

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

- 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-CTL: variability-aware, action-based CTL
  - Preservation of formulae in v-CTL<sup>□</sup>/v-CTL<sup>Live</sup><sup>□</sup>
- 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

Recall (atomic propositions used for verification purposes):

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

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

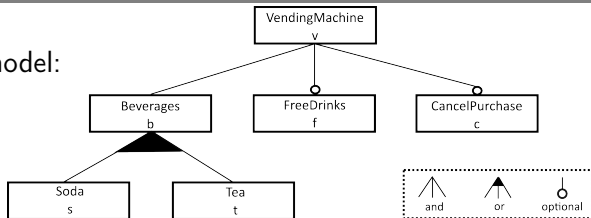
An FTS adds to this a feature model and feature expressions:

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

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

# FTS of example SPL: a vending machine

Feature model:

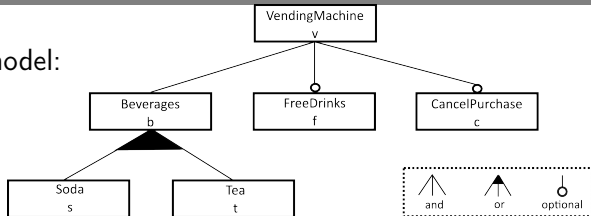


12 valid products

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

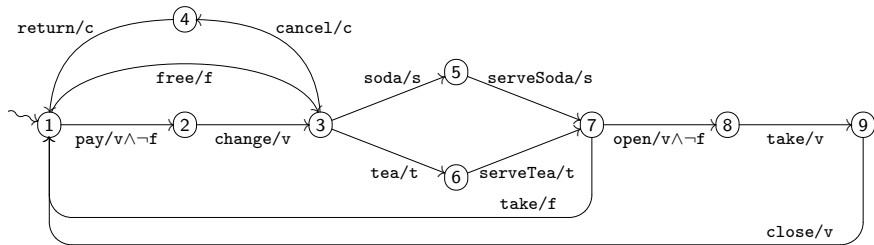
# FTS of example SPL: a vending machine

Feature model:



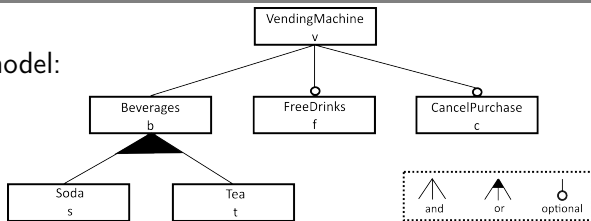
FTS of 12 valid products (LTS)

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

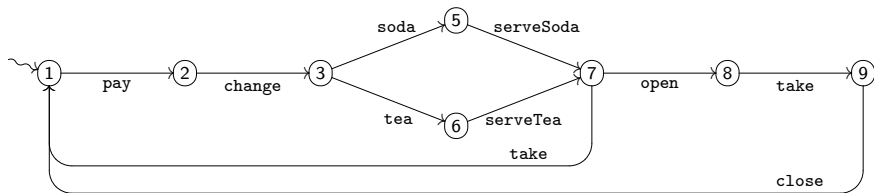


# FTS of example SPL: a vending machine

Feature model:

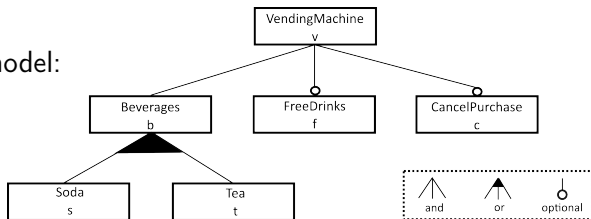


e.g.  $\{v, b, s, t\}$ ,  $\{v, b, s, c\}$

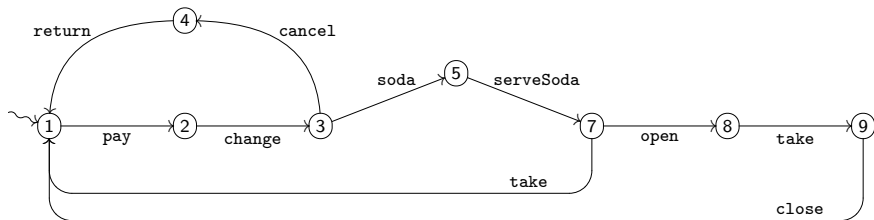


# FTS of example SPL: a vending machine

Feature model:



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



## Main ingredient: Modal Transition Systems (MTS)

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

Larsen, Thomsen © LICS'88

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

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

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

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

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

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

## MTS with...

Recall:

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

Next we define MTS with variability constraints:

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

An MTS has two distinct transition relations

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

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

## ... variability constraints (accepted by VMC)

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

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

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

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

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

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

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

## Derive products (implemented in VMC)

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

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

⇒ Each selection gives rise to a different variant

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

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

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

## Derive products (implemented in VMC)

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

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

⇒ Each selection gives rise to a different variant

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

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

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

# VMC: Variability Model Checker

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

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

1. The actual set of all valid product behaviour can explicitly be generated and the resulting LTS can all be verified against one and the same logic property (expressed in Action-based CTL)  
De Nicola, Vaandrager © J. ACM, 1995
2. A logic property (expressed in **variability-aware** ACTL, 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)
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

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

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

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

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

# VMC: Variability Model Checker

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

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

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

De Nicola, Vaandrager © J. ACM, 1995

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

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

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

# Crux: from feature constraints to action constraints

From feature model: A requires C



From FTS to MTS (naive): a REQ c



From FTS to MTS (solution):

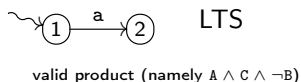
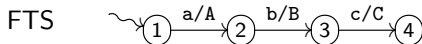
1. new action  $\forall$  feature (to handle more complex feature expressions)
2. dummy transition  $\forall$  action (to verify constraints, ignored when model checking)



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

# Crux: from feature constraints to action constraints

From feature model: A requires C



From FTS to MTS (naive): a REQ c



From FTS to MTS (solution):

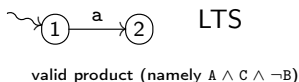
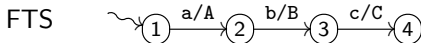
1. new action  $\forall$  feature (to handle more complex feature expressions)
2. dummy transition  $\forall$  action (to verify constraints, ignored when model checking)



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

# Crux: from feature constraints to action constraints

From feature model: A requires C



From FTS to MTS (naive): a REQ c



From FTS to MTS (solution):

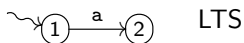
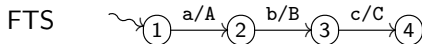
1. new action  $\forall$  feature (to handle more complex feature expressions)
2. dummy transition  $\forall$  action (to verify constraints, ignored when model checking)



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

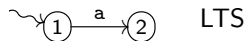
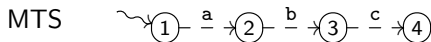
# Crux: from feature constraints to action constraints

