

# From Featured Transition Systems to Modal Transition Systems with Variability Constraints

Maurice ter Beek, Stefania Gnesi & Franco Mazzanti  
Formal Methods and Tools lab, ISTI-CNR, Pisa, Italy

Ferruccio Damiani & Luca Paolini  
University of Torino, Torino, Italy

SEFM'15  
York, UK  
11-9-2015

# Outline

- 1 **Background: Software Product Line Engineering**
  - Featured Transition Systems
  - Modal Transition Systems with variability constraints
  - VMC: Variability Model Checker
- 2 **Contribution: transformation from FTS to MTS**
  - From feature constraints to action constraints
  - Soundness of model transformation
- 3 **Application: model checking transformed FTS in VMC**
  - v-ACTL: variability-aware, action based CTL
  - Preservation of formulae in  $v\text{-ACTL}^\square / v\text{-ACTLive}^\square$
  - Family-based and product-based verification with VMC
- 4 **Conclusions and future work**

# (Software) Product Line Engineering

SPLE: develop a product line (family) using a shared architecture or platform (**commonalities**) and mass customisation (**variabilities**) to serve, e.g., different markets, thus allowing for (software) reuse

⇒ maximise commonalities whilst minimising cost of variations (i.e. of individual products)

Variability in terms of **features**:

- stakeholder visible pieces of functionality representing both commonalities (e.g. mandatory, required) and variabilities (e.g. optional, alternative)
- only specific feature combinations concern valid products

*"We always have 126,000,000 different bicycles in store!  
But only the parts for 1,000..."*

# (Software) Product Line Engineering

SPLE: develop a product line (family) using a shared architecture or platform (**commonalities**) and mass customisation (**variabilities**) to serve, e.g., different markets, thus allowing for (software) reuse

⇒ maximise commonalities whilst minimising cost of variations (i.e. of individual products)

Variability in terms of **features**:

- stakeholder visible pieces of functionality representing both commonalities (e.g. mandatory, required) and variabilities (e.g. optional, alternative)
- only specific feature combinations concern valid products

*“We always have 126,000,000 different bicycles in store!  
But only the parts for 1,000. . .”*

# SPL example

Configure your BMW vehicle

http://www.bmw.com/com/en/general/carconfigurator/content.html

BMW dealer Brochures Corporate/Direct Sales Shop BMW Financial Services Used Vehicles Search OK

Home 1 2 3 4 5 6 7 X Z4 BMW M BMW i BMW Owners BMW Insights

Configure vehicle

The international BMW website

BMW  
Sheer Driving Pleasure

### Configure your BMW vehicle

Are you interested in configuring your ideal BMW? Please select a country to visit the configurator in the Virtual Center or contact your local BMW dealer who will be happy to answer all your questions about the BMW model you are interested in.

### Related topics

Request information  
Order product catalogues, brochures and equipment lists direct from BMW.

## FIND YOUR BMW.

### Filter

> Reset filter

Budget

Vehicle type

All

Petrol

Diesel

Hybrid

Electric Vehicle

Body type

Saloon

Touring

Convertible

Coupé

Gran Turismo

Sports Hatch

Roadster

Sports Activity Coupé

Sports Activity Vehicle

### 30 Vehicles (465 Model variants)

1



**BMW 1 Series 3-door Sports Hatch (34)**  
from £ 17,775.00



**BMW 1 Series 5-door Sports Hatch (39)**  
from £ 18,305.00

2



**BMW 2 Series Coupé (14)**  
from £ 24,265.00

3



**BMW 3 Series Saloon (56)**  
from £ 23,550.00



**BMW 3 Series Touring (54)**  
from £ 24,865.00



**BMW 3 Series Gran Turismo (39)**

# 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 © ESEC/FSE'07, SPLC'08, Asirelli et al. © IFM'10, SPLC'11, FMOODS'11

### 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

# 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 © ESEC/FSE'07, SPLC'08, Asirelli et al. © IFM'10, SPLC'11, FMOODS'11

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

# 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 © ESEC/FSE'07, SPLC'08, Asirelli et al. © iFM'10, SPLC'11, FMOODS'11

