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
UNIPI, 14/3/16
Background: Software Product Line Engineering (4/3/16)

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

v-ACTL: variability-aware, action-based CTL
- Preservation of formulae in v-ACTL / v-ACTLive

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)
  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** ($L^2$TS) is a sextuple $(Q, A, \overline{q}, \rightarrow, AP, L)$ with states $Q$, actions $A$, initial state $\overline{q}$, 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:

A **Featured Transition System** (FTS) is an octuple $(Q, A, \overline{q}, \rightarrow, AP, L, FD, \gamma)$ with underlying $L^2$TS $(Q, A, \overline{q}, \rightarrow, AP, L)$, 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:

\[
\text{VendingMachine} \\
\text{Beverages} \quad \text{FreeDrinks} \quad \text{CancelPurchase} \\
\text{Soda} \quad \text{Tea} \\
\text{serveSoda} \quad \text{serveTea} \quad \text{open} \quad \text{take} \\
\text{pay} \quad \text{change} \quad \text{tea} \quad \text{soda} \\
\text{close} \\
\]
FTS of example SPL: a vending machine

Feature model:

- VendingMachine
  - Beverages
    - Soda
    - Tea
  - FreeDrinks
  - CancelPurchase

Example features:
- \{v, b, s, t\}
- \{v, b, s, c\}
MTS for SPLE

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

Next we define MTS with variability constraints:

A Modal Transition System (MTS) is a quintuple \((Q, A, \overline{q}, \rightarrow^\Box, \rightarrow^\Diamond)\) such that \((Q, A, \overline{q}, \rightarrow^\Box \cup \rightarrow^\Diamond)\) 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 \(\rightarrow^\Box \subseteq Q \times A \times Q\): required transitions

By definition, any required transition is also possible: \(\rightarrow^\Box \subseteq \rightarrow^\Diamond\) (denote \(--\rightarrow \equiv \rightarrow^\Diamond \setminus \rightarrow^\Box\): optional transitions)
...variability constraints (accepted by VMC)

Variability constraints of form ALTernative, EXCludes, REQuires, etc.

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

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

\( a_1 \text{ EXC} a_2 : \) at most one of the actions \( a_1 \) and \( a_2 \) is reachable in \( L \)

\( a_1 \text{ REQ} a_2 : \) action \( a_2 \) is reachable in \( L \) whenever \( a_1 \) is reachable in \( L \)

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

\( a_1 \text{ REQ} (a_2 \text{ OR} \cdots \text{ OR} a_n) : \) at least one among the \( n \geq 2 \) actions \( a_2, \ldots, a_n \) is reachable in \( L \) if \( a_1 \) is reachable in \( L \)
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 \((Q, A, q, \delta^\lozenge, \delta^\Box)\) be a coherent MTS, i.e. \(\exists \xrightarrow{a} \implies \nexists \xrightarrow{a}\)

The set \(\{P_i = (Q_i, A, q, \delta_i) \mid i > 0\}\) of product LTS is obtained by considering each pair of \(Q_i \subseteq Q\) and \(\delta_i \subseteq \delta^\lozenge \cup \delta^\Box\) to be defined s.t.

1. every \(q \in Q_i\) is reachable in \(P_i\) from \(q\) via transitions from \(\delta_i\)
2. there exists no \((q, a, q') \in \delta^\Box \setminus \delta_i\) such that \(q \in Q_i\)
3. LTS is consistent: both \(\xrightarrow{a} \sim \xrightarrow{a}\) and \(\not\xrightarrow{a}\) 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 \((Q, A, \bar{q}, \delta^{\lozenge}, \delta^{\Box})\) be a coherent MTS, i.e. \(\exists \quad \not\rightarrow \quad \not\rightarrow \quad \# \quad \rightarrow\)

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 \(\delta_i \subseteq \delta^{\lozenge} \cup \delta^{\Box}\) to be defined s.t.