From feature model: A requires C



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

From FTS to MTS (naive): a REQ c



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

From FTS to MTS (solution):

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

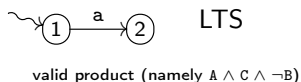
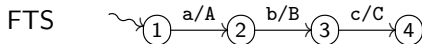


valid product (satisfies a REQ c)

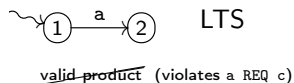
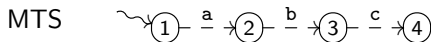
! Consistency guarantees that whenever a c-labelled may transition from the initial state is preserved in this LTS, then also any other **reachable** c-labelled may transition must be preserved

# Cruz: from feature constraints to action constraints

From feature model: A requires C



From FTS to MTS (naive): a REQ c



From FTS to MTS (solution):

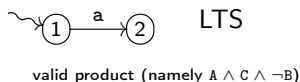
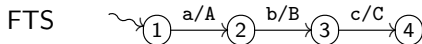
1. new action  $\forall$  feature (to handle more complex feature expressions)
2. dummy transition  $\forall$  action (to verify constraints, ignored when model checking)



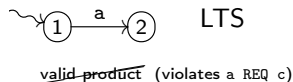
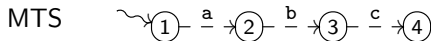
! Consistency guarantees that whenever a c-labelled may transition from the initial state is preserved in this LTS, then also any other *reachable* c-labelled may transition must be preserved

# Crux: from feature constraints to action constraints

From feature model: A requires C



From FTS to MTS (naive): a REQ c



From FTS to MTS (solution):

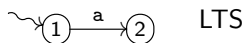
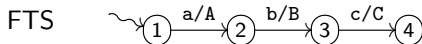
1. new action  $\forall$  feature (to handle more complex feature expressions)
2. dummy transition  $\forall$  action (to verify constraints, **ignored when model checking**)



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

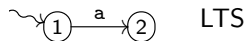
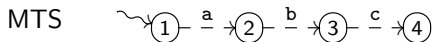
# Crux: from feature constraints to action constraints

From feature model: A requires C



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

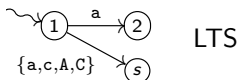
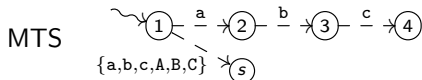
From FTS to MTS (naive): a REQ c



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

From FTS to MTS (solution):

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

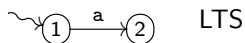
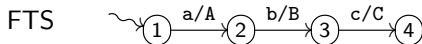


valid product (satisfies a REQ c)

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

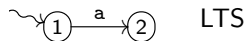
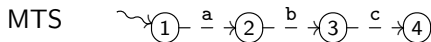
# Crux: from feature constraints to action constraints

From feature model: A requires C



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

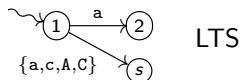
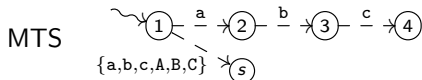
From FTS to MTS (naive): a REQ c



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

From FTS to MTS (solution):

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

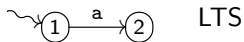
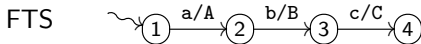


valid product (satisfies a REQ c)

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

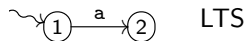
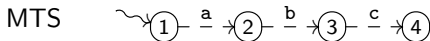
# Cruz: from feature constraints to action constraints

From feature model: A requires C



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

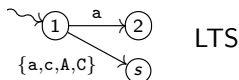
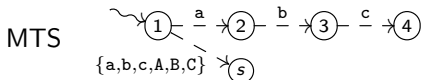
From FTS to MTS (naive): a REQ c



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

From FTS to MTS (solution):

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



valid product (satisfies a REQ c)

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

# Model Transformation (1/4)

---

Step 1: definition of valid products in terms of features

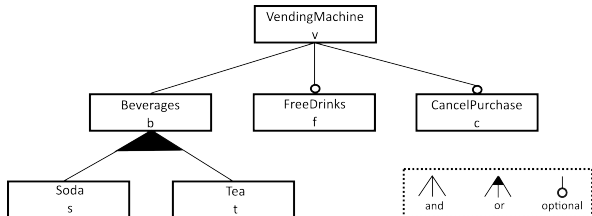
Translate feature model in a set of variability constraints on features

# Model Transformation (1/4)

## Step 1: definition of valid products in terms of features

Translate feature model in a set of variability constraints on features

Constraints {  
Soda OR Tea  
}



## Model Transformation (2/4)

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

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

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

## Model Transformation (2/4)

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

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

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

## Model Transformation (3/4)

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

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

```
Behaviour = behaviour.T1
```

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

```
T2 = change.T3
```

```
...
```

```
T9 = close.T1
```

```
Signature = signature.(
```

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

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

```
)
```

```
VMCmodel = Behaviour + Signature
```

## 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/ $\phi$ '
2. Add dummy transition for each 'may' action / non-mandatory feature
3. Create a new initial process with special action behaviour leading to FTS encoding, whereas signature leads to dummy transitions

```
Behaviour = behaviour.T1
```

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

```
T2 = change.T3
```

```
...
```

```
T9 = close.T1
```

```
Signature = signature.(
```

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

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

```
)
```

```
VMCmodel = Behaviour + Signature
```

## 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/ $\phi$ '
2. Add dummy transition for each 'may' action / non-mandatory feature
3. Create a new initial process with special action behaviour leading to FTS encoding, whereas signature leads to dummy transitions

```
Behaviour = behaviour.T1
```

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

```
T2 = change.T3
```

```
...
```

```
T9 = close.T1
```

```
Signature = signature.(
```

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

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

```
)
```

```
VMCmodel = Behaviour + Signature
```

## 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/ $\phi$ '
2. Add dummy transition for each 'may' action / non-mandatory feature
3. Create a new initial process with special action behaviour leading to FTS encoding, whereas signature leads to dummy transitions

```
Behaviour = behaviour.T1
```

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

```
T2 = change.T3
```

```
...
```

```
T9 = close.T1
```

```
Signature = signature.(
```

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

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

```
)
```

```
VMCmodel = Behaviour + Signature
```

## 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/ $\phi$ '
2. Add dummy transition for each 'may' action / non-mandatory feature
3. Create a new initial process with special action behaviour leading to FTS encoding, whereas signature leads to dummy transitions

```
Behaviour = behaviour.T1
```

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

```
T2 = change.T3
```

```
...
```

```
T9 = close.T1
```

```
Signature = signature.(
```

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

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

```
)
```

```
VMCmodel = Behaviour + Signature
```

# Model Transformation (4/4)

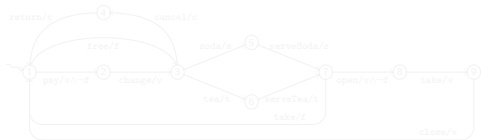
## Step 4: definition of live action sets and introduction of must transitions

