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

Maurice ter Beek

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

Modelling and Analysis of Variability in Software Product Lines
ISTI–CNR, Pisa, June 26–30, 2017

Outline

1. Background: Software Product Line Engineering
2. Formal Methods and Analysis in Software Product Line Engineering
   - Recall: Featured Transition Systems
   - Modal Transition Systems with variability constraints
   - Transformation: from feature constraints to action constraints
3. v-ACTL: variability-aware, action-based CTL
   - Preservation of formulae in v-ACTL$^{\Box}$/v-ACTLive$^{\Box}$
4. VMC: Variability Model Checker
   - Family-based and product-based verification with VMC
   - A value-passing modal process algebra
   - An example family of Bike-Sharing Systems (BSS)

Formal methods and tools in SPLE

Computer-aided analysis of feature models

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

Goal: rigorously establish critical requirements of (software) systems

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

Widely used behavioural SPL models with dedicated model checkers

- Modal Transition Systems (MTS) with variability constraints

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

  Variability Model Checker VMC

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

- Featured Transition Systems (FTS)

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

  SNIP, ProVeLines, NuSMV extension

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

FTS for SPLE

Recall (atomic propositions used for verification purposes):

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

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

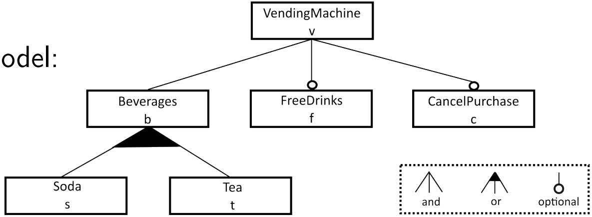
An FTS adds to this a feature model and feature expressions:

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

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

FTS of example SPL: a vending machine

Feature model:

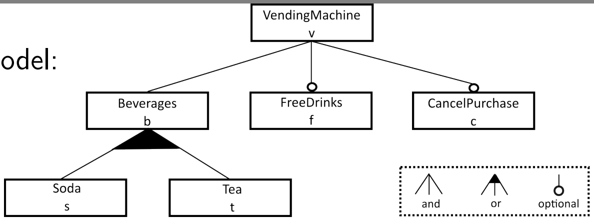


12 valid products

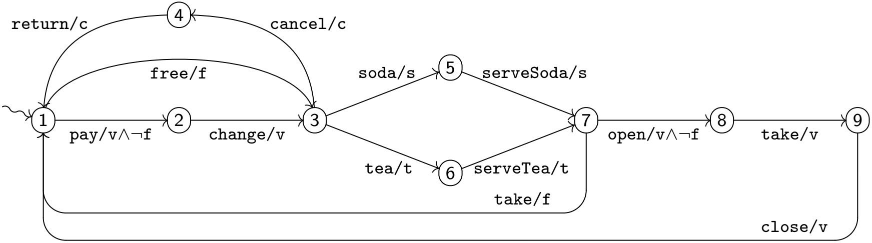
e.g. $\{\mathtt{v,b,s,t}\}$, $\{\mathtt{v,b,s,c}\}$

FTS of example SPL: a vending machine

Feature model:

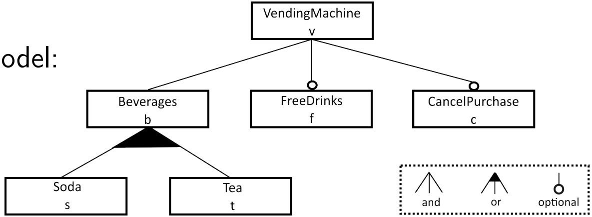


FTS of 12 valid products (LTS) e.g. $\{\mathtt{v,b,s,t}\}$, $\{\mathtt{v,b,s,c}\}$

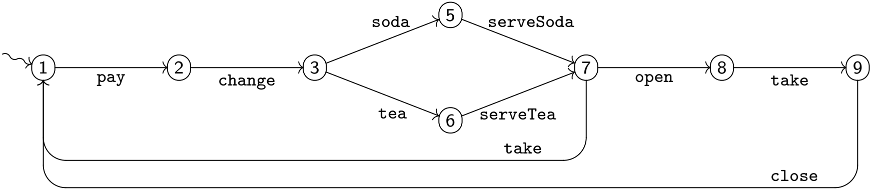


FTS of example SPL: a vending machine

Feature model:

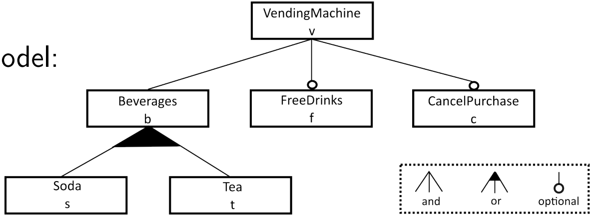


e.g. $\color{red}{\{\texttt{v,b,s,t}\}}$, $\{\texttt{v,b,s,c}\}$

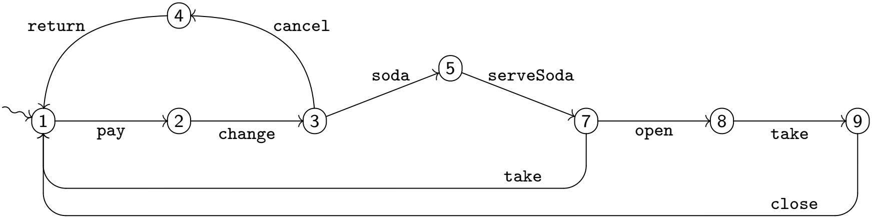


FTS of example SPL: a vending machine

Feature model:



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



MTS for SPLE

Main ingredient: Modal Transition Systems (MTS)

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

  Larsen, Thomsen @ LICS'88

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

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

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

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

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

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

MTS with...

Recall:

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

Next we define MTS with variability constraints:

A Modal Transition System (MTS) is a quintuple $(Q, A, \overline{q}, \rightarrow_\Box, \rightarrow_\Diamond)$ such that $(Q, A, \overline{q}, \rightarrow_\Box \cup \rightarrow_\Diamond)$ is an LTS, called its underlying LTS

An MTS has two distinct transition relations

1. may transition relation $\rightarrow_\Diamond \subseteq Q \times A \times Q$: possible transitions
2. must transition relation $\rightarrow_\Box \subseteq Q \times A \times Q$: required transitions

By definition, any required transition is also possible: $\rightarrow_\Box \subseteq \rightarrow_\Diamond$
(denote $\dashrightarrow \equiv \rightarrow_\Diamond \setminus \rightarrow_\Box$: optional transitions)

...variability constraints (accepted by VMC)

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

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

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

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

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

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

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

Derive products (implemented in VMC)

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

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

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

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

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 $\overset{a}{\dashrightarrow} \rightsquigarrow \xrightarrow{a}$ and $\cancel{\overset{a}{\dashrightarrow}}$ 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. $\exists \overset{a}{\dashrightarrow} \implies \nexists \xrightarrow{a}$

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 $\overset{a}{\dashrightarrow} \;\rightsquigarrow\; \xrightarrow{a}$ and $\cancel{\overset{a}{\dashrightarrow}}$ not allowed

VMC: Variability Model Checker

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

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

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

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

2. A logic property (expressed in variability-aware ACTL, 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

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

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

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

VMC: Variability Model Checker

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

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

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

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

2. A logic property (expressed in variability-aware ACTL, 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

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

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

`http://fmt.isti.cnr.it/vmc/`

VMC: Variability Model Checker

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

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

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

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

2. A logic property (expressed in variability-aware ACTL, 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

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

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

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

VMC: Variability Model Checker

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

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

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

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

2. A logic property (expressed in variability-aware ACTL, 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

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

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

`http://fmt.isti.cnr.it/vmc/`

Crux: from feature constraints to action constraints

From feature model: `A` requires `C`

FTS

LTS

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

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 $\rightsquigarrow 1 \xrightarrow{\texttt{a/A}} 2 \xrightarrow{\texttt{b/B}} 3 \xrightarrow{\texttt{c/C}} 4$

