" transitions, full edges denote "must" transitions.

Family/MTS of coffee machines explored in VMC



The screenshot shows a web browser window titled "VMC" with the URL "fmtlab.isti.cnr.it/vmc/V5.3/vmc.html". The interface is divided into a left sidebar and a main content area. The sidebar, titled "VMC v5.3", contains a "Commands Menu" with buttons for "New Model ...", "Edit Current Model", "Explore the MTS", "Modelcheck MTS ...", "View Current Model", "View Family MTS", "Generate Products", "Welcome", and "Quit". Below the menu is a small image of a bridge over a stream, labeled "Kandisky 1908". The main content area displays the following information:

The Path from The Initial Configuration to Configuration C6 is:
C1 -> C2 {dollar(optional)} /* ... */
C2 -> C4 {no_sugar} /* ... */
C4 -> C6 {unsugared_cappuccino(optional)} /* ... */

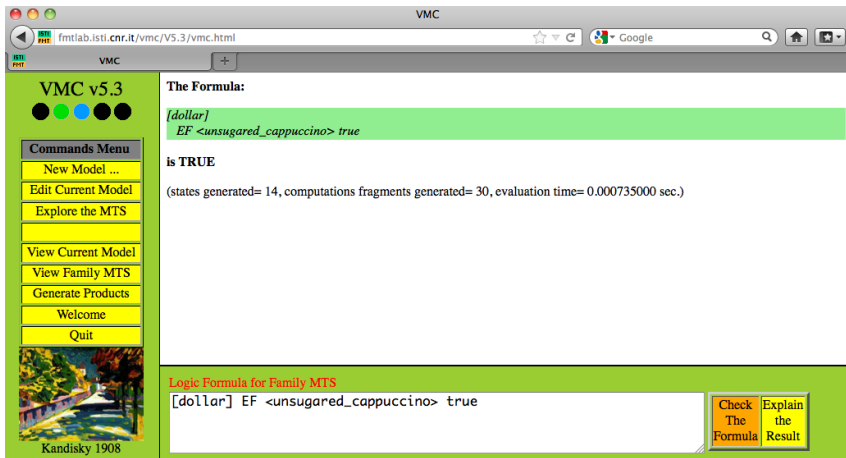
The Current Configuration is **C6** ([show details ...](#))

The Abstract State Labels of Configuration C6 are:
T9

The Possible Evolutions From Configuration C6 are:
1) C6 -> C8 { pour_coffee } /* ... */
2) C6 -> C9 { pour_milk } /* ... */

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 dollar EXC unsugared_cappuccino₃₂ / 44

Outcome of a property verified over a family with VMC



The screenshot shows a web browser window titled "VMC" with the URL `fmtlab.isti.cnr.it/vmc/V5.3/vmc.html`. The interface is divided into several sections:

- VMC v5.3**: A header with four colored circles (black, green, blue, black).
- Commands Menu**: A vertical list of buttons including "New Model ...", "Edit Current Model", "Explore the MTS", "View Current Model", "View Family MTS", "Generate Products", "Welcome", and "Quit".
- The Formula:** A section with a green background containing the text:

```
[dollar]  
EF <unsugared_cappuccino> true
```
- is TRUE**: A section with a white background containing the text:

```
(states generated= 14, computations fragments generated= 30, evaluation time= 0.000735000 sec.)
```
- Logic Formula for Family MTS**: A section with a green background containing a text input field with the formula `[dollar] EF <unsugared_cappuccino> true` and two buttons: "Check The Formula" and "Explain the Result".
- Kandisky 1908**: A small image of a painting with the caption "Kandisky 1908".

The formula expresses that every path through the MTS starting with a dollar insertion, eventually leads to an unsugared cappuccino

Outcome of a property explained by VMC

The screenshot shows the VMC v5.3 web interface. On the left is a sidebar with a 'Commands Menu' containing options like 'New Model ...', 'Edit Current Model', 'Explore the MTS', 'View Current Model', 'View Family MTS', 'Generate Products', 'Welcome', and 'Quit'. Below the menu is a small image of a landscape painting titled 'Kandisky 1908'. The main content area displays the results of a model check for a property. It shows three instances of the property being checked, each with its status and the reason for the outcome.

VMC v5.3

Commands Menu

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

Kandisky 1908

The formula:
 $[dollar] EF \langle unsugared_cappuccino \rangle true$
is **FOUND_TRUE** in State C1

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

And the evolutions which satisfy the action formula dollar also satisfy the subformula:
 $EF \langle unsugared_cappuccino \rangle true$

In particular:
In state C2 the subformula:
 $EF \langle unsugared_cappuccino \rangle true$ is Satisfied .