We perform two optimisations for model-checking purposes

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

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

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



## Model Transformation (4/4)

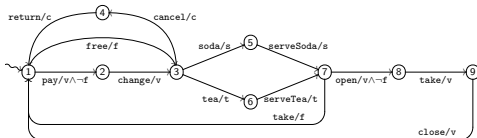
### Step 4: definition of live action sets and introduction of must transitions

We perform two optimisations for model-checking purposes

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

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

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



## Soundness of model transformation

Given FTS  $S$  and MTS  $S'$

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

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

### Theorem

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

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

### Proof.

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

## Recall: Action-based CTL

Syntax over **action formulas** (boolean compositions of actions, denoted by  $\psi$ ), **state formulas** ( $\phi$ ) and **path formulas** ( $\pi$ )

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

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

$\mu$  and  $\nu$ : **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  $\psi$ ), **state formulas** ( $\phi$ ) and **path formulas** ( $\pi$ )

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

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

$\mu$  and  $\nu$ : 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  $\psi$ ), **state formulas** ( $\phi$ ) and **path formulas** ( $\pi$ )

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

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

$\mu$  and  $\nu$ : **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  $\psi$ , state formulae  $\phi$ , path formulae  $\pi$

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

ter Beek, Gnesi, Mazzanti © PSI'14

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

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

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

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

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

# Informal semantics of v-ACTL

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

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

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

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

$\langle \langle \psi \rangle^{\square}$  and  $[\psi]^{\square}$  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  $\psi$ ,  $\phi$  holds

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

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

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

$\langle \langle \psi \rangle^{\square}$  and  $[\psi]^{\square}$  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  $\psi$ ,  $\phi$  holds

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

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

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

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

## Informal semantics of v-CTL (cont.)

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

$E \pi$  there exists a full path on which  $\pi$  holds

$A \pi$  on all possible full paths,  $\pi$  holds

$F \phi$  there exists a future state in which  $\phi$  holds

$F^{\square} \phi$  there exists a future state in which  $\phi$  holds and all transitions until that state are **must** transitions

$F\{\psi\} \phi$  there exists a future state, reached by an action satisfying  $\psi$ , in which  $\phi$  holds

$F^{\square}\{\psi\} \phi$  there exists a future state, reached by an action satisfying  $\psi$ , in which  $\phi$  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  $\phi$  holds in all states

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

## Informal semantics of v-ACTL (cont.)

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

$E \pi$  there exists a full path on which  $\pi$  holds

$A \pi$  on all possible full paths,  $\pi$  holds

$F \phi$  there exists a future state in which  $\phi$  holds

$F^\square \phi$  there exists a future state in which  $\phi$  holds and all transitions until that state are **must** transitions

$F \{ \psi \} \phi$  there exists a future state, reached by an action satisfying  $\psi$ , in which  $\phi$  holds

$F^\square \{ \psi \} \phi$  there exists a future state, reached by an action satisfying  $\psi$ , in which  $\phi$  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  $\phi$  holds in all states

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

## Informal semantics of v-ACTL (cont.)

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

$E \pi$  there exists a full path on which  $\pi$  holds

$A \pi$  on all possible full paths,  $\pi$  holds

$F \phi$  there exists a future state in which  $\phi$  holds

$F^\square \phi$  there exists a future state in which  $\phi$  holds and all transitions until that state are **must** transitions

$F \{ \psi \} \phi$  there exists a future state, reached by an action satisfying  $\psi$ , in which  $\phi$  holds

$F^\square \{ \psi \} \phi$  there exists a future state, reached by an action satisfying  $\psi$ , in which  $\phi$  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  $\phi$  holds in all states

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

# Semantics of $\nu$ -ACTL over MTS $(Q, A, \bar{q}, \delta^\diamond, \delta^\square)$

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

$$q \models \neg\phi \text{ iff } q \not\models \phi$$

$$q \models \phi \wedge \phi' \text{ iff } q \models \phi \text{ and } q \models \phi'$$

$$q \models [\psi]\phi \text{ iff } \forall q' \in Q \text{ s.t. } q \xrightarrow{a(e^X)}_\diamond q' \text{ and } a(e^X) \models \psi, \text{ we have } q' \models \phi$$

$$q \models [\psi]^\square\phi \text{ iff } \forall q' \in Q \text{ s.t. } q \xrightarrow{a(e^X)}_\square q' \text{ and } a(e^X) \models \psi, \text{ we have } q' \models \phi$$

$$q \models E\pi \text{ iff } \exists \text{ full path } \sigma' \text{ from } q: \sigma' \models \pi$$

$$q \models A\pi \text{ iff } \forall \text{ full path } \sigma' \text{ from } q: \sigma' \models \pi$$

$$q \models \mu Y.\phi(Y) \text{ iff } \bigvee_{i \geq 0} \phi^i(\text{false})$$

$$q \models \nu Y.\phi(Y) \text{ iff } \bigwedge_{i \geq 0} \phi^i(\text{true})$$

$$q \models F\phi \text{ iff } \exists j \geq 1: \sigma(j) \models \phi$$

$$q \models F^\square\phi \text{ iff } \exists j \geq 1: \sigma(j) \models \phi \text{ and } \forall 1 \leq i < j: (\sigma(i), \sigma\{i\}, \sigma(i+1)) \in \delta^\square$$

$$q \models F\{\psi\}\phi \text{ iff } \exists j \geq 1: \sigma\{j\} \models \psi \text{ and } \sigma(j+1) \models \phi$$

$$q \models F^\square\{\psi\}\phi \text{ iff } \exists j \geq 1: \sigma\{j\} \models \psi \text{ and } \sigma(j+1) \models \phi, \text{ and } \forall 1 \leq i \leq j: (\sigma(i), \sigma\{i\}, \sigma(i+1)) \in \delta^\square$$

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

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

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

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

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

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

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

# Preservation of formulae in v-ACTL<sup>□</sup>/v-ACTLive<sup>□</sup>

v-ACTL<sup>□</sup>/v-ACTLive<sup>□</sup>:

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

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

v-ACTL<sup>-</sup>:

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

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

## Live states use SPL-specific information

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

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

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



Assume  
a OR b



LTS

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

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

## Live states use SPL-specific information

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

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

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

MTS



Assume  
a OR b



LTS



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

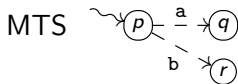
$\Rightarrow p$  is a live state, since a OR b gives rise to a live action set  $\{a, b\}$

## Live states use SPL-specific information

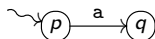
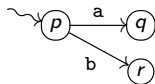
$$S \models \phi \Rightarrow S_p \models \phi \quad \forall \text{ product LTS } S_p \text{ of MTS } S$$

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

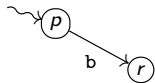
**Live action sets** define **live states** (not occur as final in any product)