$\rightsquigarrow 1 \xrightarrow{\texttt{a}} 2$ LTS

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

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

MTS $\rightsquigarrow 1 \dashrightarrow{\texttt{a}} 2 \dashrightarrow{\texttt{b}} 3 \dashrightarrow{\texttt{c}} 4$

$\rightsquigarrow 1 \xrightarrow{\texttt{a}} 2$ 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 $\rightsquigarrow 1 \dashrightarrow{\texttt{a}} 2 \dashrightarrow{\texttt{b}} 3 \dashrightarrow{\texttt{c}} 4$, $1 \dashrightarrow{\{\texttt{a,b,c,A,B,C}\}} s$

$\rightsquigarrow 1 \xrightarrow{\texttt{a}} 2$, $1 \xrightarrow{\{\texttt{a,c,A,C}\}} s$ LTS

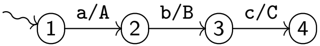
valid product (satisfies `a REQ c`)

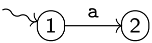
! Consistency guarantees that whenever a c-labelled may transition from the initial state is preserved in this LTS, then also any other reachable c-labelled may transition must be preserved

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

MTS

LTS

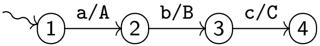
valid product (satisfies `a REQ c`)

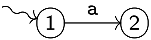
! Consistency guarantees that whenever a *c*-labelled may transition from the initial state is preserved in this LTS, then also any other reachable *c*-labelled may transition must be preserved

Crux: from feature constraints to action constraints

From feature model: `A` requires `C`

FTS



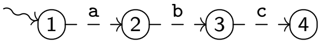


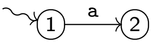
LTS

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

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

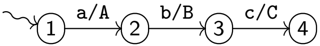
valid product (satisfies `a REQ c`)

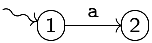
! Consistency guarantees that whenever a *c*-labelled may transition from the initial state is preserved in this LTS, then also any other reachable *c*-labelled may transition must be preserved

Crux: from feature constraints to action constraints

From feature model: `A` requires `C`

FTS



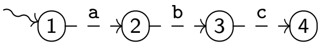


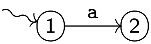
LTS

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

MTS

LTS

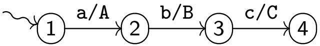
valid product (satisfies `a REQ c`)

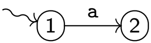
! Consistency guarantees that whenever a c-labelled may transition from the initial state is preserved in this LTS, then also any other reachable c-labelled may transition must be preserved

Crux: from feature constraints to action constraints

From feature model: `A` requires `C`

FTS



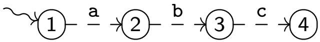


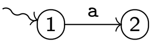
LTS

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

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

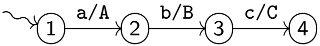
valid product (satisfies `a REQ c`)

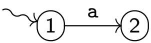
! Consistency guarantees that whenever a c-labelled may transition from the initial state is preserved in this LTS, then also any other reachable c-labelled may transition must be preserved

Crux: from feature constraints to action constraints

From feature model: `A` requires `C`

FTS



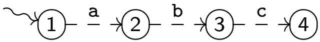


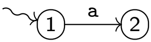
LTS

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

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

MTS





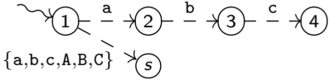
LTS

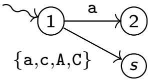
~~valid product~~ (violates `a REQ c`)

From FTS to MTS (solution):

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

MTS





LTS

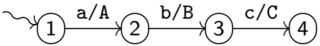
valid product (satisfies `a REQ c`)

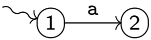
! Consistency guarantees that whenever a c-labelled may transition from the initial state is preserved in this LTS, then also any other reachable c-labelled may transition must be preserved

Crux: from feature constraints to action constraints

From feature model: `A` requires `C`

FTS



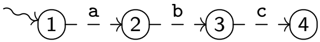


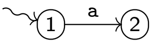
LTS

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

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

MTS





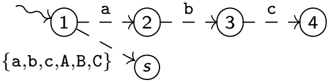
LTS

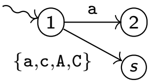
~~valid product~~ (violates `a REQ c`)

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

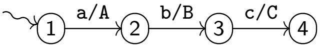
valid product (satisfies `a REQ c`)

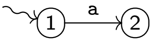
! Consistency guarantees that whenever a c-labelled may transition from the initial state is preserved in this LTS, then also any other reachable c-labelled may transition must be preserved

Crux: from feature constraints to action constraints

From feature model: `A` requires `C`

FTS



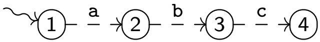


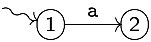
LTS

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

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

MTS





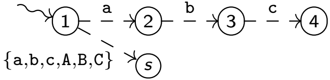
LTS

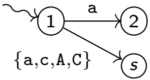
~~valid product~~ (violates `a REQ c`)

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

Model Transformation (1/4)

Step 1: definition of valid products in terms of features

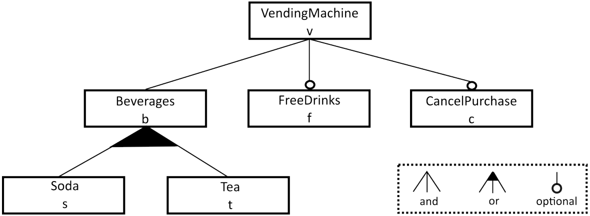
Translate feature model in a set of variability constraints on features

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 ‘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)

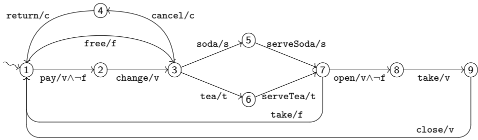
Step 4: definition of live action sets and introduction of must transitions

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 $\mathtt{F}$ for each feature $\mathtt{F}$ 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. $\square$

Recall: Action-based CTL

Syntax over action formulas (boolean compositions of actions, denoted by ψ), state formulas (ϕ) and path formulas (π)

$$
\begin{aligned}
\phi &::= \mathit{true} \mid \neg\phi \mid \phi \wedge \phi \mid [\psi]\,\phi \mid \langle\psi\rangle\,\phi \mid E\,\pi \mid A\,\pi \mid \\
&\qquad \mu\, Y.\phi(Y) \mid \nu\, Y.\phi(Y) \\
\pi &::= X\,\{\psi\}\,\phi \mid [\phi\,\{\psi\}\,U\,\{\psi'\}\,\phi'] \mid [\phi\,\{\psi\}\,W\,\{\psi'\}\,\phi'] \mid \\
&\qquad [\phi\,\{\psi\}\,U\,\phi'] \mid [\phi\,\{\psi\}\,W\,\phi'] \mid F\,\phi \mid F\,\{\psi\}\,\phi \mid G\,\phi
\end{aligned}
$$

(Y is a propositional variable, $\phi(Y)$ is syntactically monotone in Y)

μ and ν: recursion ("finite looping"/"liveness" and "looping"/"safety")

X, U, W, F: action-based neXt, Until, Weak until, Future ("eventually")

Recall: Action-based CTL

Syntax over action formulas (boolean compositions of actions, denoted by ψ), state formulas (ϕ) and path formulas (π)

$$
\begin{aligned}
\phi &::= \mathit{true} \mid \neg\phi \mid \phi\wedge\phi \mid [\psi]\,\phi \mid \langle\psi\rangle\,\phi \mid E\,\pi \mid A\,\pi \mid \\
&\quad\ \mu\,Y.\phi(Y) \mid \nu\,Y.\phi(Y) \\
\pi &::= X\,\{\psi\}\;\phi \mid [\phi\;\{\psi\}\,U\,\{\psi'\}\;\phi'] \mid [\phi\;\{\psi\}\,W\,\{\psi'\}\;\phi'] \mid \\
&\quad\ [\phi\;\{\psi\}\,U\;\phi'] \mid [\phi\;\{\psi\}\,W\;\phi'] \mid F\,\phi \mid F\,\{\psi\}\;\phi \mid G\,\phi
\end{aligned}
$$