### 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

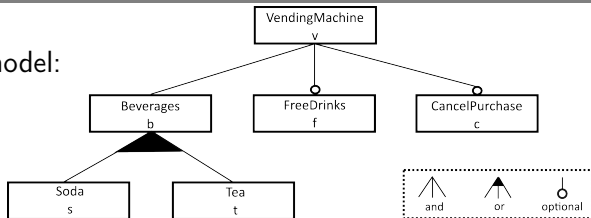
### SNIP, ProVeLines, NuSMV extension

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



# FTS of example SPL

Feature model:

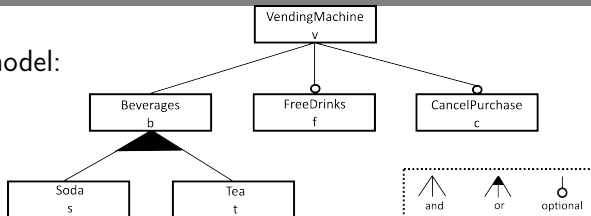


12 valid products

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

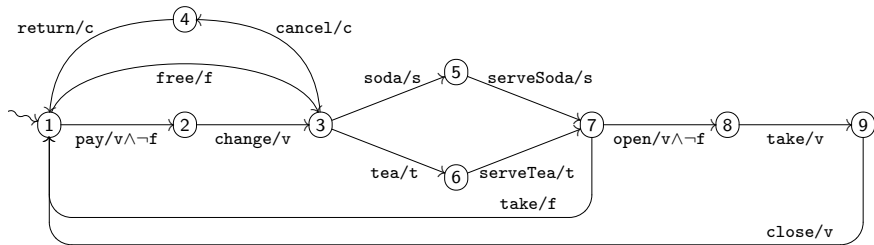
# FTS of example SPL

Feature model:



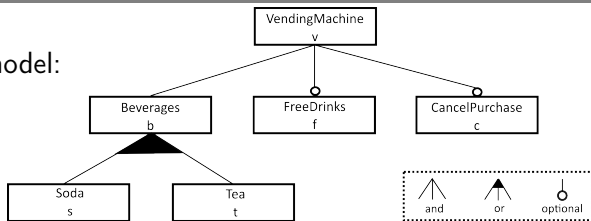
FTS of 12 valid products (LTSs)

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

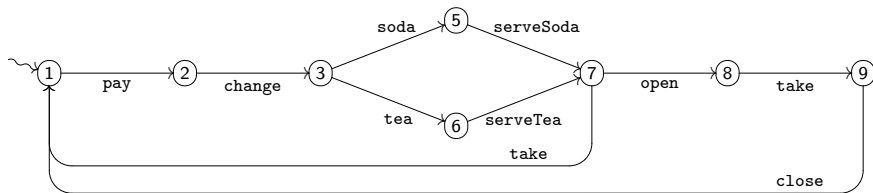


# FTS of example SPL

Feature model:

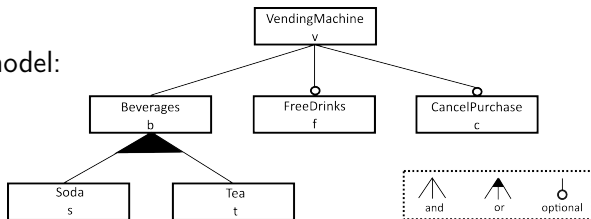


e.g.  $\{v, b, s, t\}$ ,  $\{v, b, s, c\}$

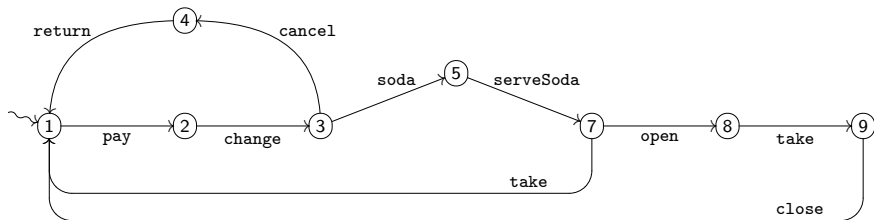


# FTS of example SPL

Feature model:



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



# MTS with variability constraints

## 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

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

Asirelli, ter Beek, Fantechi, Gnesi @ SPLC'11

# MTS for SPLE

A behavioural model, amenable to model checking, able to formalise

1. **shared behaviour**: common among all variants
2. **variation points**: differentiate between variants

