From Featured Transition Systems to Modal Transition Systems with Variability Constraints: A Model Transformation

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

Final General Meeting CINA
Civitanova Marche, 19–21/1/2016

Outline

1. Background: Software Product Line Engineering
   - Featured Transition Systems
   - Modal Transition Systems with variability constraints
   - VMC: Variability Model Checker
2. SEFM'15 paper: 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-ACTL$^{\square}$/v-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

$\Rightarrow$ 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

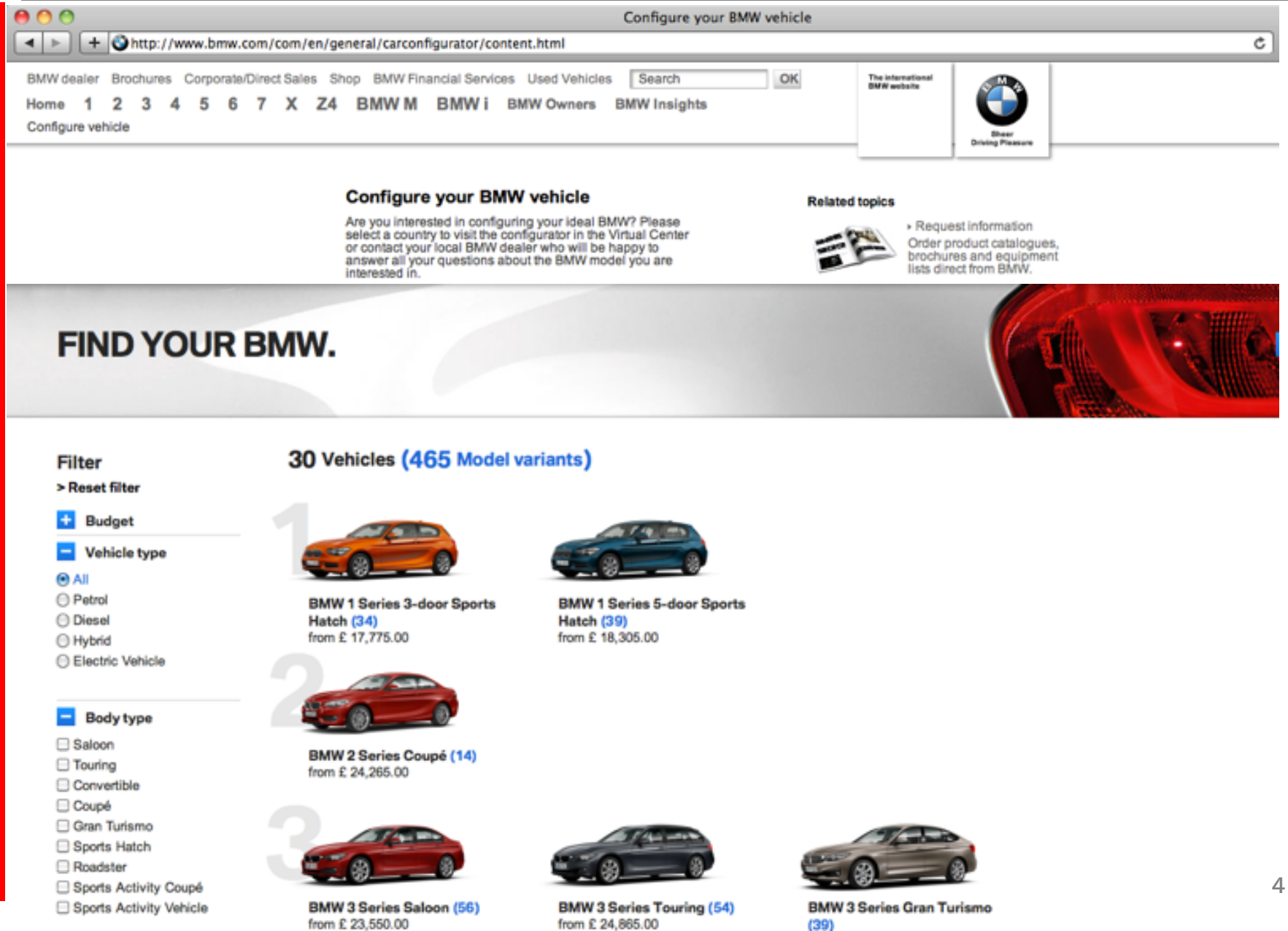
$\Rightarrow$ maximise commonalities whilst minimising cost of variations (i.e. of individual products)

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
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)
BMW 1 Series 3-door Sports Hatch (34)
from £ 17,775.00
BMW 1 Series 5-door Sports Hatch (39)
from £ 18,305.00
BMW 2 Series Coupé (14)
from £ 24,265.00
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

$\Rightarrow$ 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*, 2015

  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

$\Rightarrow$ 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*, 2015

  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 @ SPLC'08, Asirelli et al. @ iFM'10, SPLC'11, ter Beek et al. @ *JLAMP*, 2015