The formula:
 $EF \langle unsugared_cappuccino \rangle true$
is **FOUND_TRUE** in State C2

This happens because
C2 \rightarrow C4 {no_sugar} /* ... */
and the subformula:
 $\langle unsugared_cappuccino \rangle true$
is Satisfied in State C4

The formula:
 $\langle unsugared_cappuccino \rangle true$
is **FOUND_TRUE** in State C4

This happens because
C4 \rightarrow C11 {unsugared_cappuccino(optional)}
the transition label satisfies the action expression $unsugared_cappuccino$
and in State C11 the subformula:
 $true$ is Satisfied .

Logic Formula for Family MTS

$[dollar] EF \langle unsugared_cappuccino \rangle true$

Check The Formula Explain the Result

Products of family of coffee machines derived by VMC




The screenshot shows a web browser window titled "VMC" with the URL "fmtlab.isti.cnr.it/vmc/V5.3/vmc.html". The browser's address bar includes a search engine icon and the text "Google". The page content is divided into two main sections:

- Left Sidebar:**
 - Header: "VMC v5.3" with four colored circles (green, blue, black, black).
 - Section: "Commands Menu" with a list of buttons: "New Model ...", "Edit Current Model", "Explore the MTS", "Modelcheck Products", "View Current Model", "View Family MTS", "Generate Products", "Welcome", and "Quit".
 - Image: A painting of a path through a forest, labeled "Kandisky 1908".
- Main Content Area:**
 - Section: "Valid Products of the Family" with a list of ten blue hyperlinks representing different product configurations:
 - [product101-dollar-sugared_tea-unsugared_tea-ring_a_tone](#)
 - [product102-dollar-sugared_tea-unsugared_tea-no_ring](#)
 - [product11-euro-ring_a_tone](#)
 - [product12-euro-no_ring](#)
 - [product20-dollar-ring_a_tone](#)
 - [product21-dollar-no_ring](#)
 - [product71-euro-sugared_cappuccino-unsugared_cappuccino-ring_a_tone](#)
 - [product83-euro-sugared_tea-unsugared_tea-ring_a_tone](#)
 - [product84-euro-sugared_tea-unsugared_tea-no_ring](#)
 - [product95-euro-sugared_cappuccino-sugared_tea-unsugared_cappuccino-unsugared_tea-ring_a_tone](#)

VMC indeed generates all 10 valid products/LTSs that are derivable from the family/MTS if the variability constraints are considered

Outcomes of a property verified over products with VMC



The screenshot shows the VMC v5.3 web interface. The browser address bar displays `fmlab.lsti.cnr.it/vmc/V5.3/vmc.html`. The page title is "VMC". The main content area is titled "Evaluation of the formula "[dollar] EF true" on all family products". Below this title, a list of products is shown, each with a corresponding evaluation result for the formula. The products and their results are:

Product	Formula evaluates
product101-dollar-sugared_tea-unsugared_tea-ring_a_tone	FALSE
product102-dollar-sugared_tea-unsugared_tea-no_ring	FALSE
product11-euro-ring_a_tone	TRUE
product12-euro-no_ring	TRUE
product20-dollar-ring_a_tone	FALSE
product21-dollar-no_ring	FALSE
product71-euro-sugared_cappuccino-unsugared_cappuccino-ring_a_tone	TRUE
product83-euro-sugared_tea-unsugared_tea-ring_a_tone	TRUE
product84-euro-sugared_tea-unsugared_tea-no_ring	TRUE
product95-euro-sugared_cappuccino-sugared_tea-unsugared_cappuccino-unsugared_tea-ring_a_tone	TRUE

At the bottom of the interface, there is a section for the logic formula used for all products: `[dollar] EF <unsugared_cappuccino> true`. To the right of this formula are two buttons: "Check The Formula" and "Explain the Result".