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

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)

## 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  $\delta_i$
2. there exists no  $(q, a, q') \in \delta^\square \setminus \delta_i$  such that  $q \in Q_i$
3. LTS are 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  $\delta_i$
2. there exists no  $(q, a, q') \in \delta^\square \setminus \delta_i$  such that  $q \in Q_i$
3. LTS are consistent: both  $\xrightarrow{a} \rightsquigarrow \xrightarrow{a}$  and  $\cancel{\xrightarrow{a}}$  not allowed



## Variability constraints (implemented in VMC)

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

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

$a_1$  **OR**  $\cdots$  **OR**  $a_n$  : at least one of the  $n \geq 2$  actions  $a_1, \dots, a_n$  must occur in  $\mathcal{L}$

$a_1$  **EXC**  $a_2$  : at most one of the actions  $a_1$  and  $a_2$  may occur in  $\mathcal{L}$

$a_1$  **REQ**  $a_2$  : action  $a_2$  must occur in  $\mathcal{L}$  whenever  $a_1$  occurs in  $\mathcal{L}$

$a_1$  **IFF** ( $a_2$  **ALT**  $\cdots$  **ALT**  $a_n$ ) : precisely one of the  $n \geq 2$  actions  $a_2, \dots, a_n$  must occur in  $\mathcal{L}$  if and only if  $a_1$  occurs in  $\mathcal{L}$

$a_1$  **IFF** ( $a_2$  **OR**  $\cdots$  **OR**  $a_n$ ) : at least one of the  $n \geq 2$  actions  $a_2, \dots, a_n$  must occur in  $\mathcal{L}$  if and only if  $a_1$  occurs in  $\mathcal{L}$

# 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, more later) 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 (Thms. 2 and 3)

VMC (v6.1, released in May 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, more later) 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 (Thms. 2 and 3)

VMC (v6.1, released in May 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, more later) 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 (Thms. 2 and 3)

VMC (v6.1, released in May 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, more later) 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 (Thms. 2 and 3)

VMC (v6.1, released in May 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):

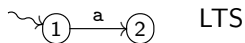
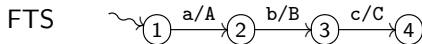
1. new action  $\forall$  feature (to handle more complex feature expressions)
2. dummy transition  $\forall$  action (to verify constraints, ignored when model checking)



! 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



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)

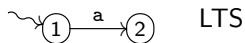
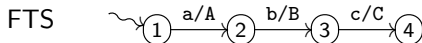


valid product (satisfies a REQ c)

! 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)



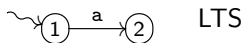
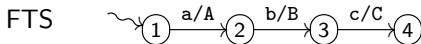
valid product (satisfies a REQ c)

! 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



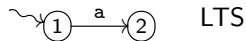
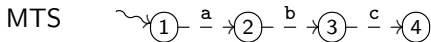
# Cruz: from feature constraints to action constraints

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)

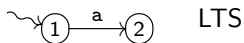
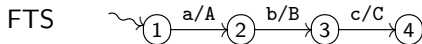


valid product (satisfies a REQ c)

! 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

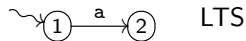
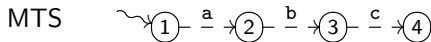
# Crux: from feature constraints to action constraints

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)

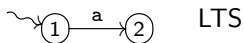
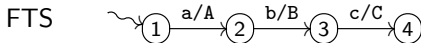


valid product (satisfies a REQ c)

! 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

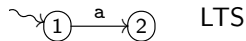
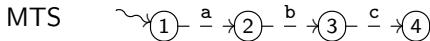
# Crux: from feature constraints to action constraints

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**)

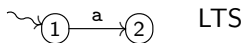
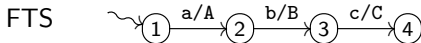


valid product (satisfies a REQ c)

! 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

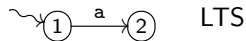
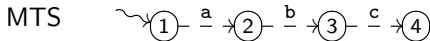
# Crux: from feature constraints to action constraints