(Y is a propositional variable, $\phi(Y)$ is syntactically monotone in Y)

μ and ν: recursion ("finite looping"/"liveness" and "looping"/"safety")

X, U, W, F: action-based neXt, Until, Weak until, Future ("eventually")

Recall: Action-based CTL

Syntax over action formulas (boolean compositions of actions, denoted by ψ), state formulas (ϕ) and path formulas (π)

$$\begin{aligned}\phi \; ::= \; & \mathit{true} \;\mid\; \neg\,\phi \;\mid\; \phi \wedge \phi \;\mid\; [\psi]\,\phi \;\mid\; \langle\psi\rangle\,\phi \;\mid\; E\,\pi \;\mid\; A\,\pi \;\mid \\ & \mu\,Y.\phi(Y) \;\mid\; \nu\,Y.\phi(Y) \\ \pi \; ::= \; & X\,\{\psi\}\;\phi \;\mid\; [\phi\;\{\psi\}\,U\,\{\psi'\}\;\phi'] \;\mid\; [\phi\;\{\psi\}\,W\,\{\psi'\}\;\phi'] \;\mid \\ & [\phi\;\{\psi\}\,U\;\phi'] \mid [\phi\;\{\psi\}\,W\;\phi'] \mid F\,\phi \;\mid\; F\,\{\psi\}\;\phi \;\mid\; G\,\phi\end{aligned}$$

(Y is a propositional variable, $\phi(Y)$ is syntactically monotone in Y)

μ and ν: recursion (“finite looping”/“liveness” and “looping”/“safety”)

X, U, W, F: action-based neXt, Until, Weak until, Future (“eventually”)

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

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

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

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

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

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

Informal semantics of v-ACTL

$[\psi]\,\phi$ in all next states reachable by a may transition executing an action satisfying ψ, ϕ holds

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

$\langle\psi\rangle\,\phi \equiv \neg[\psi]\,\neg\phi$ a next state exists, reachable by a may transition executing an action satisfying ψ, in which ϕ holds

$\langle\psi\rangle^{\Box}\phi \equiv \neg[\psi]^{\Box}\neg\phi$ a next state exists, reachable by a must transition executing an action satisfying ψ, in which ϕ holds

($\langle\psi\rangle^{\Box}$ and $[\psi]^{\Box}$ represent the classic deontic modalities O and P)

Informal semantics of v-ACTL

$[\psi]\,\phi$ in all next states reachable by a may transition executing an action satisfying ψ, ϕ holds

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

$\langle\psi\rangle\,\phi \equiv \neg[\psi]\,\neg\phi$ a next state exists, reachable by a may transition executing an action satisfying ψ, in which ϕ holds

$\langle\psi\rangle^{\Box}\phi \equiv \neg[\psi]^{\Box}\neg\phi$ a next state exists, reachable by a must transition executing an action satisfying ψ, in which ϕ holds

($\langle\psi\rangle^{\Box}$ and $[\psi]^{\Box}$ represent the classic deontic modalities O and P)

Informal semantics of v-ACTL

$[\psi]\,\phi$ in all next states reachable by a may transition executing an action satisfying ψ, ϕ holds

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

$\langle\psi\rangle\,\phi \equiv \neg[\psi]\,\neg\phi$ a next state exists, reachable by a may transition executing an action satisfying ψ, in which ϕ holds

$\langle\psi\rangle^{\Box}\phi \equiv \neg[\psi]^{\Box}\neg\phi$ a next state exists, reachable by a must transition executing an action satisfying ψ, in which ϕ holds

($\langle\psi\rangle^{\Box}$ and $[\psi]^{\Box}$ represent the classic deontic modalities O and P)

Informal semantics of v-ACTL (cont.)

A full path is a path that cannot be extended further ($q \cdots$ or $q \not\rightarrow$)

- $E\,\pi$ there exists a full path on which π holds
- $A\,\pi$ on all possible full paths, π holds

- $F\,\phi$ there exists a future state in which ϕ holds
- $F^{\Box}\,\phi$ there exists a future state in which ϕ holds and all transitions until that state are must transitions
- $F\,\{\psi\}\,\phi$ there exists a future state, reached by an action satisfying ψ, in which ϕ holds
- $F^{\Box}\,\{\psi\}\,\phi$ there exists a future state, reached by an action satisfying ψ, in which ϕ holds and all transitions until that state are must transitions

$G\,\phi \equiv \neg F\,\neg\phi$ the path is a full path on which ϕ holds in all states

$AG\,\phi \equiv \neg EF\,\neg\phi$ in all states on all paths, ϕ holds

Informal semantics of v-ACTL (cont.)

A full path is a path that cannot be extended further ($q \cdots$ or $q \not\rightarrow$)

- $E\,\pi$ — there exists a full path on which π holds
- $A\,\pi$ — on all possible full paths, π holds

- $F\,\phi$ — there exists a future state in which ϕ holds
- $F^{\square}\,\phi$ — there exists a future state in which ϕ holds and all transitions until that state are must transitions
- $F\,\{\psi\}\,\phi$ — there exists a future state, reached by an action satisfying ψ, in which ϕ holds
- $F^{\square}\,\{\psi\}\,\phi$ — there exists a future state, reached by an action satisfying ψ, in which ϕ holds and all transitions until that state are must transitions

- $G\,\phi \equiv \neg F\,\neg\phi$ — the path is a full path on which ϕ holds in all states
- $AG\,\phi \equiv \neg EF\,\neg\phi$ — in all states on all paths, ϕ holds

Informal semantics of v-ACTL (cont.)

A full path is a path that cannot be extended further ($q \cdots$ or $q \not\rightarrow$)

$E\,\pi$	there exists a full path on which π holds
$A\,\pi$	on all possible full paths, π holds
$F\,\phi$	there exists a future state in which ϕ holds
$F^{\Box}\,\phi$	there exists a future state in which ϕ holds and all transitions until that state are must transitions
$F\,\{\psi\}\,\phi$	there exists a future state, reached by an action satisfying ψ, in which ϕ holds
$F^{\Box}\,\{\psi\}\,\phi$	there exists a future state, reached by an action satisfying ψ, in which ϕ holds and all transitions until that state are must transitions

$G\,\phi \equiv \neg F\,\neg\phi$ the path is a full path on which ϕ holds in all states

$AG\,\phi \equiv \neg EF\,\neg\phi$ in all states on all paths, ϕ holds

Semantics of v-ACTL over MTS $(Q, A, \overline{q}, \delta^{\Diamond}, \delta^{\Box})$

Let $q \in Q$ and σ a full path (from q) with ith state $\sigma(i)$ and ith action $\sigma\{i\}$

$$
\begin{array}{lcl}
q & \models & \neg\,\phi \text{ iff } q \not\models \phi \\
q & \models & \phi \wedge \phi' \text{ iff } q \models \phi \text{ and } q \models \phi' \\
q & \models & [\psi]\,\phi \text{ iff } \forall\, q' \in Q \text{ s.t. } q \xrightarrow{a(\cancel{e})}_{\Diamond} q' \text{ and } a(\cancel{e}) \models \psi, \text{ we have } q' \models \phi \\
q & \models & [\psi]^{\Box}\,\phi \text{ iff } \forall\, q' \in Q \text{ s.t. } q \xrightarrow{a(\cancel{e})}_{\Box} q' \text{ and } a(\cancel{e}) \models \psi, \text{ we have } q' \models \phi \\
q & \models & E\,\pi \text{ iff } \exists \text{ full path } \sigma' \text{ from } q \colon \sigma' \models \pi \\
q & \models & A\,\pi \text{ iff } \forall \text{ full path } \sigma' \text{ from } q \colon \sigma' \models \pi \\
q & \models & \mu\, Y.\phi(Y) \text{ iff } \bigvee_{i \geq 0} \phi^{i}(\mathit{false}) \\
q & \models & \nu\, Y.\phi(Y) \text{ iff } \bigwedge_{i \geq 0} \phi^{i}(\mathit{true}) \\
q & \models & F\,\phi \text{ iff } \exists\, j \geq 1 \colon \sigma(j) \models \phi \\
q & \models & F^{\Box}\,\phi \text{ iff } \exists\, j \geq 1 \colon \sigma(j) \models \phi \text{ and } \forall\, 1 \leq i < j \colon (\sigma(i), \sigma\{i\}, \sigma(i+1)) \in \delta^{\Box} \\
q & \models & F\,\{\psi\}\,\phi \text{ iff } \exists\, j \geq 1 \colon \sigma\{j\} \models \psi \text{ and } \sigma(j+1) \models \phi \\
q & \models & F^{\Box}\,\{\psi\}\,\phi \text{ iff } \exists\, j \geq 1 \colon \sigma\{j\} \models \psi \text{ and } \sigma(j+1) \models \phi, \text{ and } \forall\, 1 \leq i \leq j \colon \\
 & & (\sigma(i), \sigma\{i\}, \sigma(i+1)) \in \delta^{\Box}
\end{array}
$$