  Variability Model Checker VMC

  ter Beek et al. @ FM'12, SPLC'12, SPLat'14
- Featured Transition Systems (FTS)

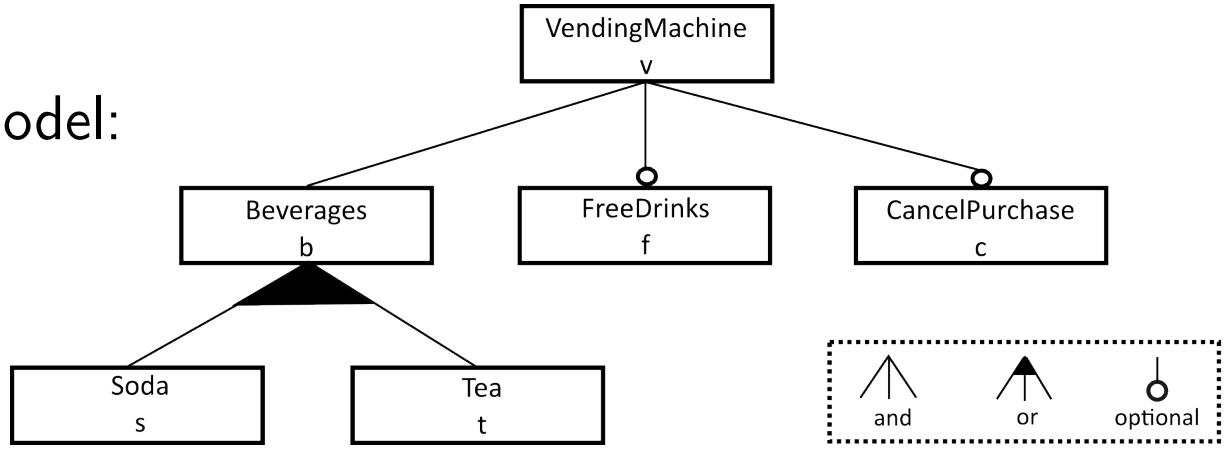
  Classen et al. @ ICSE'10, *IEEE TSE*, 2013, *Sci. Comput. Program.*, 2014

  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:

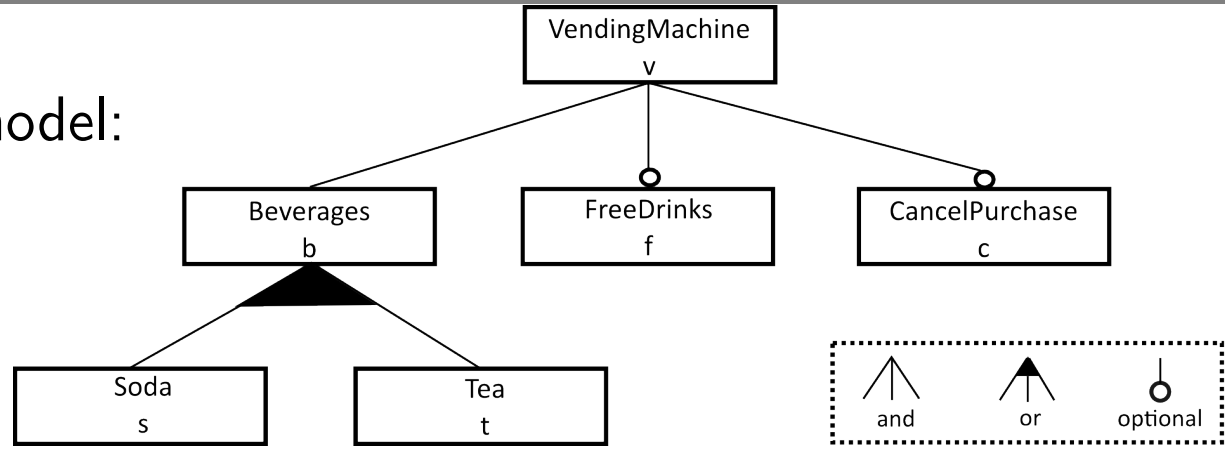


12 valid products

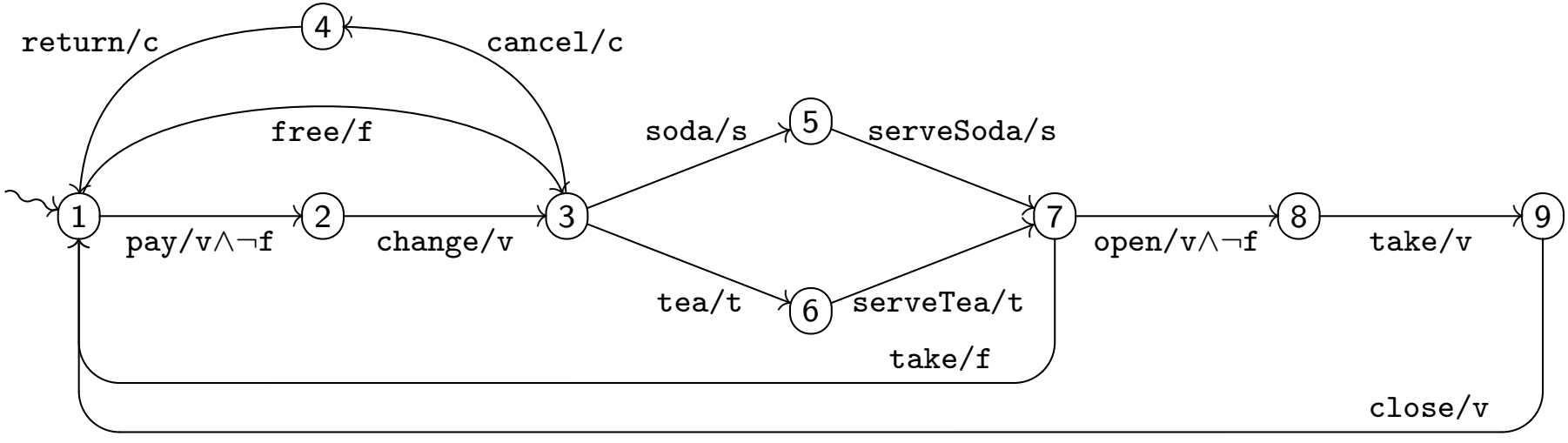
e.g. $\{\mathtt{v},\mathtt{b},\mathtt{s},\mathtt{t}\}$, $\{\mathtt{v},\mathtt{b},\mathtt{s},\mathtt{c}\}$

FTS of example SPL

Feature model:

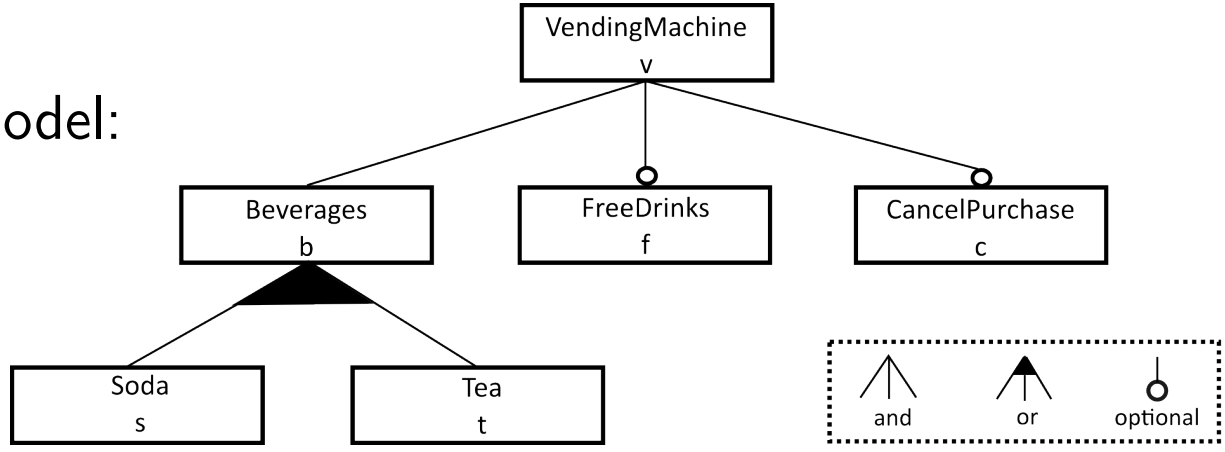


FTS of 12 valid products (LTSs) e.g. {v,b,s,t}, {v,b,s,c}

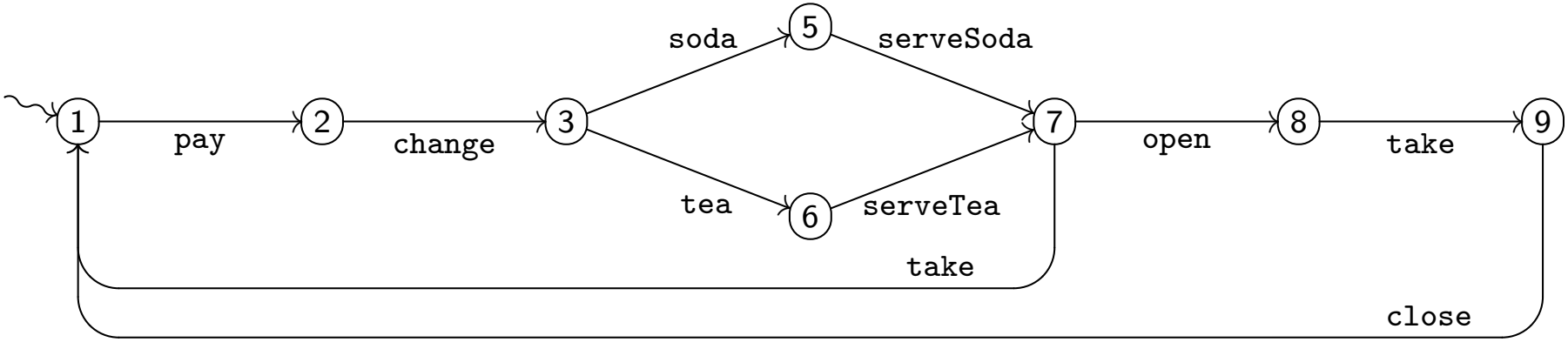


FTS of example SPL

Feature model:

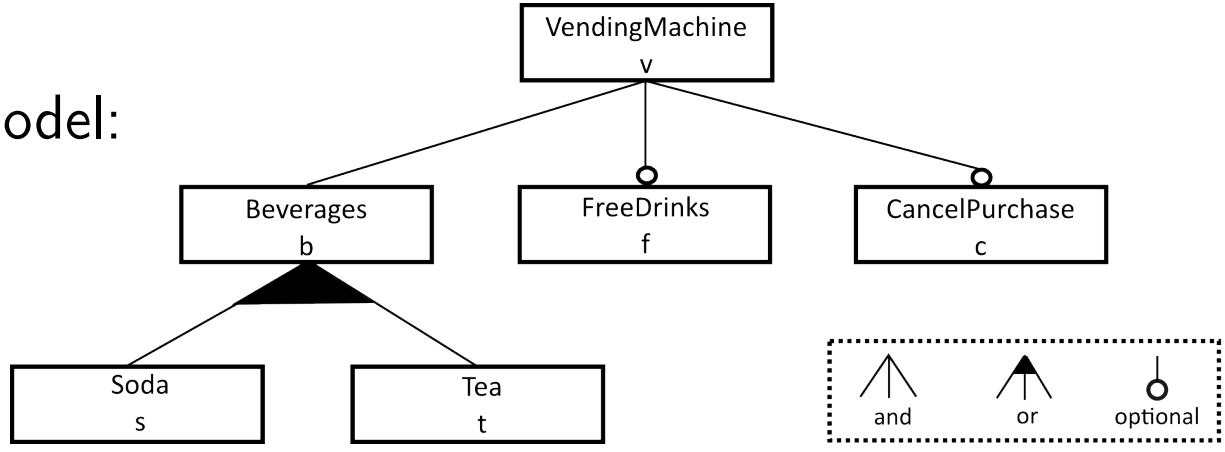


e.g. {v,b,s,t}, {v,b,s,c}

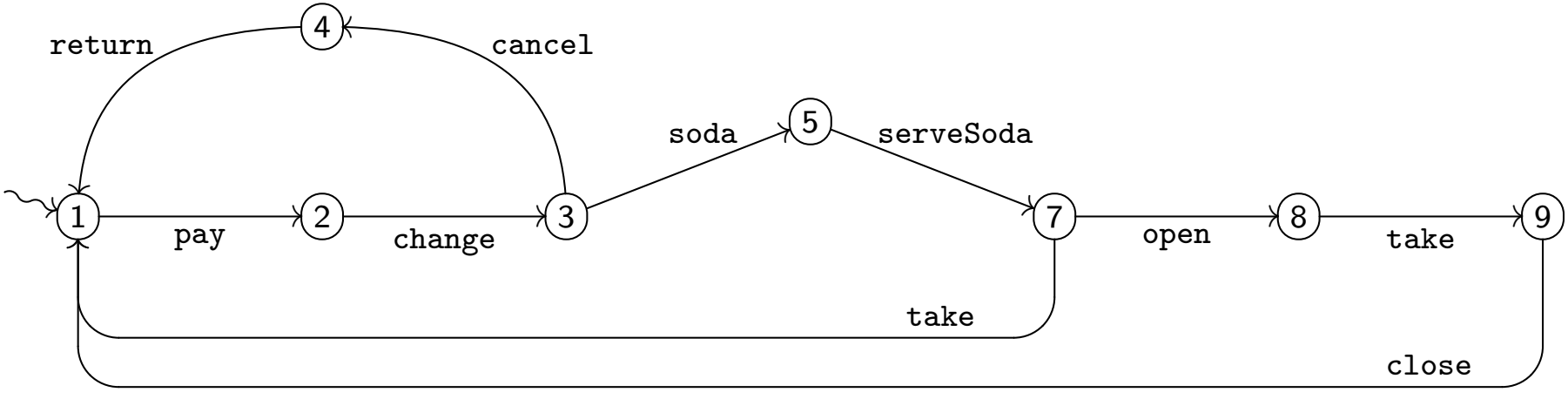


FTS of example SPL

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 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*, 2015

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 $(Q, A, \overline{q}, \rightarrow)$ with states Q, actions A, initial state $\overline{q}$ and transitions $\rightarrow \subseteq Q \times A \times Q$