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



The screenshot shows a web browser window titled "VMCP" with the URL `fmlab.isitl.cnr.it/vmcp/V5.2/vmcp.html?remotemodel`. The browser has two tabs: "VMC" and "VMCP". The "VMC" tab is active and displays the "VMC v5.3 (on product)" interface. On the left side of the "VMC" tab, there is a "Commands Menu" with buttons for "New Model ...", "Load Current Model", and "Quit". Below the menu is a small image of a bridge over a stream, labeled "Kandisky 1908". The main content area of the "VMC" tab shows a textual encoding of a product specification, which is displayed in a monospaced font. The text is as follows:

```
1 -----
2 -- product71-euro-sugared_cappuccino-unsugared_cappuccino-ring_a_tone
3 -----
4 T1 = euro.T2
5 T2 = sugar.T3 + no_sugar.T4
6 T3 = coffee.T5 + sugared_cappuccino.T6
7 T4 = coffee.T8 + unsugared_cappuccino.T9
8 T5 = pour_sugar.T8
9 T6 = pour_sugar.T9
10 T8 = pour_coffee.T11
11 T9 = pour_coffee.T12 + pour_milk.T13
12 T11 = ring_a_tone.T14
13 T12 = pour_milk.T11
14 T13 = pour_coffee.T11
15 T14 = take_cup.T1
16
17 net SYS = T1
18
19
20
21
22
23
24
25
26
27
```

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

Product/LTS on previous slide visualized by VMC

VMCP

fmlab.isti.cnr.it/vmcp/V5.2/vmcp.html?remotemodel

VMC VMCP

VMC v5.3
(on product)

Commands Menu

- New Model ...
- Edit Current Model
- Explore the LTS
- View Current Model
- View the LTS Graph
- Quit

Kandisky 1919

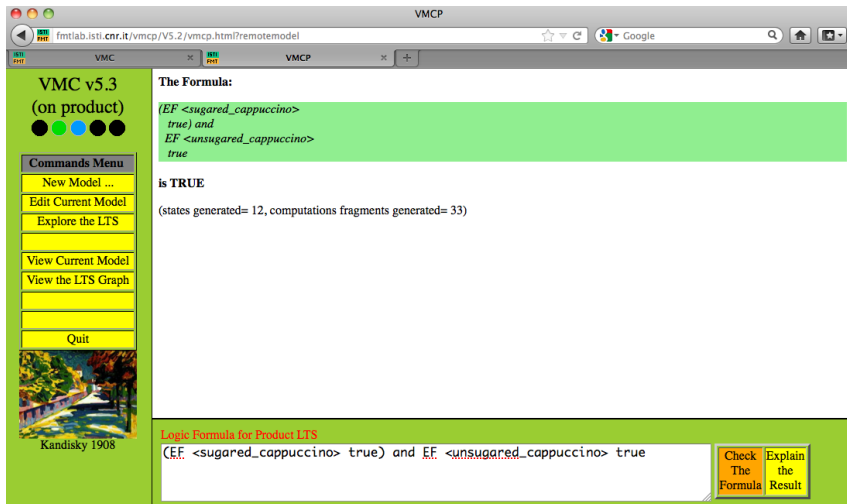
Product Model Evolutions Chart (LTS)

Zoom Out Zoom In

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



The screenshot shows the VMC v5.3 web interface. The browser address bar displays `fmlab.isti.cnr.it/vmcp/V5.2/vmcp.htm?remotemodel`. The page title is "VMC v5.3 (on product)". On the left, there is a "Commands Menu" with buttons for "New Model ...", "Edit Current Model", "Explore the LTS", "View Current Model", "View the LTS Graph", and "Quit". Below the menu is a small image of a landscape painting by Kandinsky from 1908. The main content area displays the following text:

The Formula:

```
(EF <sugared_cappuccino> true) and  
EF <unsugared_cappuccino> true
```

is **TRUE**

(states generated= 12, computations fragments generated= 33)

At the bottom, there is a section titled "Logic Formula for Product LTS" containing the formula:

```
(EF <sugared_cappuccino> true) and EF <unsugared_cappuccino> true
```

Two buttons are visible in the bottom right corner: "Check The Formula" and "Explain the Result".

The formula expresses that in this particular LTS, there exists both a path to a sugared cappuccino and to an unsugared cappuccino


Outcome property on previous slide explained by VMC

The screenshot shows the VMC v5.3 web interface. The browser address bar displays `fmlab.lsti.cnr.it/vmcp/V5.2/vmcp.html?remotemodel`. The interface is divided into a left sidebar and a main content area.