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

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

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

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

v-ACTL^{-}:

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

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

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

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

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

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

v-ACTL^{-}:

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

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

Live states use SPL-specific information

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

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

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 $\{a, b\}$

Live states use SPL-specific information

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

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

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 $\{a, b\}$

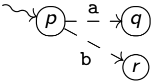
Live states use SPL-specific information

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

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

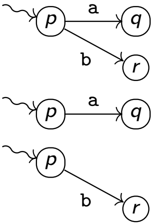
Live action sets define live states (not occur as final in any product)

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 $\{a, b\}$

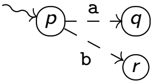
Live states use SPL-specific information

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

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

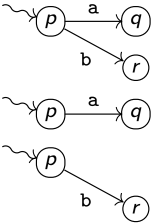
Live action sets define live states (not occur as final in any product)

MTS



Assume

a OR b



LTS

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

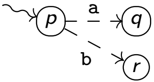
$\Rightarrow$ p is a live state, since a OR b gives rise to a live action set $\{a, b\}$

Live states use SPL-specific information

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

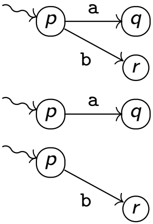
Recall: all (reachable) must transitions are preserved ($\langle\,\rangle^{\Box}$, $[\,]^{\Box}$, $F^{\Box}$)

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 $\{a, b\}$

Recall VMC: Variability Model Checker

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

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

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

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

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

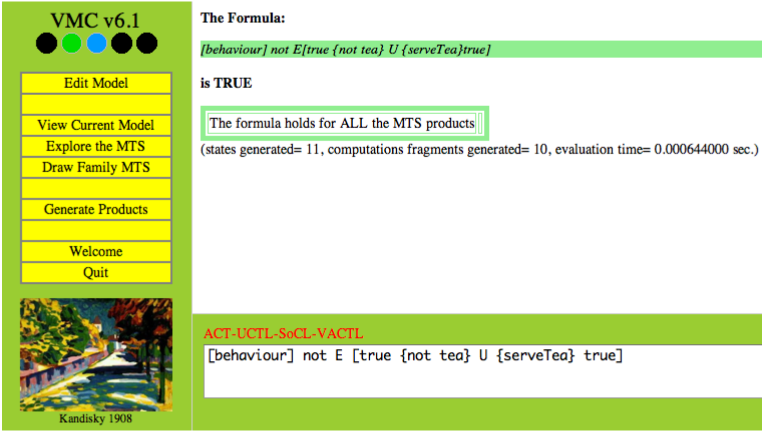
VMC accepts as input a specification in (value-passing) modal process algebra, possibly with additional variability constraints

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

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

Vending Machine: family-based verification

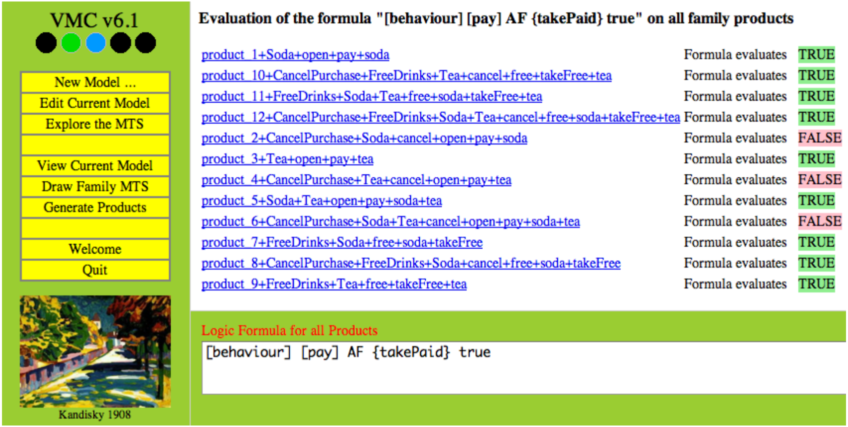
VMC notifies whenever preservation of a verification result is applicable



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

Vending Machine: product-based verification

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

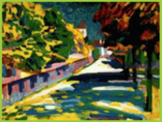


Whenever pay occurs, eventually takePaid occurs

Specification of the family of coffee machines in VMC

VMC v6.2

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



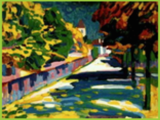
Kandisky 1908

```
T1 = euro(may).T2 + dollar(may).T2
T2 = sugar.T3 + no_sugar.T4
T3 = tea(may).T5 + coffee(may).T6 + cappuccino(may).T7
T4 = cappuccino(may).T8 + coffee(may).T9 + tea(may).T10
T5 = pour_sugar.T10
T6 = pour_sugar.T9
T7 = pour_sugar.T8
T8 = pour_milk.T9
T9 = pour_espresso(may).T11 + pour_regular(may).T11
T10 = pour_tea.T11
T11 = take_cup.T1

net SYS = T1

Constraints {
  euro ALT dollar
  cappuccino OR coffee OR tea
  dollar EXC tea
  cappuccino REQ coffee
  coffee REQ (pour_espresso ALT pour_regular)
}
```