1. every \(q \in Q_i\) is reachable in \(\mathcal{P}_i\) from \(\bar{q}\) via transitions from \(\delta_i\)
2. there exists no \((q, a, q') \in \delta^{\Box} \setminus \delta_i\) such that \(q \in Q_i\)
3. LTS is consistent: both \(\not\rightarrow \quad \not\rightarrow \quad \rightarrow \quad \not\rightarrow\) and \(\not\rightarrow\) 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/
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/
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 ∀ feature (to handle more complex feature expressions)
2. dummy transition ∀ 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): $a \text{ REQ } c$

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 ∀ feature (to handle more complex feature expressions)
2. dummy transition ∀ 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

FTS

\[ 1 \xrightarrow{a / A} 2 \xrightarrow{b / B} 3 \xrightarrow{c / C} 4 \]

LTS

\[ 1 \xrightarrow{a} 2 \]

valid product (namely \( A \land C \land \neg B \))

From FTS to MTS (naive): a REQ c

MTS

\[ 1 \xrightarrow{a} 2 \xrightarrow{b} 3 \xrightarrow{c} 4 \]

LTS

\[ 1 \xrightarrow{a} 2 \]

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)

MTS

\[ \{a, b, c, A, B, C\} \xrightarrow{s} \]

\[ \{a, c, A, C\} \xrightarrow{s} \]

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

From FTS to MTS (naive): a REQ c

From FTS to MTS (solution):
1. new action ∀ feature (to handle more complex feature expressions)
2. dummy transition ∀ 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 ∀ feature (to handle more complex feature expressions)
2. dummy transition ∀ 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

FTS

\[1 \xrightarrow{a/A} 2 \xrightarrow{b/B} 3 \xrightarrow{c/C} 4\]

LTS

\[1 \xrightarrow{a} 2\]

valid product (namely \(A \land C \land \neg B\))

From FTS to MTS (naive): a REQ c

MTS

\[1 \xrightarrow{a} 2 \xrightarrow{b} 3 \xrightarrow{c} 4\]

LTS

\[1 \xrightarrow{a} 2\]

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)

MTS

\[1 \xrightarrow{a} 2 \xrightarrow{b} 3 \xrightarrow{c} 4\]

\{a, b, c, A, B, C\} \xrightarrow{s}

LTS

\[1 \xrightarrow{a} 2\]

\{a, c, A, C\} \xrightarrow{s}

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

FTS \[1 \xrightarrow{a/A} 2 \xrightarrow{b/B} 3 \xrightarrow{c/C} 4\]  
LTS \[1 \xrightarrow{a} 2\]  
valid product (namely \(A \land C \land \neg B\))

From FTS to MTS (naive): \(a \text{ REQ } c\)

MTS \[1 \xrightarrow{a} 2 \xrightarrow{b} 3 \xrightarrow{c} 4\]  
LTS \[1 \xrightarrow{a} 2\]  
valid product (violates \(a \text{ REQ } c\))

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)

MTS \[1 \xrightarrow{a} 2 \xrightarrow{b} 3 \xrightarrow{c} 4\]  
\(\{a,b,c,A,B,C\}\) \(\xrightarrow{s}\)  
LTS \[1 \xrightarrow{a} 2\]  
\(\{a,c,A,C\}\) \(\xrightarrow{s}\)  
valid product (satisfies \(a \text{ REQ } c\))

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

FTS

\[ 1 \xrightarrow{a/A} 2 \xrightarrow{b/B} 3 \xrightarrow{c/C} 4 \]

LTS

\[ 1 \xrightarrow{a} 2 \]

valid product (namely \( A \land C \land \neg B \))

From FTS to MTS (naive): a REQ c

MTS

\[ 1 \xrightarrow{a} 2 \xrightarrow{b} 3 \xrightarrow{c} 4 \]

LTS

\[ 1 \xrightarrow{a} 2 \]

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)

MTS

\[ 1 \xrightarrow{a} 2 \xrightarrow{b} 3 \xrightarrow{c} 4 \]

\[ \{a, b, c, A, B, C\} \xrightarrow{s} \]

LTS

\[ 1 \xrightarrow{a} 2 \xrightarrow{\{a, c, A, C\}} s \]

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 
}
Step 2: definition of valid products in terms of actions

Define logic formula ‘a ↔ φ’ for each transition \( \frac{a/φ}{\rightarrow} \) in FTS (feature expressions not translatable in VMC format are transformed in CNF, which is the reason for which the \( n \)-ary OR construct \( b_1 \) OR \( \cdots \) OR \( b_n \) can now contain either \( b_i = a_i \) or its negation \( b_i = \neg a_i \) )