Assume  
a OR b



LTS



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

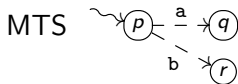
$\Rightarrow p$  is a live state, since a OR b gives rise to a live action set  $\{a, b\}$

## Live states use SPL-specific information

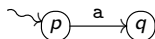
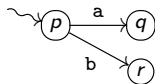
$$S \models \phi \Rightarrow S_p \models \phi \quad \forall \text{product LTS } S_p \text{ of MTS } S$$

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

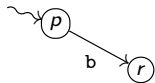
**Live action sets** define **live states** (not occur as final in any product)



Assume  
a OR b



LTS



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

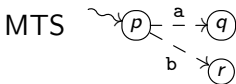
$\Rightarrow p$  is a live state, since a OR b gives rise to a live action set  $\{a, b\}$

## Live states use SPL-specific information

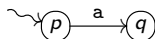
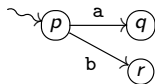
$$S \models \phi \Rightarrow S_p \models \phi \quad \forall \text{product LTS } S_p \text{ of MTS } S$$

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

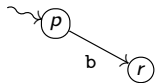
**Live action sets** define **live states** (not occur as final in any product)



Assume  
a OR b



LTS



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

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

# Recall VMC: Variability Model Checker

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

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

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

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

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

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

- interactively explore the model (MTS)
- derive and explore (all) the model's valid variants (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

**VMC v6.1**

● ● ● ● ●

Edit Model

View Current Model


Explore the MTS

Draw Family MTS

Generate Products

Welcome

Quit



Kandisky 1908

**The Formula:**

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

is **TRUE**

The formula holds for ALL the MTS products

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

**ACT-UCTL-SoCL-VACTL**

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

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

# Vending Machine: product-based verification

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

**VMC v6.1**

● ● ● ● ●

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



Kandisky 1908

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

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

**Logic Formula for all Products**

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

*Whenever pay occurs, eventually takePaid occurs*

# Specification of the family of coffee machines in VMC

VMC v6.2



Edit Model

View Current Model

Explore the MTS

Draw Family MTS

Modelcheck MTS ...

Generate Products

Welcome

Quit



Kandisky 1908

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

# Products of the family of coffee machines derived by VMC

VMC v6.2



Edit Model

View Current Model

Explore the MTS

Draw Family MTS

Modelcheck MTS ...

Generate Products

Modelcheck Products

Welcome

Quit



Kandisky 1908

## Valid Products of the Family

[product\\_01+euro+tea](#)

[product\\_02+coffee+euro+pour\\_espresso](#)

[product\\_03+coffee+euro+pour\\_regular](#)

[product\\_04+coffee+euro+pour\\_espresso+tea](#)

[product\\_05+coffee+euro+pour\\_regular+tea](#)

[product\\_06+cappuccino+coffee+euro+pour\\_espresso](#)

[product\\_07+cappuccino+coffee+euro+pour\\_regular](#)

[product\\_08+cappuccino+coffee+euro+pour\\_espresso+tea](#)

[product\\_09+cappuccino+coffee+euro+pour\\_regular+tea](#)

[product\\_10+coffee+dollar+pour\\_espresso](#)

[product\\_11+coffee+dollar+pour\\_regular](#)

[product\\_12+cappuccino+coffee+dollar+pour\\_espresso](#)

[product\\_13+cappuccino+coffee+dollar+pour\\_regular](#)

# A product of the family of coffee machines derived by VMC

FMC v6.1



Edit Model

View Current Model

Explore the Model

Draw Abstract L2TS

Draw Abstract Traces

Modelcheck L2TS ...

Welcome

Quit




Kandisky 1908

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

# A formula verified by VMC over the family of coffee machines

**VMC v6.2**  
● ● ● ● ●

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



Kandisky 1908

**The Formula:**  
 $AG [sugar] AF \{pour\_sugar\} true$

is TRUE

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

**ACTL-UCTL-SocL-vACTL**  
 $AG [sugar] AF \{pour\_sugar\} true$

Check The Formula Explain the Result

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

# A formula verified by VMC over the family of coffee machines

**VMC v6.2**

● ● ● ● ●

Edit Model

View Current Model


Explore the MTS

Draw Family MTS

Generate Products

Welcome

Quit



Kandisky 1908

**The Formula:**

$AG ((not <sugar\ and\ not\ may> true) or <no\_sugar\ and\ not\ may> true)$

is **TRUE**

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

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

**ACTL-UCTL-SocL-vACTL**

$AG ( (not <sugar>\# true) or (<no\_sugar>\# true) )$

Check The Formula

Explain the Result


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

# A formula verified by VMC over all products of the family

**VMC v6.2**

● ● ● ● ●

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



Kandisky 1908

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

<a href="#">product_01+euro+tea</a>	Formula evaluates	TRUE
<a href="#">product_02+coffee+euro+pour_espresso</a>	Formula evaluates	TRUE
<a href="#">product_03+coffee+euro+pour_regular</a>	Formula evaluates	TRUE
<a href="#">product_04+coffee+euro+pour_espresso+tea</a>	Formula evaluates	TRUE
<a href="#">product_05+coffee+euro+pour_regular+tea</a>	Formula evaluates	TRUE
<a href="#">product_06+cappuccino+coffee+euro+pour_espresso</a>	Formula evaluates	TRUE
<a href="#">product_07+cappuccino+coffee+euro+pour_regular</a>	Formula evaluates	TRUE
<a href="#">product_08+cappuccino+coffee+euro+pour_espresso+tea</a>	Formula evaluates	TRUE
<a href="#">product_09+cappuccino+coffee+euro+pour_regular+tea</a>	Formula evaluates	TRUE
<a href="#">product_10+coffee+dollar+pour_espresso</a>	Formula evaluates	FALSE
<a href="#">product_11+coffee+dollar+pour_regular</a>	Formula evaluates	FALSE
<a href="#">product_12+cappuccino+coffee+dollar+pour_espresso</a>	Formula evaluates	TRUE
<a href="#">product_13+cappuccino+coffee+dollar+pour_regular</a>	Formula evaluates	TRUE

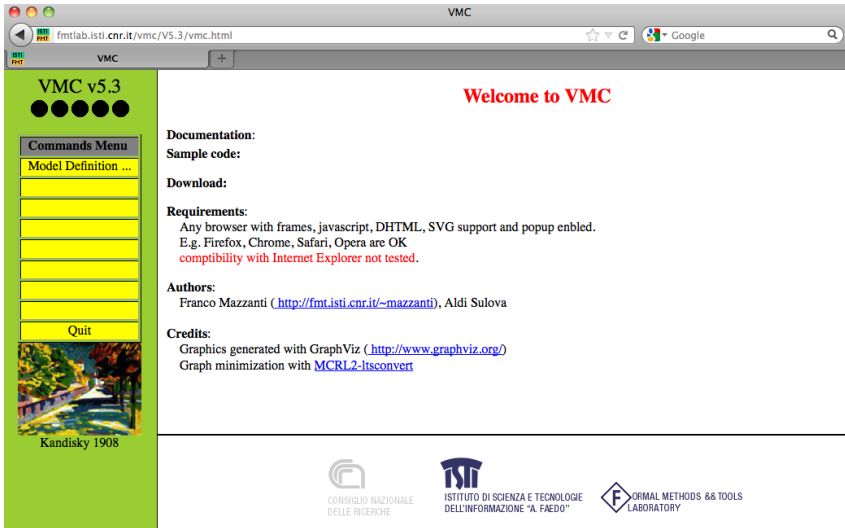
**Logic Formula for all Products**

[dollar] EF {cappuccino} true

Check The Formula   Explain the Result

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

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



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

VMC v5.3

Commands Menu

Model Definition ...