Products of the family of coffee machines derived by VMC

VMC v6.2

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

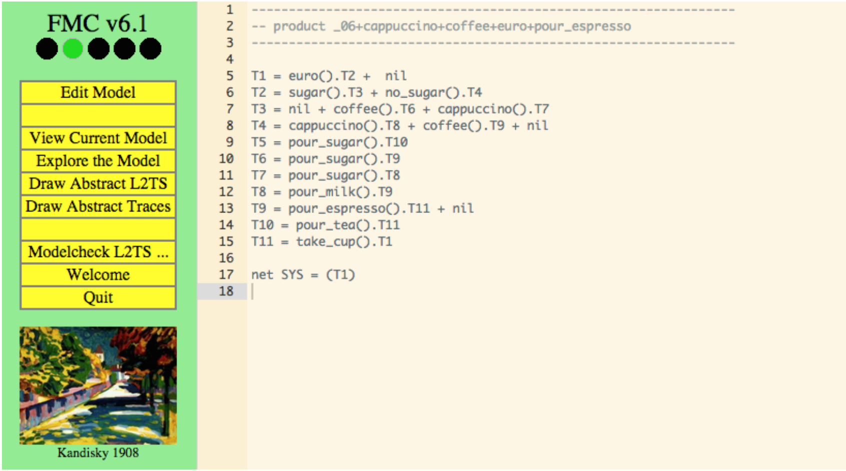


Kandisky 1908

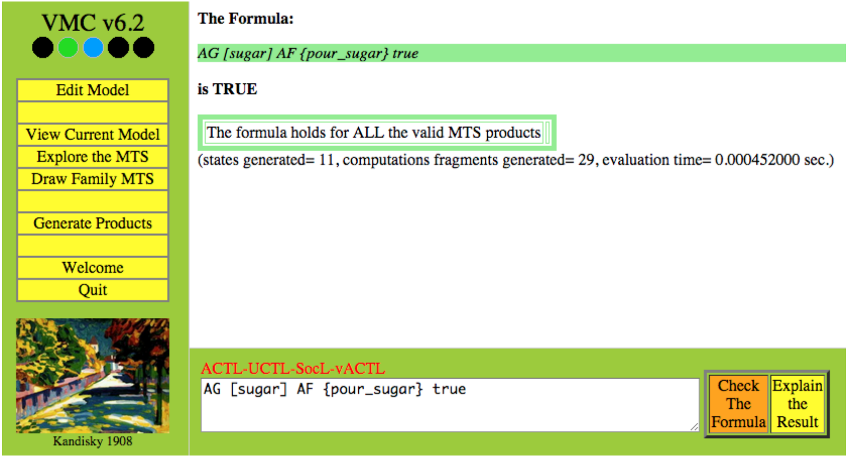
Valid Products of the Family

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

A product of the family of coffee machines derived by VMC

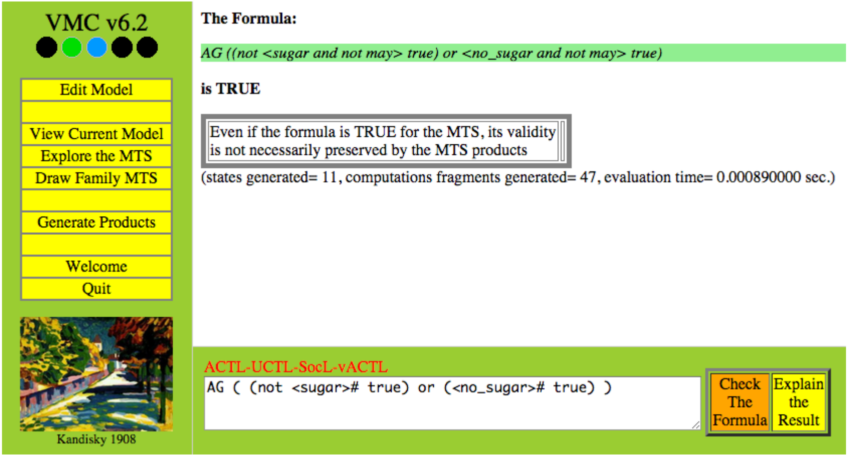


A formula verified by VMC over the family of coffee machines



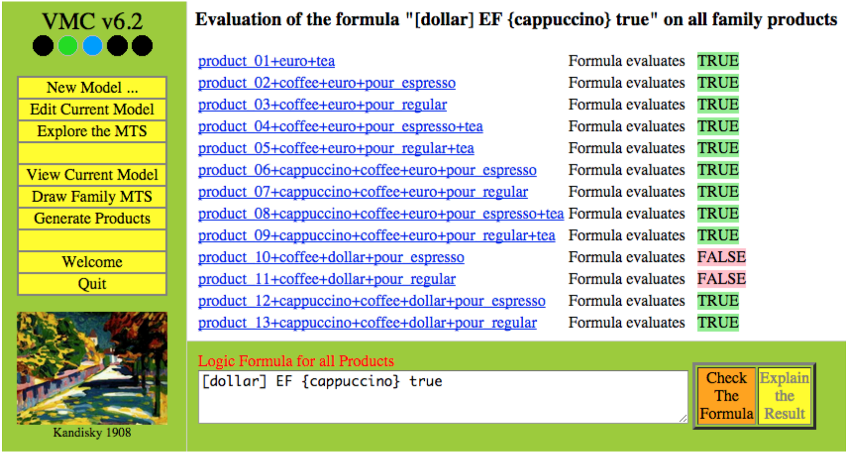
It is always the case that whenever sugar is chosen, eventually sugar is poured

A formula verified by VMC over the family of coffee machines



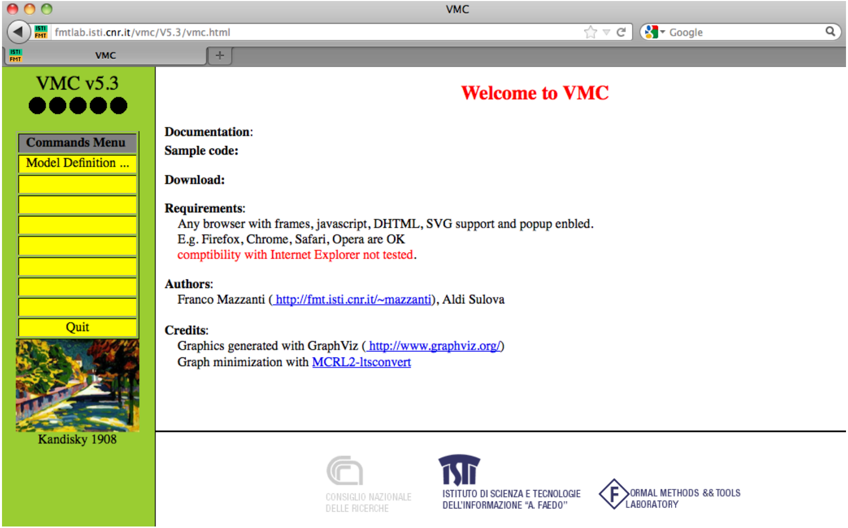
It is always the case that whenever sugar can be chosen, also no sugar can be chosen

A formula verified by VMC over all products of the family

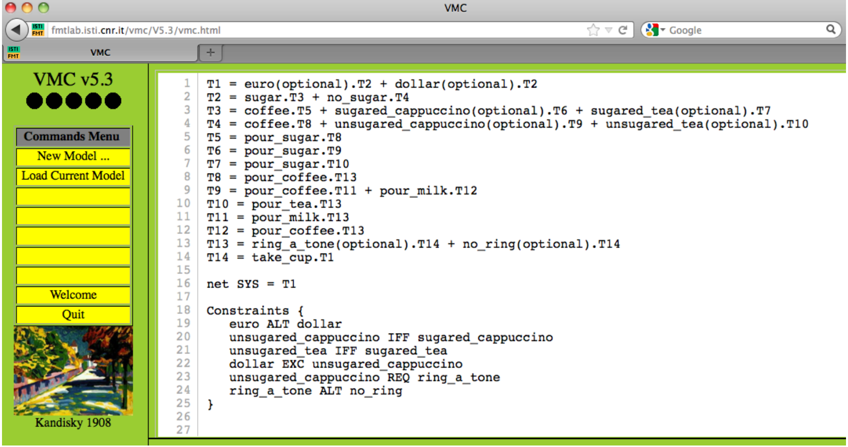


Upon the insertion of a dollar,
it might be the case that eventually a cappuccino can be chosen

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

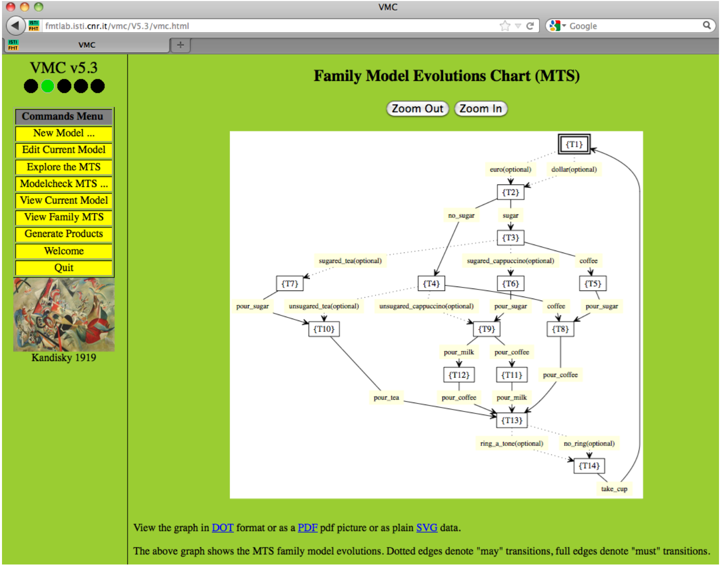


Family of coffee machines specified in VMC

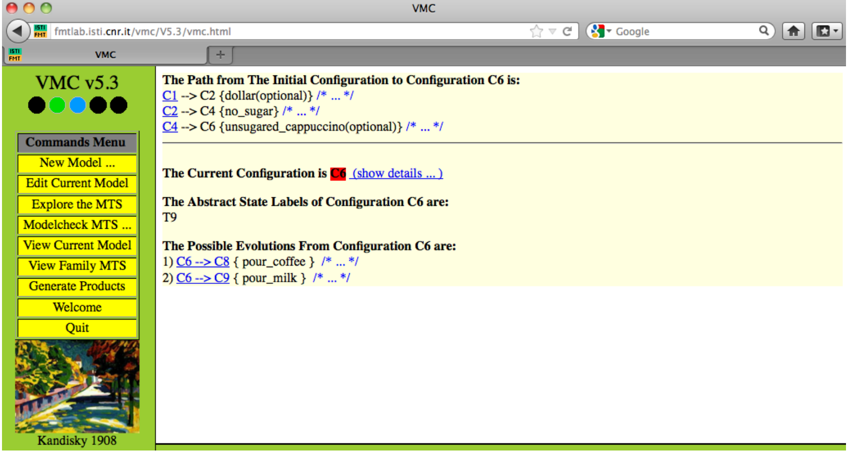


Permitted variability constraints ALTernative, EXCludes, REQuires, and IFF (shorthand for bilateral REQ) hide logic formalization from user