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 $\dashrightarrow \equiv \rightarrow_{\Diamond} \setminus \rightarrow_{\Box}$: 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

$\Rightarrow$ Each selection gives rise to a different variant

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

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

1. every $q \in Q_i$ is reachable in $\mathcal{P}_i$ from $\overline{q}$ via transitions from δ_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 $\xrightarrow{a} \rightsquigarrow \xrightarrow{a}$ 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

$\Rightarrow$ Each selection gives rise to a different variant

Let $(Q, A, \overline{q}, \delta^{\Diamond}, \delta^{\Box})$ be a coherent MTS, i.e. $\exists \xrightarrow{a} \implies \nexists \longrightarrow^{a}$

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

1. every $q \in Q_i$ is reachable in $\mathcal{P}_i$ from $\overline{q}$ via transitions from δ_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 $\xrightarrow{a} \rightsquigarrow \longrightarrow^{a}$ and $\not\xrightarrow{a}$ not allowed

Variability constraints (implemented in VMC)

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

a_1 ALT $\cdots$ ALT a_n : precisely one among the $n \geq 2$ actions $a_1, ..., a_n$ 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 $b_1, ..., b_n$ 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 $a_2, ..., a_n$ 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 $a_2, ..., a_n$ is reachable in $\mathcal{L}$ if a_1 is reachable 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)

   ter Beek, Fantechi, Gnesi, Mazzanti @ *JLAMP*, 2015

VMC (v6.2, 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, 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)

   ter Beek, Fantechi, Gnesi, Mazzanti @ *JLAMP*, 2015

VMC (v6.2, 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, 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)

   ter Beek, Fantechi, Gnesi, Mazzanti @ *JLAMP*, 2015

VMC (v6.2, 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, 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)

   ter Beek, Fantechi, Gnesi, Mazzanti @ *JLAMP*, 2015

VMC (v6.2, 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

FTS

LTS

valid product (namely A ∧ C ∧ ¬B)

From FTS to MTS (naive): a REQ c

MTS

LTS

~~valid product~~ (violates 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)

MTS

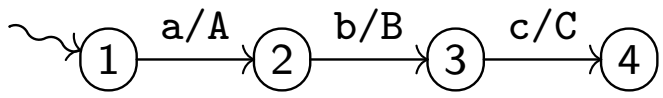
LTS

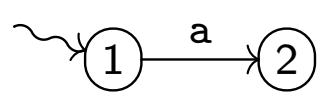
valid product (satisfies a REQ c)

! 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 



LTS

valid product (namely A ∧ C ∧ ¬B)

From FTS to MTS (naive): a REQ c

MTS

LTS

~~valid product~~ (violates 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)

MTS

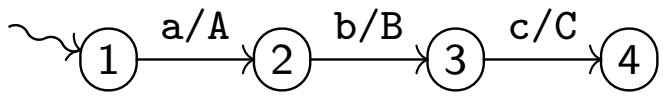
LTS

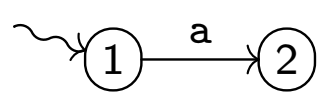
valid product (satisfies a REQ c)

! 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 



LTS

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

From FTS to MTS (naive): `a REQ c`

MTS

LTS

~~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

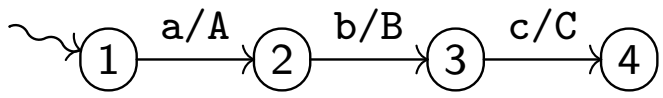
LTS

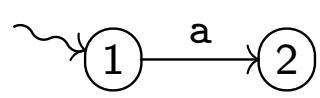
valid product (satisfies `a REQ c`)

! 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`

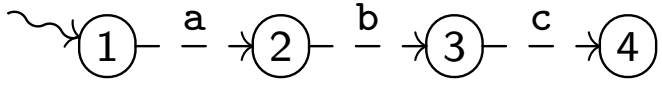
FTS 

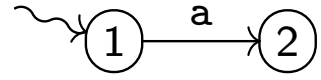


LTS

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

From FTS to MTS (naive): `a REQ c`

MTS 



LTS

~~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

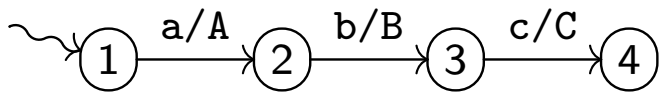
LTS

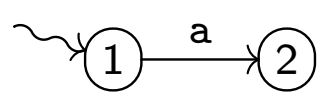
valid product (satisfies `a REQ c`)