Constraints {
    free IFF FreeDrinks
    pay ALT FreeDrinks
    cancel IFF CancelPurchase
    soda IFF Soda
    tea IFF Tea
    takeFree IFF FreeDrinks
    open ALT FreeDrinks
}
Step 2: definition of valid products in terms of actions

Define logic formula ‘a ↔ \( \phi \)’ for each transition \( \frac{a/\phi}{\text{in FTS}} \) (feature expressions not translatable in VMC format are transformed in CNF, which is the reason for which the \( n \)-ary OR construct \( b_1 \ OR \cdots \ OR \ b_n \) 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 
}
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
Step 3: definition of valid products in MTS and variability constraints

1. Define FTS process algebraically, using ‘a(may)’ for each ‘a/$\phi$’

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

\[
\text{Behaviour} = \text{behaviour}.T1 \\
T1 = \text{pay}(\text{may}).T2 + \text{free}(\text{may}).T3 \\
T2 = \text{change}.T3 \\
... \\
T9 = \text{close}.T1 \\
\]

\[
\text{Signature} = \text{signature}.( \\
\quad \text{free}(\text{may}).\text{nil} + \text{pay}(\text{may}).\text{nil} + ... + \text{open}(\text{may}).\text{nil} + \\
\quad \text{FreeDrinks}(\text{may}).\text{nil} + ... + \text{Tea}(\text{may}).\text{nil} \\
\)
\]

\[
\text{VMCmodel} = \text{Behaviour} + \text{Signature}
\]
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
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

\[
\begin{align*}
\text{Behaviour} & = \text{behaviour}.T_1 \\
T_1 & = \text{pay(may)}.T_2 + \text{free(may)}.T_3 \\
T_2 & = \text{change}.T_3 \\
\vdots \\
T_9 & = \text{close}.T_1 \\
\text{Signature} & = \text{signature}.( \\
& \quad \text{free(may)}.nil + \text{pay(may)}.nil + \ldots + \text{open(may)}.nil + \\
& \quad \text{FreeDrinks(may)}.nil + \ldots + \text{Tea(may)}.nil \\
) \\
\text{VMCmodel} & = \text{Behaviour} + \text{Signature}
\end{align*}
\]
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

\[
\text{Behaviour} = \text{behaviour}.T_1 \\
T_1 = \text{pay}(\text{may}).T_2 + \text{free}(\text{may}).T_3 \\
T_2 = \text{change}.T_3 \\
\vdots \\
T_9 = \text{close}.T_1 \\
\text{Signature} = \text{signature}.( \\
\quad \text{free}(\text{may}).\text{nil} + \text{pay}(\text{may}).\text{nil} + \ldots + \text{open}(\text{may}).\text{nil} + \\
\quad \text{FreeDrinks}(\text{may}).\text{nil} + \ldots + \text{Tea}(\text{may}).\text{nil} \\
\) \\
\text{VMCmodel} = \text{Behaviour} + \text{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
}

Diagram with nodes and transitions:

- return/v
- free/f
- canel/c
- soda/s
- serveSoda/s
- tea/t
- serveTea/t
- open/v
- close/v
- take/v
- take/f
- change/v

Arrow directions and labels indicating transitions.
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 \([S]\) 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 \([S]\) and \( \text{MTS}(S') \)
   such that each \( p \) in \([S]\) 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.
Recall v-ACTL: variability-aware, action-based CTL

Action formulae $\psi$, state formulae $\phi$, path formulae $\pi$

\[
\psi ::= true \mid a \mid a(e) \mid \neg \psi \mid \psi \land \psi
\]

\[
\phi ::= true \mid \neg \phi \mid \phi \land \phi \mid \langle \chi \rangle \phi \mid [\chi] \phi \mid E \pi \mid A \pi \mid \mu Y.\phi(Y) \mid \nu Y.\phi(Y)
\]

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

$\langle \chi \rangle \Box \phi$: a next state exists, reachable by a must transition executing an action satisfying $\chi$, in which $\phi$ holds

$[\chi] \Box \phi$: in all next states reachable by a must transition executing an action satisfying $\chi$, $\phi$ holds

$F \Box \{\chi\} \phi$: there exists a future state, reached by an action satisfying $\chi$, in which $\phi$ holds and all transitions until that state are must transitions
Preservation of formulae in $v$-ACTL$^\Box/v$-ACTLive$^\Box$