VMC v5.3 (on product)

Commands Menu

- New Model ...
- Edit Current Model
- Explore the LTS
- View Current Model
- View the LTS Graph
- Quit


Kandisky 1908

The formula:
 $(EF \langle \text{sugared_cappuccino} \rangle \text{ true})$ and $EF \langle \text{unsugared_cappuccino} \rangle \text{ true}$
is **FOUND_TRUE** in State C1

This happens because the subformula:
 $EF \langle \text{sugared_cappuccino} \rangle \text{ true}$
is **Satisfied** in State C1

And because the subformula:
 $EF \langle \text{unsugared_cappuccino} \rangle \text{ true}$
is **Satisfied** in State C1

The formula:
 $EF \langle \text{sugared_cappuccino} \rangle \text{ true}$
is **FOUND_TRUE** in State C1

This happens because
 $C1 \rightarrow C2 \{ \text{euro} \} !^* \dots !^*$
 $C2 \rightarrow C3 \{ \text{sugar} \} !^* \dots !^*$
and the subformula:
 $\langle \text{sugared_cappuccino} \rangle \text{ true}$
is **Satisfied** in State C3

The formula:
 $EF \langle \text{unsugared_cappuccino} \rangle \text{ true}$
is **FOUND_TRUE** in State C1

This happens because
 $C1 \rightarrow C2 \{ \text{euro} \} !^* \dots !^*$
 $C2 \rightarrow C4 \{ \text{no_sugar} \} !^* \dots !^*$
and the subformula:
 $\langle \text{unsugared_cappuccino} \rangle \text{ true}$
is **Satisfied** in State C4

Logic Formula for Product LTS
 $(EF \langle \text{sugared_cappuccino} \rangle \text{ true})$ and $EF \langle \text{unsugared_cappuccino} \rangle \text{ true}$

Check The Formula | **Explain the Result**

Non-trivial for branching logics!

Thanks!

Next: SPL model checking of FTSs with mCRL2 (18/3/16)

A value-passing modal process algebra

Let \mathcal{A} be a set of actions, let $a \in \mathcal{A}$ and let $L \subseteq \mathcal{A}$

Processes are built from terms and actions according to the syntax

$$N ::= [P]$$

$$P ::= K(e) \mid P/L/P$$

$[P]$ denotes a closed system, i.e. it cannot evolve on input actions $a(?v)$
 $K(e)$ is a process identifier from set of process definitions of form $K(v) \stackrel{\text{def}}{=} T$

$$T ::= nil \mid K(e) \mid A.T \mid T + T \mid [e \bowtie e] T$$

$$A ::= a(e) \mid a(\text{may}, e) \mid a(?v) \mid a(\text{may}, ?v)$$

$$e ::= v \mid \mathbf{int} \mid e \pm e$$

$\bowtie \in \{<, \leq, =, \neq, \geq, >\}$, v is a variable, \mathbf{int} is an integer, $\pm \in \{+, -, \times, \div\}$

- nil terminated process that has finished execution
- K process identifier that is used for modelling recursive sequential processes
- $A.P$ process that can execute action A and then behave as P
- $P + Q$ process that can non-deterministically choose to behave as P or as Q
- $P / L / Q$ process formed by the parallel composition of P and Q (it can synchronize on actions in L and interleave others)

We distinguish **must** actions $a \in \delta^\square$ and **may but not must** actions $a(may) \in \delta^\diamondsetminus \delta^\square$ (each action type is treated differently in the SOS semantics)

Semantics in SOS style over MTS

$$\text{(sys)} \frac{P \xrightarrow{a(e)} P'}{[P] \xrightarrow{a(e)} [P']}$$

$$\text{(act}_{\square}\text{)} \frac{}{\alpha.P \xrightarrow{\alpha} P} \quad \alpha \in \{a(e), a(?v)\}$$

$$\text{(or}_{\square}\text{)} \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \quad \alpha \in \{a(e), a(?v)\}$$

$$\text{(int}_{\square}\text{)} \frac{P \xrightarrow{\ell} P'}{P / L / Q \xrightarrow{\ell} P' / L / Q} \quad \ell \notin L$$

$$\text{(par}_{\square}\text{)} \frac{P \xrightarrow{a(e_1)} P' \quad Q \xrightarrow{a(e_2)} Q'}{P / L / Q \xrightarrow{a} P' / L / Q'} \quad a \in L, e_1 = e_2$$

$$\text{(par}_{\square}\text{)} \frac{P \xrightarrow{a(?v)} P' \quad Q \xrightarrow{a(e)} Q'}{P / L / Q \xrightarrow{a} P'[e/v] / L / Q'} \quad a \in L$$

$$\text{(guard)} \frac{}{[e_1 \bowtie e_2] P(e_3) \rightarrow P(e_3)} \quad e_1 \bowtie e_2$$

(similarly in case of may actions and for the remaining operators)