Quit

Kandisky 1908

Welcome to VMC

**Documentation:**

**Sample code:**

**Download:**

**Requirements:**  
Any browser with frames, javascript, DHTML, SVG support and popup enabled.  
E.g. Firefox, Chrome, Safari, Opera are OK  
*compatibility with Internet Explorer not tested.*

**Authors:**  
Franco Mazzanti (<http://fmt.isti.cnr.it/~mazzanti>), Aldi Sulova

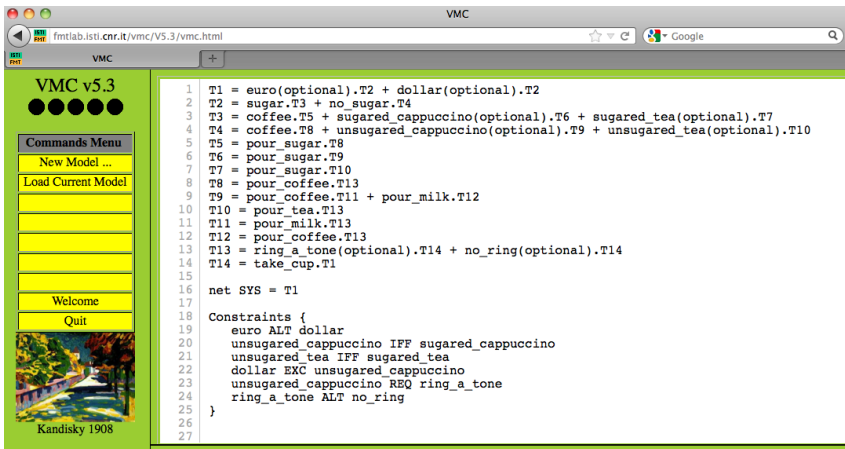
**Credits:**  
Graphics generated with GraphViz (<http://www.graphviz.org/>)  
Graph minimization with [MCRL2-Itscnvert](#)

CONSIGLIO NAZIONALE DELLE RICERCHE

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

FORMAL METHODS && TOOLS LABORATORY

# Family of coffee machines specified in VMC



The screenshot shows a web browser window titled "VMC" with the URL "fmlab.isti.cnr.it/vmc/V5.3/vmc.html". The interface is split into a left sidebar and a main content area. The sidebar, titled "VMC v5.3", contains a "Commands Menu" with buttons for "New Model ...", "Load Current Model", "Welcome", and "Quit", along with a small image of a landscape painting labeled "Kandisky 1908". The main content area displays a list of transition rules (T1 to T14) and a set of constraints. The rules are defined as follows:

```
1 T1 = euro(optional).T2 + dollar(optional).T2
2 T2 = sugar.T3 + no_sugar.T4
3 T3 = coffee.T5 + sugared_cappuccino(optional).T6 + sugared_tea(optional).T7
4 T4 = coffee.T8 + unsugared_cappuccino(optional).T9 + unsugared_tea(optional).T10
5 T5 = pour_sugar.T8
6 T6 = pour_sugar.T9
7 T7 = pour_sugar.T10
8 T8 = pour_coffee.T13
9 T9 = pour_coffee.T11 + pour_milk.T12
10 T10 = pour_tea.T13
11 T11 = pour_milk.T13
12 T12 = pour_coffee.T13
13 T13 = ring_a_tone(optional).T14 + no_ring(optional).T14
14 T14 = take_cup.T1
15
16 net SYS = T1
17
18 Constraints {
19   euro ALT dollar
20   unsugared_cappuccino IFF sugared_cappuccino
21   unsugared_tea IFF sugared_tea
22   dollar EXC unsugared_cappuccino
23   unsugared_cappuccino REQ ring_a_tone
24   ring_a_tone ALT no_ring
25 }
26
27
```

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

# Family/MTS of coffee machines visualized by VMC

VMC v5.3

Commands Menu

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

Kandisky 1919

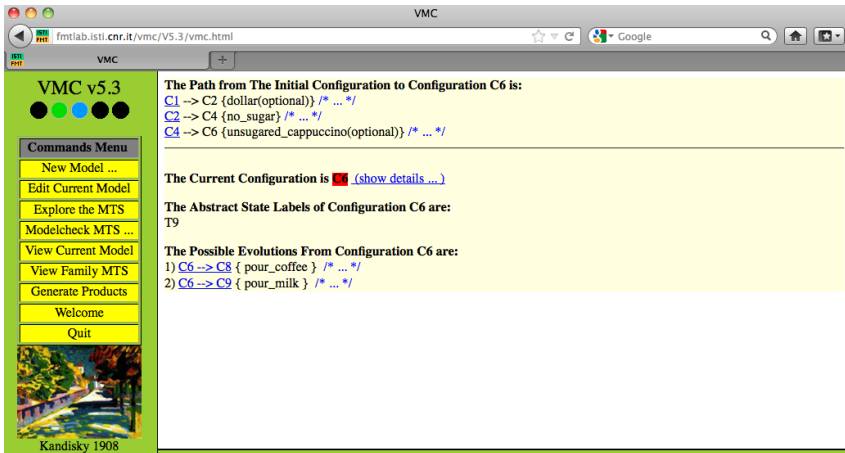
### Family Model Evolutions Chart (MTS)

Zoom Out Zoom In

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

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

# Family/MTS of coffee machines explored in VMC

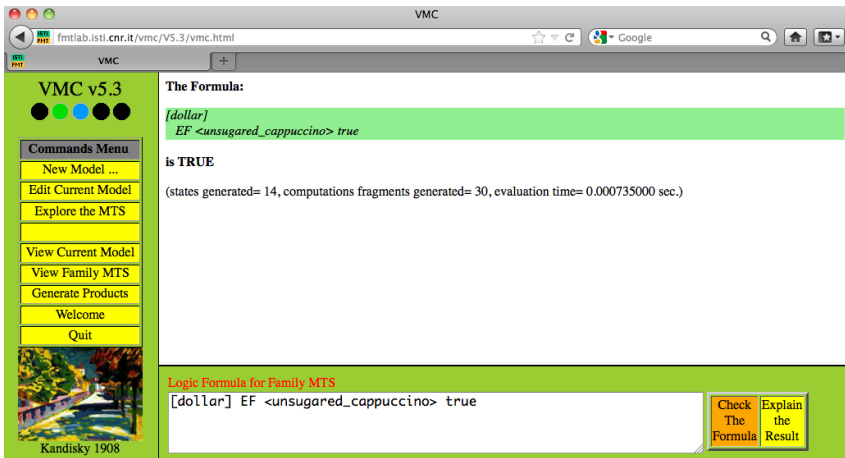


The screenshot shows a web browser window titled "VMC" with the URL "fmtlab.isti.cnr.it/vmc/V5.3/vmc.html". The interface is divided into a left sidebar and a main content area. The sidebar, titled "VMC v5.3", contains a "Commands Menu" with buttons for "New Model ...", "Edit Current Model", "Explore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