$v$-ACTL$^\Box/v$-ACTLive$^\Box$:

$$
\phi ::= \text{false} \mid \text{true} \mid \phi \land \phi \mid \phi \lor \phi \mid [\psi] \phi \mid \langle \psi \rangle \Box \phi \mid EF \Box \phi \mid EF \{\psi\} \phi \mid AF \Box \phi \mid AF \{\psi\} \phi \mid AF \phi \mid AF \{\psi\} \phi
$$

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

$v$-ACTL$^-$/

$$
\chi ::= \text{false} \mid \text{true} \mid \chi \land \chi \mid \chi \lor \chi \mid \langle \psi \rangle \chi \mid EF \chi \mid EF\{\psi\} \chi
$$

any formula that is false for MTS, is also false for all products (LTSs)
Preservation of formulae in $v$-ACTL$\Box$/$v$-ACTL$\Box$:

$v$-ACTL$\Box$/$v$-ACTL$\Box$:

$$\phi ::= \text{false} \mid \text{true} \mid \phi \land \phi \mid \phi \lor \phi \mid [\psi] \phi \mid \langle \psi \rangle \Box \phi \mid EF \Box \phi \mid EF\{\psi\} \phi \mid AF \Box \phi \mid AF\{\psi\} \phi \mid AG \phi \mid AF \phi \mid AF\{\psi\} \phi$$

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

$v$-ACTL$\neg$:

$$\chi ::= \text{false} \mid \text{true} \mid \chi \land \chi \mid \chi \lor \chi \mid \langle \psi \rangle \chi \mid EF \chi \mid EF\{\psi\} \chi$$

any formula that is false for MTS, is also false for all products (LTSs)
Live states use SPL-specific information

\[ S \models \phi \implies S_p \models \phi \quad \forall \text{ product LTS } S_p \text{ of MTS } S \]

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

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

MTS

Assume

LTS

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

\[ \implies p \text{ is a live state, since } a \text{ OR } b \text{ gives rise to a live action set } \{a, b\} \]
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 \((\langle \rangle \Box, [\ ] \Box, F \Box)\)

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

\[ \text{MTS } \quad \text{Assume } \quad \text{LTS} \]

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

\[ \Rightarrow p \text{ is a live state, since } a \text{ OR } b \text{ gives rise to a live action set } \{a, b\} \]
Live states use SPL-specific information

\[ S \models \phi \Rightarrow S_p \models \phi \ \forall \text{product LTS } S_p \text{ of MTS } S \]

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

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

\[
\begin{align*}
\text{MTS} & \quad \!
\begin{array}{c}
\!
\end{array}
\end{align*}
\]

\[
\begin{align*}
\text{Assume} & \quad \!
\begin{array}{c}
\!
\end{array}
\end{align*}
\]

\[
\begin{align*}
\text{a OR b} & \quad \!
\begin{array}{c}
\!
\end{array}
\end{align*}
\]

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

\[ \Rightarrow p \text{ is a live state, since } \text{a OR b gives rise to a live action set } \{a, b\} \]
Live states use SPL-specific information

\[ S \models \phi \Rightarrow S_p \models \phi \ \forall \text{ product LTS } S_p \text{ of } MTS \ S \]

Recall: all (reachable) must transitions are preserved \((\langle \rangle \Box, [\cdot] \Box, F \Box)\)

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

\[
\begin{align*}
\text{MTS} & \quad \xymatrix{ p \ar@{-}[r]^a & q \ar@{-}[r]^b & r } \quad \text{Assume} \quad a \text{ OR } b \\
\text{LTS} & \quad \xymatrix{ p \ar@{-}[r]^a & q \ar@{-}[r]^b & r }
\end{align*}
\]

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

\[ \Rightarrow p \text{ is a live state, since } a \text{ OR } b \text{ gives rise to a live action set } \{a, b\} \]
Live states use SPL-specific information

\[ S \models \phi \Rightarrow S_p \models \phi \ \forall \text{product LTS } S_p \text{ of MTS } S \]

Recall: all (reachable) must transitions are preserved (\(\llcorner \), \([\ ]\), \(F\))

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

MTS

\[
\begin{array}{c}
\text{a} \\
\text{b}
\end{array}
\]

\[ \xymatrix{ p \ar[r]^a & q \\
\ar[r]^b & r }
\]

Assume

\[ a \text{ OR } b \]

\[ p \cdot a \rightarrow q \]

\[ p \cdot a \rightarrow q \]

\[ p \cdot b \rightarrow r \]

\[ p \cdot b \rightarrow r \]

LTS

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

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

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.