! 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`

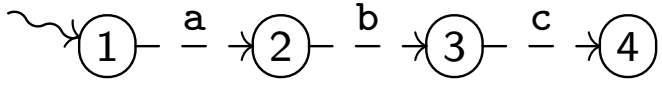
FTS 

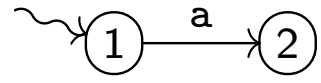


LTS

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

From FTS to MTS (naive): `a REQ c`

MTS 



LTS

~~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

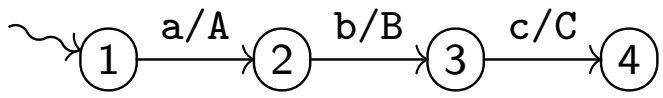
LTS

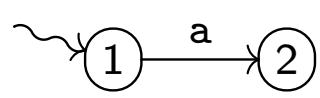
valid product (satisfies `a REQ c`)

! 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`

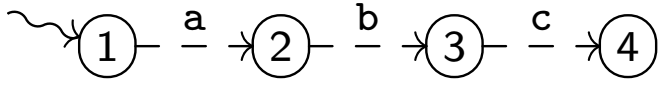
FTS 

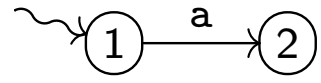


LTS

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

From FTS to MTS (naive): `a REQ c`

MTS 



LTS

~~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

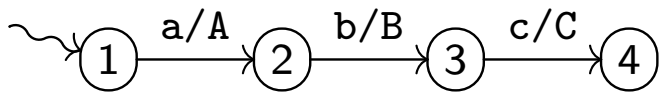
LTS

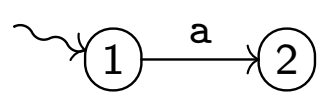
valid product (satisfies `a REQ c`)

! 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`

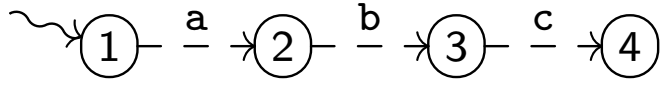
FTS 

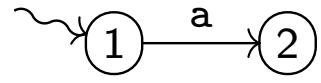


LTS

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

From FTS to MTS (naive): `a REQ c`

MTS 

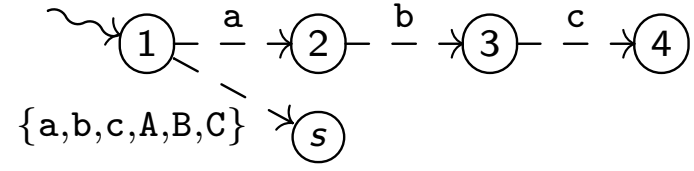


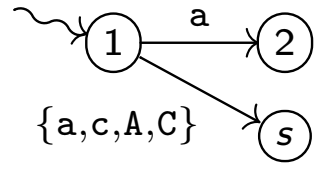
LTS

~~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 



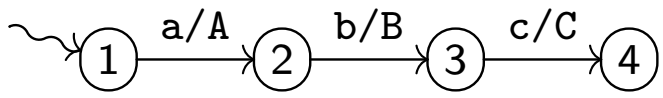
LTS

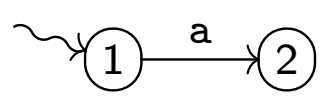
valid product (satisfies `a REQ c`)

! 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`

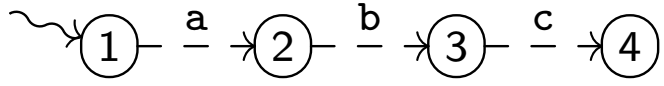
FTS 

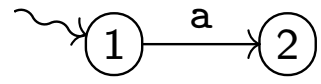


LTS

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

From FTS to MTS (naive): `a REQ c`

MTS 

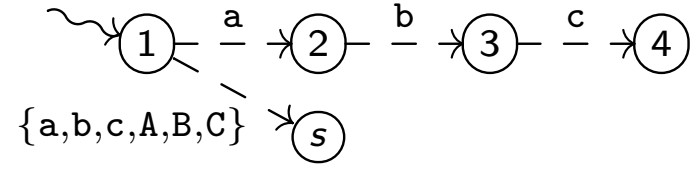


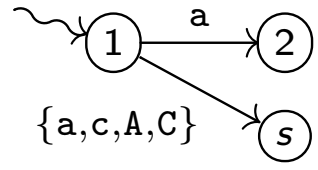
LTS

~~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 



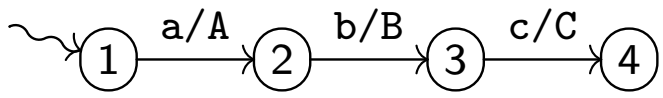
LTS

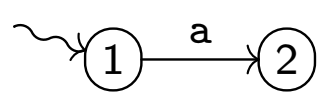
valid product (satisfies `a REQ c`)