- The Path from The Initial Configuration to Configuration C6 is:**
  - C1 -> C2 {dollar(optional)} /\* ... \*/
  - C2 -> C4 {no\_sugar} /\* ... \*/
  - C4 -> C6 {unsugared\_cappuccino(optional)} /\* ... \*/
- The Current Configuration is C6** ([show details ...](#))
- The Abstract State Labels of Configuration C6 are:**  
T9
- The Possible Evolutions From Configuration C6 are:**
  - C6 -> C8 { pour\_coffee } /\* ... \*/
  - C6 -> C9 { pour\_milk } /\* ... \*/

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

# Outcome of a property verified over a family with VMC



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

- VMC v5.3**: A header with four colored circles (black, green, blue, black).
- Commands Menu**: A vertical list of buttons including "New Model ...", "Edit Current Model", "Explore the MTS", "View Current Model", "View Family MTS", "Generate Products", "Welcome", and "Quit".
- The Formula:** A section displaying the formula `[dollar] EF <unsugared_cappuccino> true` on a green background.
- is TRUE**: A section indicating the result of the verification, with a note: "(states generated= 14, computations fragments generated= 30, evaluation time= 0.000735000 sec.)".
- Logic Formula for Family MTS**: A section with a text input field containing the same formula `[dollar] EF <unsugared_cappuccino> true` and two buttons: "Check The Formula" and "Explain the Result".
- Kandisky 1908**: A small image of a painting with the caption "Kandisky 1908".

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

# Outcome of a property explained by VMC

The screenshot shows the VMC v5.3 web interface. On the left is a 'Commands Menu' with options like 'New Model ...', 'Edit Current Model', 'Explore the MTS', 'View Current Model', 'View Family MTS', 'Generate Products', 'Welcome', and 'Quit'. Below the menu is a small image of a bridge and the text 'Kandisky 1908'. The main content area displays the following information:

**VMC v5.3**

**The formula:**  
 $[dollar] EF \langle unsugared\_cappuccino \rangle true$   
is **FOUND\_TRUE** in State C1

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

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

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

---

**The formula:**  
 $EF \langle unsugared\_cappuccino \rangle true$   
is **FOUND\_TRUE** in State C2

**This happens because**  
C2  $\rightarrow$  C4 {no\_sugar} /\* ... \*/  
**and the subformula:**  
 $\langle unsugared\_cappuccino \rangle true$   
is **Satisfied** in State C4

---

**The formula:**  
 $\langle unsugared\_cappuccino \rangle true$   
is **FOUND\_TRUE** in State C4

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

**Logic Formula for Family MTS**  
 $[dollar] EF \langle unsugared\_cappuccino \rangle true$

Buttons: Check The Formula, Explain the Result

# Products of family of coffee machines derived by VMC

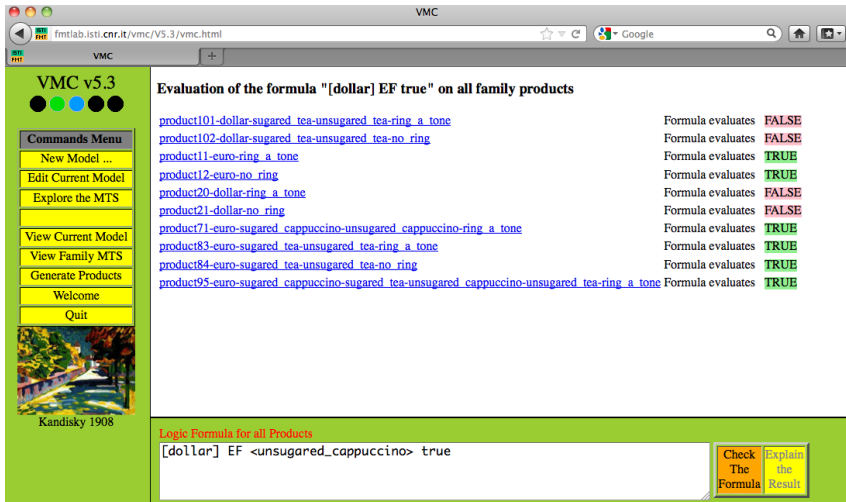


The screenshot shows a web browser window titled "VMC" with the URL "fmtlab.isti.cnr.it/vmc/V5.3/vmc.html". The browser's address bar includes a search engine (Google) and a search icon. The page content is divided into two main sections:

- Left Sidebar:** Contains the title "VMC v5.3" with four colored circles (green, blue, black, black) below it. A "Commands Menu" is listed with the following items: "New Model ...", "Edit Current Model", "Explore the MTS", "Modelcheck Products", "View Current Model", "View Family MTS", "Generate Products", "Welcome", and "Quit". At the bottom of the sidebar is a small image of a landscape painting titled "Kandisky 1908".
- Main Content Area:** Titled "Valid Products of the Family", it lists ten product identifiers as blue underlined links:
  - [product101-dollar-sugared\\_tea-unsugared\\_tea-ring\\_a\\_tone](#)
  - [product102-dollar-sugared\\_tea-unsugared\\_tea-no\\_ring](#)
  - [product11-euro-ring\\_a\\_tone](#)
  - [product12-euro-no\\_ring](#)
  - [product20-dollar-ring\\_a\\_tone](#)
  - [product21-dollar-no\\_ring](#)
  - [product71-euro-sugared\\_cappuccino-unsugared\\_cappuccino-ring\\_a\\_tone](#)
  - [product83-euro-sugared\\_tea-unsugared\\_tea-ring\\_a\\_tone](#)
  - [product84-euro-sugared\\_tea-unsugared\\_tea-no\\_ring](#)
  - [product95-euro-sugared\\_cappuccino-sugared\\_tea-unsugared\\_cappuccino-unsugared\\_tea-ring\\_a\\_tone](#)