Family/MTS of coffee machines visualized by VMC

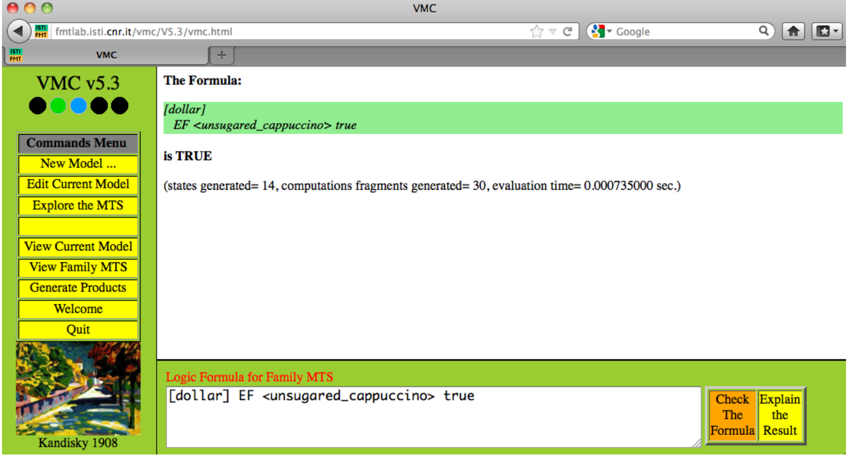


Family/MTS of coffee machines explored in VMC



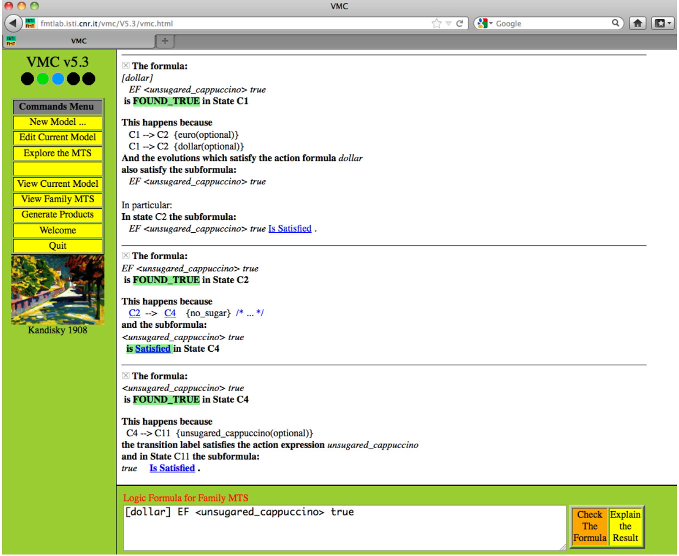
MTS model of coffee machine family actually permits a user to buy a cappuccino with a dollar, something which is forbidden for its products by variability constraint `dollar EXC unsugared_cappuccino`

Outcome of a property verified over a family with VMC

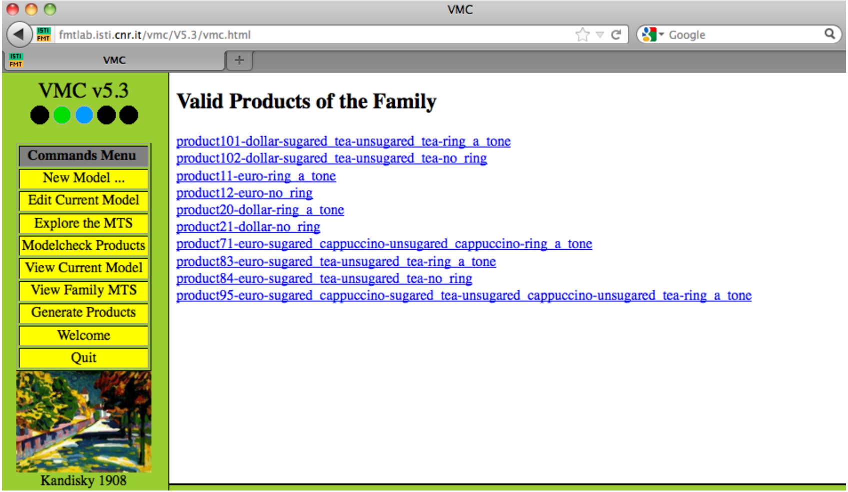


The formula expresses that every path through the MTS starting with a dollar insertion, eventually leads to an unsugared cappuccino

Outcome of a property explained by VMC



Products of family of coffee machines derived by VMC



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

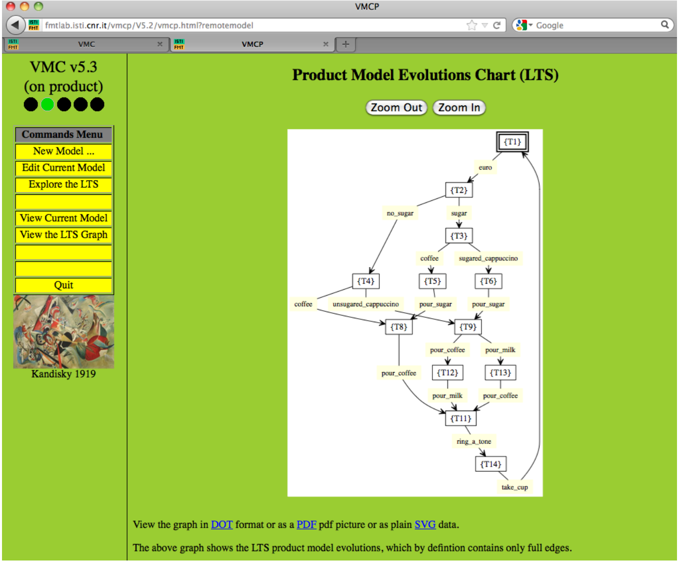
Outcomes of a property verified over products with VMC

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

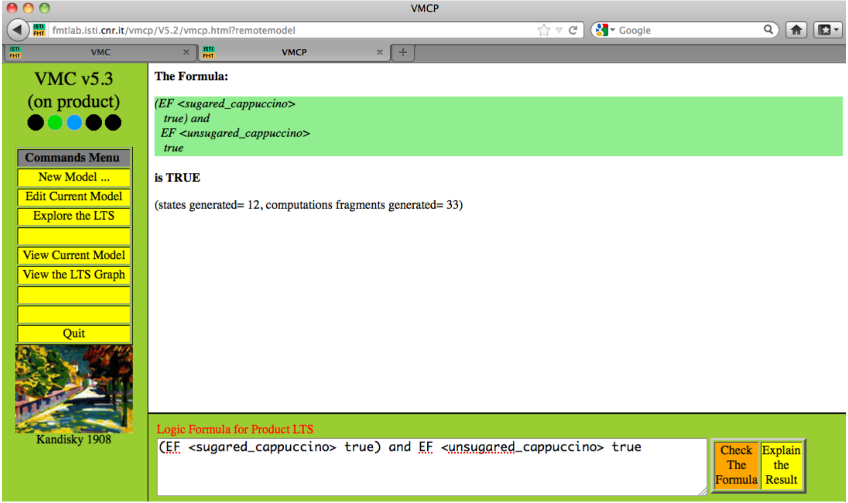
Specification of one of the products derived by VMC