! 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`

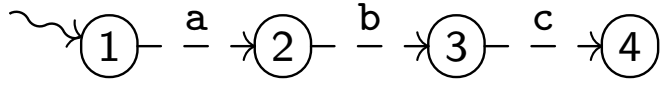
FTS 

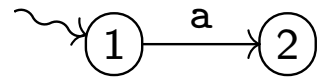


LTS

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

From FTS to MTS (naive): a `REQ` c

MTS 

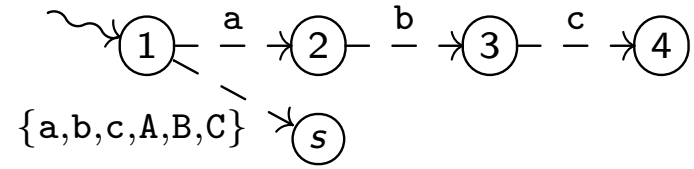


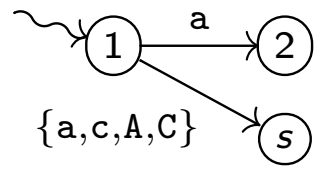
LTS

~~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 



LTS

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

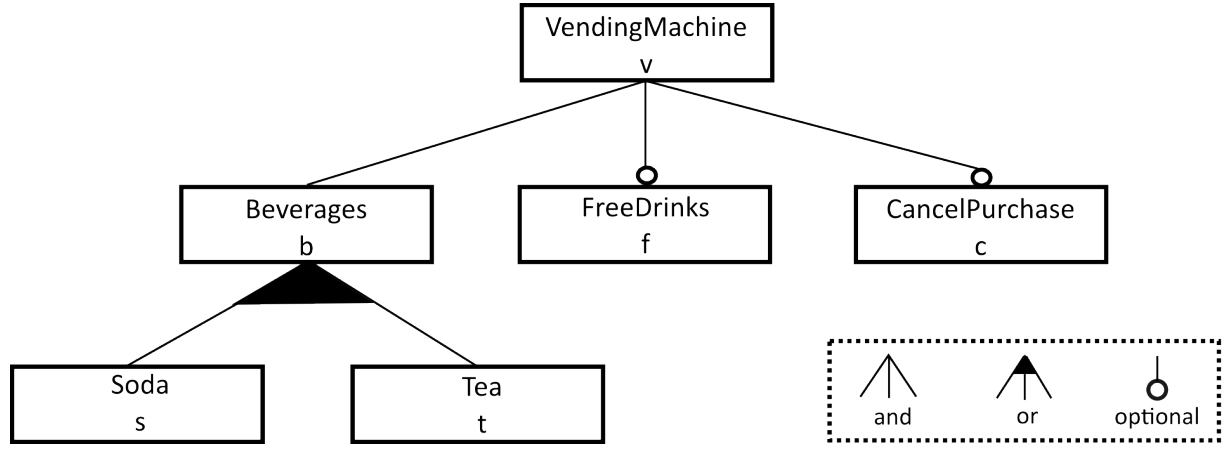
Translate feature model in a set of variability constraints on features