From feature model: A requires C



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

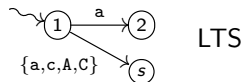
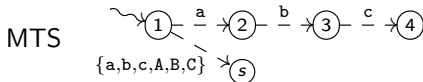
From FTS to MTS (naive): a REQ c



~~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**)

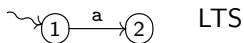
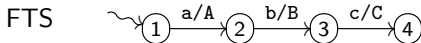


valid product (satisfies a REQ c)

! 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

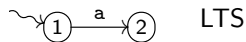
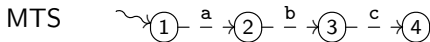
# Crux: from feature constraints to action constraints

From feature model: A requires C



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

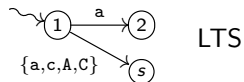
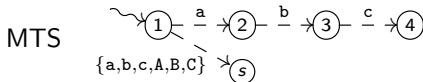
From FTS to MTS (naive): a REQ c



~~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**)

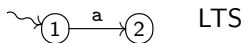
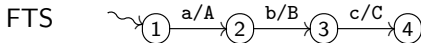


valid product (satisfies a REQ c)

! 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

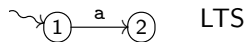
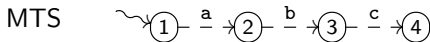
# Crux: from feature constraints to action constraints

From feature model: A requires C



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

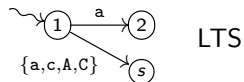
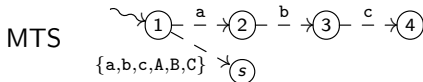
From FTS to MTS (naive): a REQ c



~~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

## Transformation (1/4)

---

Step 1: definition of valid products in terms of features

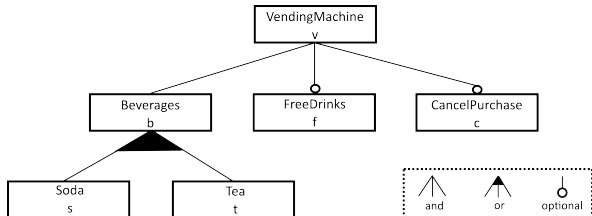
Translate feature model in a set of variability constraints on features

# 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  
}





## 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, for which  $a_1 \text{ OR } \dots \text{ OR } a_n$  can contain either  $a_i$  or its negation  $\sim a_i$ )

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

## 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, for which  $a_1 \text{ OR } \dots \text{ OR } a_n$  can contain either  $a_i$  or its negation  $\sim a_i$ )

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

## 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/ $\phi$ '
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

```
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
```

## 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/ $\phi$ '
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

```
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
```

## 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/ $\phi$ '
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

```
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
```

## 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/ $\phi$ '
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

```
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
```

## 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/ $\phi$ '
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

```
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
```