Whenever pay occurs, eventually takePaid occurs.
Specification of the family of coffee machines in VMC

```
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 net SYS = T1
13
14 Constraints {
15    euro ALT dollar
16    cappuccino OR coffee OR tea
17    dollar EXC tea
18    cappuccino REQ coffee
19    coffee REQ (pour_espresso ALT pour_regular)
20 }
21
22
```
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
A formula verified by VMC over the family of coffee machines

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

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

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

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-itsconvert
Family of coffee machines specified in VMC

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

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

The above graph shows the MTS family model evolutions. Dotted edges denote "may" transitions, full edges denote "must" transitions.
MTS model of coffee machine family actually permits a user to buy a cappuccino with a dollar, something which is forbidden for its products by the variability constraint dollar \( \text{EXC} \) unsugared_cappuccino.
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 formula:

\[ \text{dollar} \]

\[ \text{EF <unsugared_cappuccino> true} \]

is \text{FOUND_TRUE} in State C1

This happens because

- C1 -> C2 \{euro(optional)\}
- C1 -> C2 \{dollar(optional)\}

And the evolutions which satisfy the action formula \text{dollar}
also satisfy the subformula:

\[ \text{EF <unsugared_cappuccino> true} \]

In particular:

In state C2 the subformula:

\[ \text{EF <unsugared_cappuccino> true} \text{ is Satisfied} \]

The formula:

\[ \text{EF <unsugared_cappuccino> true} \]

is \text{FOUND_TRUE} in State C2

This happens because

- C2 -> C4 \{no_sugar\} /* ... */

and the subformula:

\[ \text{<unsugared_cappuccino> true} \text{ is Satisfied in State C4} \]

The formula:

\[ \text{<unsugared_cappuccino> true} \]

is \text{FOUND_TRUE} in State C4

This happens because

- C4 -> C11 \{unsugared_cappuccino(optional)\}
- \text{the transition label satisfies the action expression} \text{unsugared_cappuccino}

and in State C11 the subformula:

\[ \text{true} \text{ is Satisfied} \]

Logic Formula for Family MTS

\[ \text{[dollar]} \text{ EF <unsugared_cappuccino> true} \]
Products of family of coffee machines derived by VMC

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

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

Clicking on a product, VMC opens a window with its textual encoding
Product/LTS on previous slide visualized by VMC

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 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

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 $K(v) \overset{\text{def}}{=} T$

\[
T ::= \text{nil} \mid K(e) \mid A.T \mid T + T \mid [e \Join e]T \\
A ::= a(e) \mid a(\text{may}, e) \mid a(?v) \mid a(\text{may}, ?v) \\
e ::= v \mid \text{int} \mid e \pm e
\]

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

\[ \text{nil} \] 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 \textbf{must} actions \( a \in \delta \) and \textbf{may but not must} actions \( a(may) \in \delta \setminus \delta \) (each action type is treated differently in the SOS semantics)
Semantics in SOS style over MTS

\[
\begin{aligned}
\text{(sys)} & \quad \frac{P \xrightarrow{a(e)} P'}{[P] \xrightarrow{a(e)} [P']} \\
\text{(act)} & \quad \frac{\alpha.P \xrightarrow{\alpha} P}{\alpha \in \{a(e), a(?v)\}} \\
\text{(or)} & \quad \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \\
\text{(int)} & \quad \frac{P \xrightarrow{\ell} P'}{P \xrightarrow{\ell} P' / Q \xrightarrow{\ell} P' / L / Q} \\
\text{(par)} & \quad \frac{P \xrightarrow{a(e_1)} P' \quad Q \xrightarrow{a(e_2)} Q'}{P \xrightarrow{a} P' / Q \xrightarrow{a} Q'} \\
\text{(guard)} & \quad \frac{[e_1 \times e_2] P(e_3) \rightarrow P(e_3)}{e_1 \times e_2}
\end{aligned}
\]

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