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

# Outcomes of a property verified over products with VMC



The screenshot shows the VMC v5.3 web interface. The browser address bar displays `fmlab.lsti.cnr.it/vmc/V5.3/vmc.html`. The page title is "VMC". The main content area is titled "Evaluation of the formula "[dollar] EF true" on all family products". Below this title, a list of products is shown, each with a corresponding evaluation result for the formula. The results are: product101-dollar-sugared\_tea-unsugared\_tea-ring\_a\_tone (FALSE), product102-dollar-sugared\_tea-unsugared\_tea-no\_ring (FALSE), product11-euro-ring\_a\_tone (TRUE), product12-euro-no\_ring (TRUE), product20-dollar-ring\_a\_tone (FALSE), product21-dollar-no\_ring (FALSE), product71-euro-sugared\_cappuccino-unsugared\_cappuccino-ring\_a\_tone (TRUE), product83-euro-sugared\_tea-unsugared\_tea-ring\_a\_tone (TRUE), product84-euro-sugared\_tea-unsugared\_tea-no\_ring (TRUE), and product95-euro-sugared\_cappuccino-sugared\_tea-unsugared\_cappuccino-unsugared\_tea-ring\_a\_tone (TRUE). On the left side, there is a "Commands Menu" with buttons for "New Model ...", "Edit Current Model", "Explore the MTS", "View Current Model", "View Family MTS", "Generate Products", "Welcome", and "Quit". Below the menu is a small image of a path and the text "Kandisky 1908". At the bottom, there is a "Logic Formula for all Products" section with the formula `[dollar] EF <unsugared_cappuccino> true` and two buttons: "Check The Formula" and "Explain the Result".

VMC v5.3

Commands Menu

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

Kandisky 1908

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

<a href="#">product101-dollar-sugared_tea-unsugared_tea-ring_a_tone</a>	Formula evaluates	FALSE
<a href="#">product102-dollar-sugared_tea-unsugared_tea-no_ring</a>	Formula evaluates	FALSE
<a href="#">product11-euro-ring_a_tone</a>	Formula evaluates	TRUE
<a href="#">product12-euro-no_ring</a>	Formula evaluates	TRUE
<a href="#">product20-dollar-ring_a_tone</a>	Formula evaluates	FALSE
<a href="#">product21-dollar-no_ring</a>	Formula evaluates	FALSE
<a href="#">product71-euro-sugared_cappuccino-unsugared_cappuccino-ring_a_tone</a>	Formula evaluates	TRUE
<a href="#">product83-euro-sugared_tea-unsugared_tea-ring_a_tone</a>	Formula evaluates	TRUE
<a href="#">product84-euro-sugared_tea-unsugared_tea-no_ring</a>	Formula evaluates	TRUE
<a href="#">product95-euro-sugared_cappuccino-sugared_tea-unsugared_cappuccino-unsugared_tea-ring_a_tone</a>	Formula evaluates	TRUE

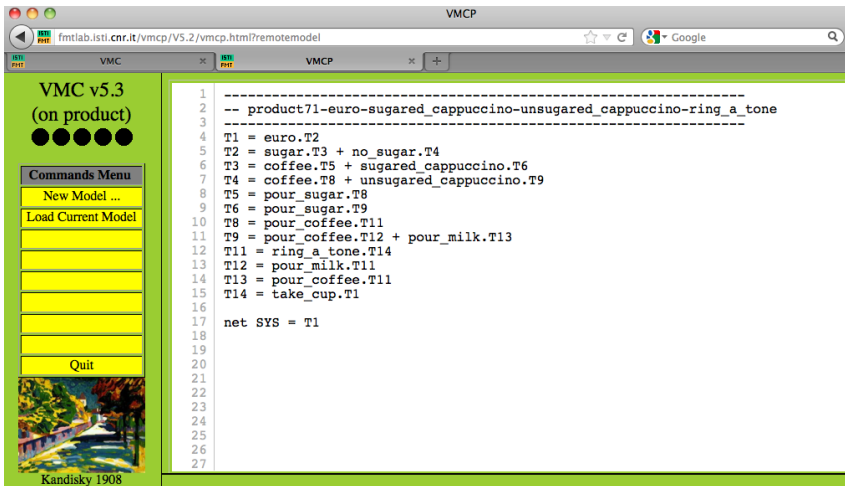
Logic Formula for all Products

```
[dollar] EF <unsugared_cappuccino> true
```

Check The Formula Explain the Result

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

# Specification of one of the products derived by VMC



The screenshot shows a web browser window titled "VMCP" with the URL `fmlab.isitl.cnr.it/vmcp/V5.2/vmcp.html?remotemodel`. The interface is split into two main sections. On the left, a green sidebar contains the text "VMC v5.3 (on product)" with five black circles below it. Underneath is a "Commands Menu" with buttons for "New Model ...", "Load Current Model", and "Quit". At the bottom of the sidebar is a small image of a bridge over a stream, labeled "Kandisky 1908". The main area on the right is a text editor showing a textual encoding of a product. The text is as follows:

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

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

# Product/LTS on previous slide visualized by VMC

VMCP

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


VMC VMCP

## VMC v5.3 (on product)

● ● ● ● ● ●

### Commands Menu

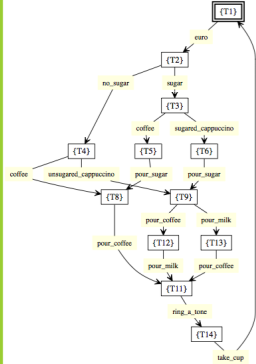
- New Model ...
- Edit Current Model
- Explore the LTS
- View Current Model
- View the LTS Graph
- Quit



Kandisky 1919

## Product Model Evolutions Chart (LTS)

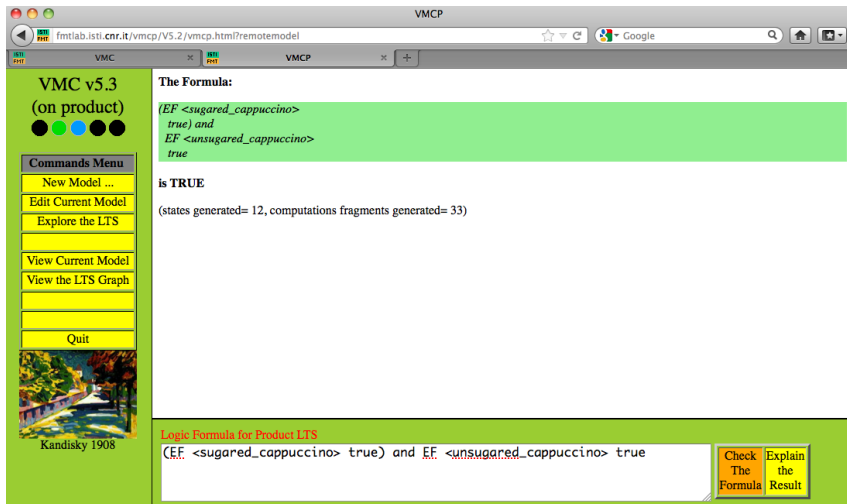
Zoom Out Zoom In



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

The above graph shows the LTS product model evolutions, which by definition contains only full edges.