# 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  
}
```



## 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 paper



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

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

$$\psi ::= \text{true} \mid a(e) \mid \neg\psi \mid \psi \wedge \psi$$
$$\phi ::= \text{true} \mid \neg\phi \mid \phi \wedge \phi \mid \langle\chi\rangle\phi \mid [\chi]\phi \mid E\pi \mid A\pi \mid \mu Y.\phi(Y) \mid \nu Y.\phi(Y)$$
$$\pi ::= [\phi \{ \chi \} U \{ \chi' \} \phi'] \mid [\phi \{ \chi \} U \phi'] \mid [\phi \{ \chi \} W \{ \chi' \} \phi'] \mid [\phi \{ \chi \} W \phi'] \mid X \{ \chi \} \phi \mid F\phi \mid F \{ \chi \} \phi \mid G\phi$$

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

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

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

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

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  $\{a, b\}$

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

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

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)

MTS



Assume  
a OR b



LTS



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

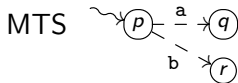
$\Rightarrow p$  is a live state, since a OR b gives rise to a live action set  $\{a, b\}$

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

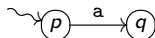
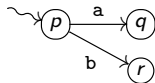
$$S \models \phi \Rightarrow S_p \models \phi \quad \forall \text{ product LTS } S_p \text{ of MTS } S$$

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

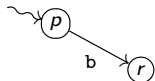
**Live action sets** define **live states** (not occur as final in any product)



Assume  
a OR b



LTS



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

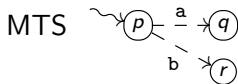
$\Rightarrow p$  is a live state, since a OR b gives rise to a live action set  $\{a, b\}$

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

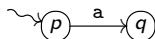
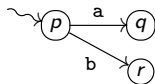
$$S \models \phi \Rightarrow S_p \models \phi \quad \forall \text{product LTS } S_p \text{ of MTS } S$$

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

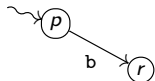
**Live action sets** define **live states** (not occur as final in any product)



Assume  
a OR b



LTS



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

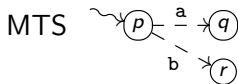
$\Rightarrow p$  is a live state, since a OR b gives rise to a live action set  $\{a, b\}$

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

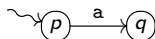
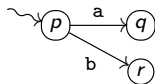
$$S \models \phi \Rightarrow S_p \models \phi \quad \forall \text{product LTS } S_p \text{ of MTS } S$$

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

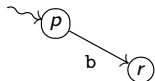
**Live action sets** define **live states** (not occur as final in any product)



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  $\{a, b\}$



## Family-based verification (by preservation)

---

VMC notifies whenever preservation of a verification result is applicable


# Family-based verification (by preservation)

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:**

$$[\text{behaviour}] \text{ not } E[\text{true} \{ \text{not tea} \} U \{ \text{serveTea} \} \text{true}]$$

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**

$$[\text{behaviour}] \text{ not } E [\text{true} \{ \text{not tea} \} U \{ \text{serveTea} \} \text{true}]$$

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

## 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


# 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



Kandisky 1908

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

<a href="#">product_1+Soda+open+pay+soda</a>	Formula evaluates	TRUE
<a href="#">product_10+CancelPurchase+FreeDrinks+Tea+cancel+free+takeFree+tea</a>	Formula evaluates	TRUE
<a href="#">product_11+FreeDrinks+Soda+Tea+free+soda+takeFree+tea</a>	Formula evaluates	TRUE
<a href="#">product_12+CancelPurchase+FreeDrinks+Soda+Tea+cancel+free+soda+takeFree+tea</a>	Formula evaluates	TRUE
<a href="#">product_2+CancelPurchase+Soda+cancel+open+pay+soda</a>	Formula evaluates	FALSE
<a href="#">product_3+Tea+open+pay+tea</a>	Formula evaluates	TRUE
<a href="#">product_4+CancelPurchase+Tea+cancel+open+pay+tea</a>	Formula evaluates	FALSE
<a href="#">product_5+Soda+Tea+open+pay+soda+tea</a>	Formula evaluates	TRUE
<a href="#">product_6+CancelPurchase+Soda+Tea+cancel+open+pay+soda+tea</a>	Formula evaluates	FALSE
<a href="#">product_7+FreeDrinks+Soda+free+soda+takeFree</a>	Formula evaluates	TRUE
<a href="#">product_8+CancelPurchase+FreeDrinks+Soda+cancel+free+soda+takeFree</a>	Formula evaluates	TRUE
<a href="#">product_9+FreeDrinks+Tea+free+takeFree+tea</a>	Formula evaluates	TRUE

**Logic Formula for all Products**

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

*Whenever pay occurs, eventually takePaid occurs*

## Conclusions and future work

We presented an automatic technique to transform FTS into MTS (with variability constraints and in the format accepted by VMC)

1. helps to better understand the differences between these models (i.e. variability constraints in terms of features or actions)
2. paves the way to compare their modelling and analysis features

Future work:

1. Perform a quantitative evaluation of the expressivity, complexity and scalability of both approaches (including the model checkers)
2. Prove the correctness and complexity of the transformation
- ? Develop a front-end for VMC that implements the transformation

## Conclusions and future work

We presented an automatic technique to transform FTS into MTS (with variability constraints and in the format accepted by VMC)

1. helps to better understand the differences between these models (i.e. variability constraints in terms of features or actions)
2. paves the way to compare their modelling and analysis features

Future work:

1. Perform a quantitative evaluation of the expressivity, complexity and scalability of both approaches (including the model checkers)
2. Prove the correctness and complexity of the transformation
- ? Develop a front-end for VMC that implements the transformation



Thanks!

---