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

Product/LTS on previous slide visualized by VMC

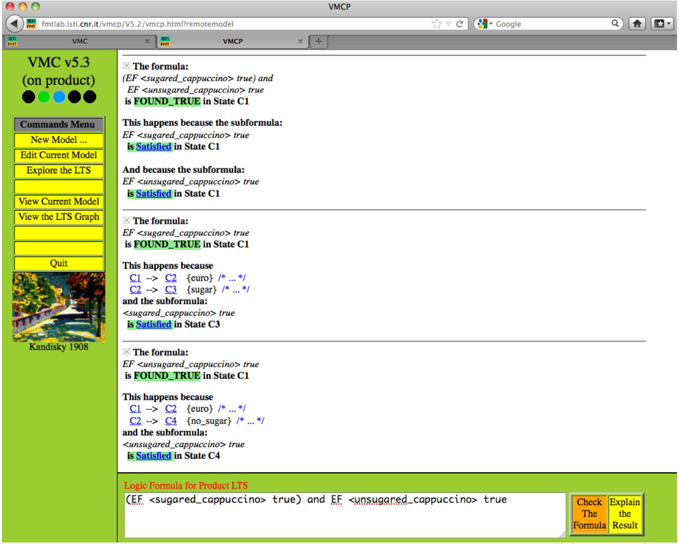


Outcome of a property verified over a product with VMC



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

Outcome property on previous slide explained by VMC



Non-trivial for branching-time temporal logics!

A value-passing modal process algebra

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

Processes are built from terms and actions according to the syntax

$$\begin{array}{lcl} N & ::= & [P] \\ P & ::= & K(e) \mid P\,/L/\,P \end{array}$$

$[P]$ denotes a closed system, i.e. it cannot evolve on input actions $a(?v)$

$K(e)$ is a process identifier from set of process definitions of form $K(v) \stackrel{\text{def}}{=} T$

$$\begin{array}{lcl} T & ::= & nil \mid K(e) \mid A.T \mid T + T \mid [e \bowtie e]\, T \\ A & ::= & a(e) \mid a(may, e) \mid a(?v) \mid a(may, ?v) \\ e & ::= & v \mid \textbf{int} \mid e \pm e \end{array}$$

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

Processes

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

We distinguish must actions $a \in \delta^{\Box}$ and may but not must actions $a(may) \in \delta^{\Diamond} \setminus \delta^{\Box}$ (each action type is treated differently in the SOS semantics)

Semantics in SOS style over MTS

$$(\text{sys})\ \frac{P \xrightarrow{a(e)} P'}{[P] \xrightarrow{a(e)} [P']} \qquad (\text{act}_{\Box})\ \frac{}{\alpha.P \xrightarrow{\alpha} P}\ {\scriptstyle \alpha \in \{a(e), a(?v)\}}$$

$$(\text{or}_{\Box})\ \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}\ {\scriptstyle \alpha \in \{a(e), a(?v)\}} \qquad (\text{int}_{\Box})\ \frac{P \xrightarrow{\ell} P'}{P\,/L/\,Q \xrightarrow{\ell} P'\,/L/\,Q}\ {\scriptstyle \ell \notin L}$$

$$(\text{par}_{\Box})\ \frac{P \xrightarrow{a(e_1)} P' \quad Q \xrightarrow{a(e_2)} Q'}{P\,/L/\,Q \xrightarrow{a} P'\,/L/\,Q'}\ {\scriptstyle a \in L, e_1 = e_2} \qquad (\text{par}_{\Box})\ \frac{P \xrightarrow{a(?v)} P' \quad Q \xrightarrow{a(e)} Q'}{P\,/L/\,Q \xrightarrow{a} P'[^{e}/_{v}]\,/L/\,Q'}\ {\scriptstyle a \in L}$$

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

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

Bike-sharing systems (BSS)

- Simple concept: a user arrives at a docking station, pays for a bike, uses it for a while and returns it to a station
- Multiple benefits: reduction of vehicular traffic (congestion), pollution, energy consumption, etc.

- Docking stations distributed over a city, typically close to other public transportation hubs (e.g. subway and tram stations)
- (Subscribed) users may rent an available bike and drop it off at any station in the city
- To improve the efficiency and the user satisfaction of BSS, the load between the different stations may be balanced
  - incentive schemes (rewards) to change the behavior of users
  - efficient (dynamic) redistribution of bikes between stations

Bike-sharing systems (BSS)

- Simple concept: a user arrives at a docking station, pays for a bike, uses it for a while and returns it to a station
- Multiple benefits: reduction of vehicular traffic (congestion), pollution, energy consumption, etc.

- Docking stations distributed over a city, typically close to other public transportation hubs (e.g. subway and tram stations)
- (Subscribed) users may rent an available bike and drop it off at any station in the city
- To improve the efficiency and the user satisfaction of BSS, the load between the different stations may be balanced
  - incentive schemes (rewards) to change the behavior of users
  - efficient (dynamic) redistribution of bikes between stations

History of BSS

DeMaio @ *Journal of Public Transportation*, 2009

- 1st generation free BSS introduced in Amsterdam (*witte fietsen*)
- 2nd generation born in Denmark, first large-scale BSS launched in Copenhagen (*Bycyklen*)
- 3rd generation technology-based BSS in > 500 cities worldwide (*Vélib'* in Paris: over $20,000$ bikes and $1,800$ stations; largest in Hangzhou: $\pm 50,000$ bikes and $2,000$ stations, one every 100m)
- 4th generation BSS are already being developed, incl. movable and solar-powered stations, electric bikes and mobile (i)phone real-time availability applications (*Bycyklen*)

We collaborate with *PisaMo S.p.A.*, an in-house public mobility company of Pisa's administration that introduced the public BSS *CicloPi* in Pisa in 2013 (currently ± 250 bikes and 24 stations), and **Bici**ncittà S.r.l. di Torino (supplying and monitoring)

History of BSS

DeMaio @ *Journal of Public Transportation*, 2009

- 1st generation free BSS introduced in Amsterdam (*witte fietsen*)
- 2nd generation born in Denmark, first large-scale BSS launched in Copenhagen (*Bycyklen*)
- 3rd generation technology-based BSS in >500 cities worldwide (*Vélib'* in Paris: over $20,000$ bikes and $1,800$ stations; largest in Hangzhou: $\pm 50,000$ bikes and $2,000$ stations, one every 100m)
- 4th generation BSS are already being developed, incl. movable and solar-powered stations, electric bikes and mobile (i)phone real-time availability applications (*Bycyklen*)

We collaborate with *PisaMo S.p.A.*, an in-house public mobility company of Pisa's administration that introduced the public BSS *CicloPi* in Pisa in 2013 (currently ± 250 bikes and 24 stations), and **Bici**ncittà S.r.l. di Torino (supplying and monitoring)

History of BSS

DeMaio @ *Journal of Public Transportation*, 2009

- 1st generation free BSS introduced in Amsterdam (*witte fietsen*)
- 2nd generation born in Denmark, first large-scale BSS launched in Copenhagen (*Bycyklen*)
- 3rd generation technology-based BSS in > 500 cities worldwide (*Vélib'* in Paris: over $20,000$ bikes and $1,800$ stations; largest in Hangzhou: $\pm 50,000$ bikes and $2,000$ stations, one every 100m)
- 4th generation BSS are already being developed, incl. movable and solar-powered stations, electric bikes and mobile (i)phone real-time availability applications (*Bycyklen*)

We collaborate with *PisaMo S.p.A.*, an in-house public mobility company of Pisa's administration that introduced the public BSS *CicloPi* in Pisa in 2013 (currently ± 250 bikes and 24 stations), and **Bici**città S.r.l. di Torino (supplying and monitoring)