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

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

```
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)

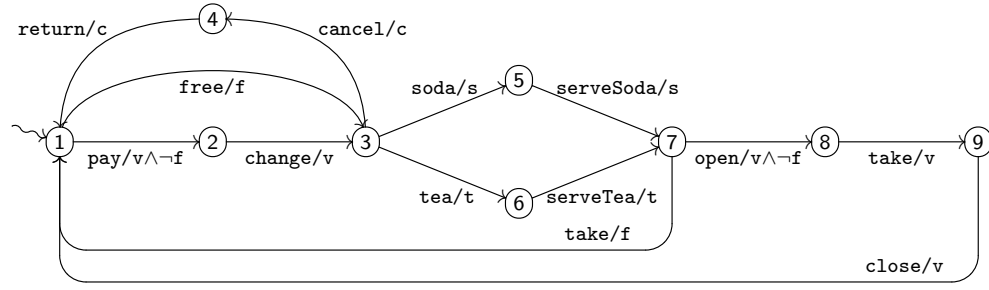
Step 4: definition of live action sets and introduction of must transitions

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 $[\![S]\!]$ denote set of valid product configurations for S
Let FTS(S) and 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 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. *FTS(S) and the set of LTS obtained by omitting the dummy transitions from the LTS in MTS(S') are equal*

Proof.

Sketch in SEFM'15 paper. Full proof in forthcoming journal paper. □

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

Action formulae ψ, state formulae ϕ, path formulae π

$\psi ::= \mathit{true} \mid a \mid \cancel{a(e)} \mid \neg\psi \mid \psi \wedge \psi$ ter Beek, Gnesi, Mazzanti @ PSI'14

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

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

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

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

$F^{\Box}\ \{\chi\}\ \phi$: 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$^{\Box}$/v-ACTLive$^{\Box}$

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

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)

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-ACTL$^{\Box}$/v-ACTLive$^{\Box}$

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

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)

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 $\{a, b\}$

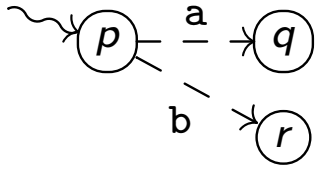
Preservation of formulae in v-ACTL$^{\Box}$/v-ACTLive$^{\Box}$

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

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

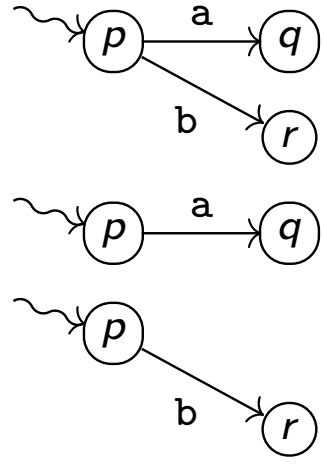
Live action sets define live states (not occur as final in any product)

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 $\{a, b\}$

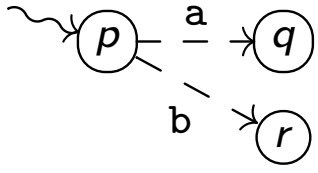
Preservation of formulae in v-ACTL$^{\Box}$/v-ACTLive$^{\Box}$

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

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

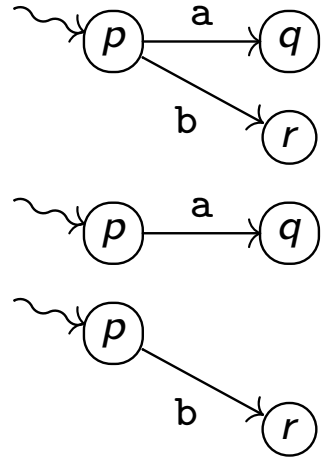
Live action sets define live states (not occur as final in any product)

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 $\{a, b\}$

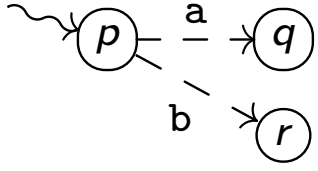
Preservation of formulae in v-ACTL$^{\Box}$/v-ACTLive$^{\Box}$

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

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

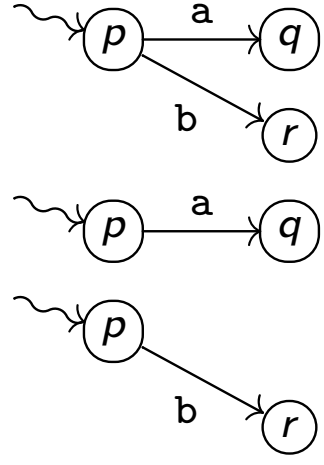
Live action sets define live states (not occur as final in any product)

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 $\{a, b\}$

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

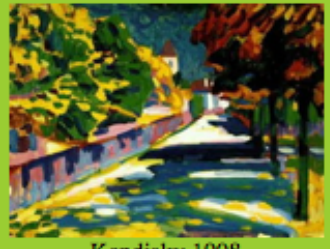
Explore the MTS

Draw Family MTS

Generate Products

Welcome

Quit



Kandisky 1908

The Formula:

[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

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

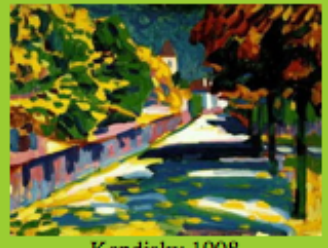
VMC lists for each product the action labels of all may transitions that have been preserved (as must transitions) in that product

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

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

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 of 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 model 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 of 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 model transformation
? Develop a front-end for VMC that implements the transformation

Thanks!

Relevant publications during CINA

- M.H. ter Beek, A. Fantechi, S. Gnesi, and F. Mazzanti, Modelling and Analysing Variability in Product Families: Model Checking of Modal Transition Systems with Variability Constraints. *Journal of Logical and Algebraic Methods in Programming* (2015), http://dx.doi.org/10.1016/j.jlamp.2015.11.006
- M.H. ter Beek, F. Damiani, S. Gnesi, F. Mazzanti, and L. Paolini. From Featured Transition Systems to Modal Transition Systems with Variability Constraints. In Proceedings SEFM'15, York, UK (R. Calinescu and B. Rumpe, eds.), LNCS 9276, Springer, 2015, 344–359.
- M.H. ter Beek, S. Gnesi, and F. Mazzanti, Model Checking Value-Passing Modal Specifications. In Perspectives of System Informatics - Revised selected papers of PSI'14, St. Petersburg, Russia (A. Voronkov and I. Virbitskaite, eds.), LNCS 8974, Springer, 2015, 304–319.
- M.H. ter Beek and F. Mazzanti, VMC: Recent Advances and Challenges Ahead. In Proceedings of the Workshop on Software Product Line Analysis Tools (SPLat'14), Florence, Italy (A. Legay and E. de Vink, eds.), Volume 2 of the Proceedings of SPLC'14, Florence, Italy, ACM, 2014, 70–77.