History of BSS

DeMaio @ *Journal of Public Transportation*, 2009

- 1st generation free BSS introduced in Amsterdam (*witte fietsen*)
- 2nd generation born in Denmark, first large-scale BSS launched in Copenhagen (*Bycyklen*)
- 3rd generation technology-based BSS in $>$ 500 cities worldwide (*Vélib'* in Paris: over $20,000$ bikes and $1,800$ stations; largest in Hangzhou: $\pm 50,000$ bikes and $2,000$ stations, one every 100m)
- 4th generation BSS are already being developed, incl. movable and solar-powered stations, electric bikes and mobile (i)phone real-time availability applications (*Bycyklen*)

We collaborate with *PisaMo S.p.A.*, an in-house public mobility company of Pisa's administration that introduced the public BSS *CicloPi* in Pisa in 2013 (currently ± 250 bikes and 24 stations), and **Bici**ncittà S.r.l. di Torino (supplying and monitoring)

History of BSS

DeMaio @ *Journal of Public Transportation*, 2009

- 1st generation free BSS introduced in Amsterdam (*witte fietsen*)
- 2nd generation born in Denmark, first large-scale BSS launched in Copenhagen (*Bycyklen*)
- 3rd generation technology-based BSS in $>$ 500 cities worldwide (*Vélib'* in Paris: over 20,000 bikes and 1,800 stations; largest in Hangzhou: $\pm$50,000 bikes and 2,000 stations, one every 100m)
- 4th generation BSS are already being developed, incl. movable and solar-powered stations, electric bikes and mobile (i)phone real-time availability applications (*Bycyklen*)

We collaborate with *PisaMo S.p.A.*, an in-house public mobility company of Pisa's administration that introduced the public BSS *CicloPi* in Pisa in 2013 (currently $\pm$250 bikes and 24 stations), and **Bici**ncittà S.r.l. di Torino (supplying and monitoring)

History of BSS

DeMaio @ *Journal of Public Transportation*, 2009

- 1st generation free BSS introduced in Amsterdam (*witte fietsen*)
- 2nd generation born in Denmark, first large-scale BSS launched in Copenhagen (*Bycyklen*)
- 3rd generation technology-based BSS in $>$ 500 cities worldwide (*Vélib'* in Paris: over $20,000$ bikes and $1,800$ stations; largest in Hangzhou: $\pm 50,000$ bikes and $2,000$ stations, one every 100m)
- 4th generation BSS are already being developed, incl. movable and solar-powered stations, electric bikes and mobile (i)phone real-time availability applications (*Bycyklen*)

We collaborate with *PisaMo S.p.A.*, an in-house public mobility company of Pisa's administration that introduced the public BSS *CicloPi* in Pisa in 2013 (currently $\pm$ 250 bikes and 24 stations), and **Bici**ncittà S.r.l. di Torino (supplying and monitoring)

Behavioral requirements of a BSS

We consider a BSS with N stations and a fleet of M bikes;
Each station i has a capacity K_i; Redistribution is optional

1. Users arrive at station i
2. If a user arrives at a station with no available bike, (s)he leaves
3. Otherwise, (s)he takes a bike and chooses station j to return it
4. If there are less than K_j bikes at station j when (s)he arrives, (s)he returns the bike and leaves
5. If the station is full she chooses another station k and goes there
6. Redistribution of bikes may be asked for and may possibly occur
7. The user rides like this again until (s)he can return the bike

Inspired by Fricker, Gast @ EURO Journal on Transportation and Logistics, 2014

Behavioral requirements of a BSS

We consider a BSS with N stations and a fleet of M bikes;
Each station i has a capacity K_i; Redistribution is optional

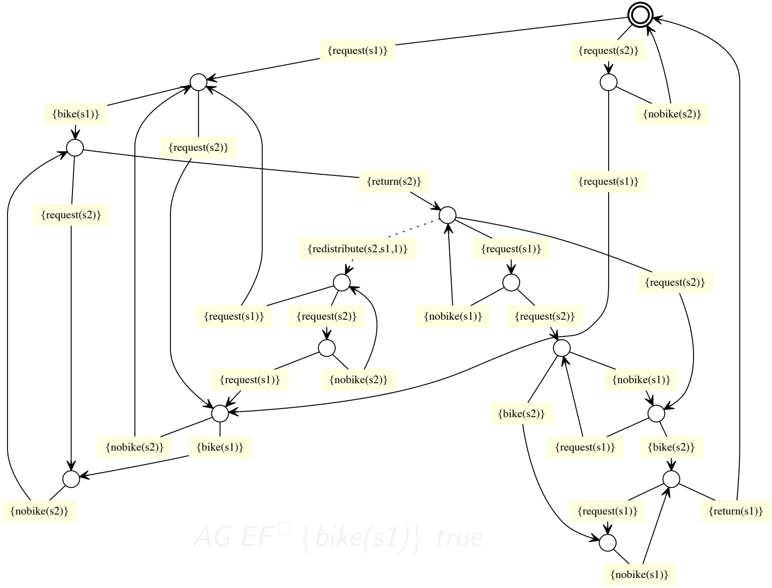
1. Users arrive at station i
2. If a user arrives at a station with no available bike, (s)he leaves
3. Otherwise, (s)he takes a bike and chooses station j to return it
4. If there are less than K_j bikes at station j when (s)he arrives, (s)he returns the bike and leaves
5. If the station is full she chooses another station k and goes there
6. Redistribution of bikes may be asked for and may possibly occur
7. The user rides like this again until (s)he can return the bike

Inspired by Fricker, Gast @ EURO Journal on Transportation and Logistics, 2014

Value-passing BSS specification

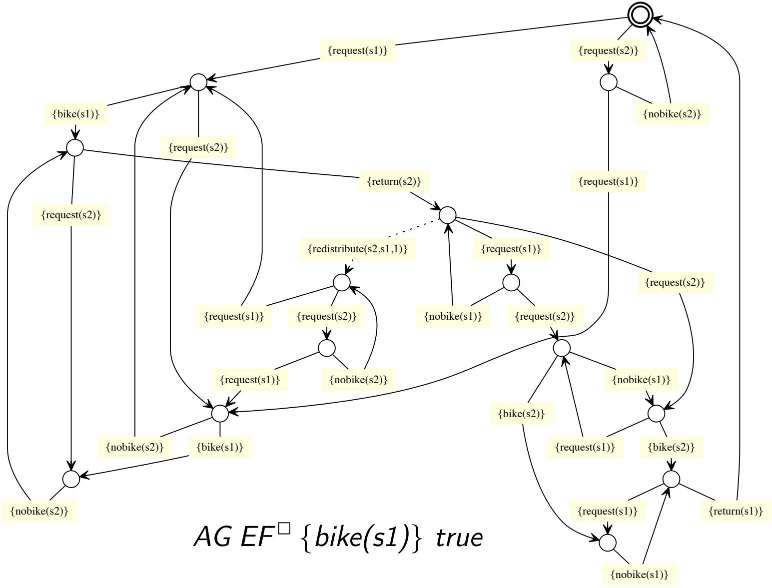
```
    Station(I,N,J,M) = request(I).
        ( [N=0] nobike(I).Station(I,N,J,M) +
          [N>0] bike(I).Station(I,N-1,J,M) ) +
      return(I).Station(I,N+1,J,M) +
      redistribute(may,?FROM,?TO,?K).
        ( [TO = I] Station(I,N+K,J,M) +
          [TO /= I] Station(I,N,J,M) ) +
      [N > M] redistribute(may,I,J,N-M).Station(I,M,J,M)

net STATIONS = Station(s1,1,s2,1) /redistribute/ Station(s2,0,s1,0)

    Users(I,J) = request(I).
        ( bike(I).return(J).Users(I,J) +
          nobike(I).Users(I,J) )

net USERS = Users(s1,s2) // Users(s2,s1)

net BSS = STATIONS /request,bike,nobike,return/ USERS
```

MTS with parameters and values



AG EF$^{\square}$ {bike(s1)} true

MTS with parameters and values



$AG\ EF^{\Box}\ \{bike(s1)\}\ true$

v-ACTL$^{\Box}$ formula: true $\forall$ products

Products generated by VMC
(not needed for verification)