# Outcome of a property verified over a product with VMC



The screenshot shows a web browser window titled "VMCP" with the URL `fmlab.isti.cnr.it/vmcp/V5.2/vmcp.htm?remotemodel`. The interface is for VMC v5.3 (on product). On the left is a "Commands Menu" with buttons for "New Model ...", "Edit Current Model", "Explore the LTS", "View Current Model", "View the LTS Graph", and "Quit". Below the menu is a small image of a path with the caption "Kandisky 1908".

The main content area displays the following information:

**The Formula:**

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

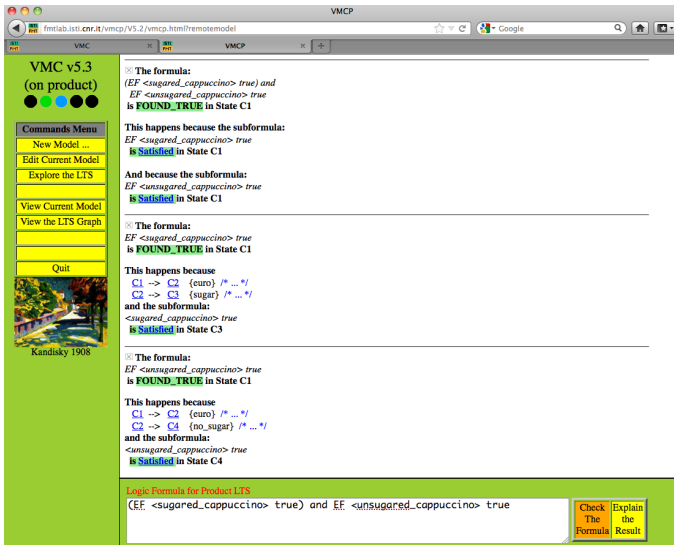
is **TRUE**

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

At the bottom, a box titled "Logic Formula for Product LTS" contains the formula: `(EF <sugared_cappuccino> true) and EF <unsugared_cappuccino> true`. To the right of this box are two buttons: "Check The Formula" and "Explain the Result".

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

# Outcome property on previous slide explained by VMC




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

**VMC v5.3 (on product)**

**Commands Menu**

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

  
Kandisky 1908

**The formula:**  
 $(EF \langle \text{sugared\_cappuccino} \rangle \text{ true}) \text{ and } EF \langle \text{unsugared\_cappuccino} \rangle \text{ true}$   
is **FOUND\_TRUE** in State C1

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

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

**The formula:**  
 $EF \langle \text{sugared\_cappuccino} \rangle \text{ true}$   
is **FOUND\_TRUE** in State C1

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

**The formula:**  
 $EF \langle \text{unsugared\_cappuccino} \rangle \text{ true}$   
is **FOUND\_TRUE** in State C1

**This happens because**  
 $C1 \rightarrow C2 \{ \text{euro} \} !^* \dots !^*$   
 $C2 \rightarrow C4 \{ \text{no\_sugar} \} !^* \dots !^*$   
**and the subformula:**  
 $\langle \text{unsugared\_cappuccino} \rangle \text{ true}$   
is **Satisfied** in State C4

**Logic Formula for Product LTS**  
 $(EF \langle \text{sugared\_cappuccino} \rangle \text{ true}) \text{ and } EF \langle \text{unsugared\_cappuccino} \rangle \text{ true}$

**Check The Formula** **Explain the Result**

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

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

$[P]$  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  $K(v) \stackrel{\text{def}}{=} T$

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

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

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

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

# Semantics in SOS style over MTS

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

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

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

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

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

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

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

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

## 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: ±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 *Bicincittà 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: ±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 *Bicincittà 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: ±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 *Bicincittà 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: ±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 *Bicincittà 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: ±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 *Bicincittà 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: ±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 **Bicincittà 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)
```

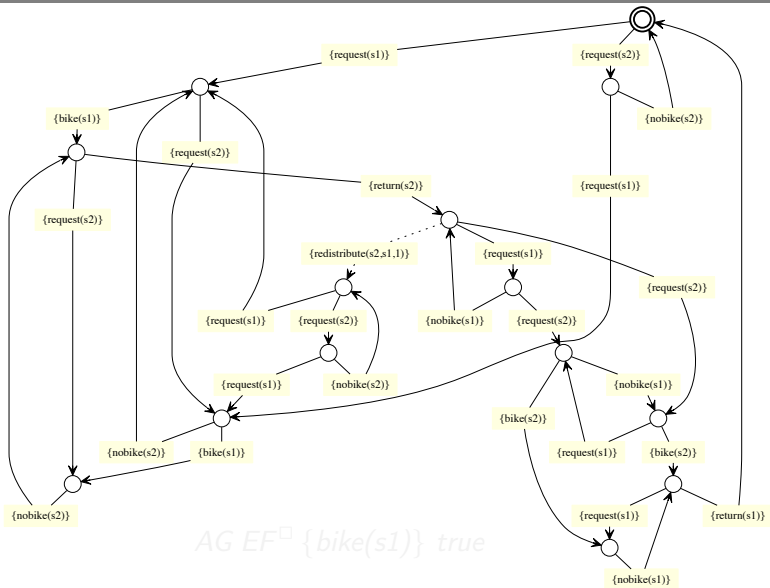
```
Users(I,J) = request(I).
```

```
  ( bike(I).return(J).Users(I,J) +  
    nobike(I).Users(I,J) )
```

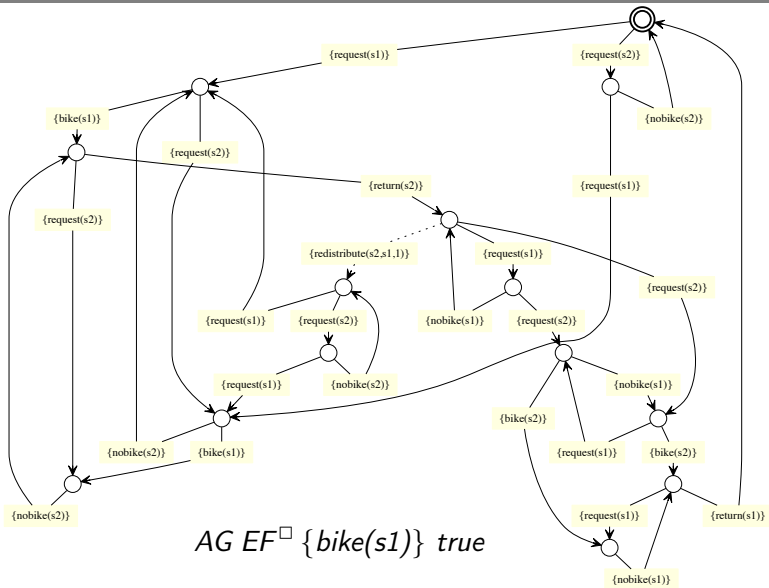
```
net USERS = Users(s1,s2) // Users(s2,s1)
```

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

# MTS with parameters and values



# MTS with parameters and values



# v-ACTL<sup>□</sup> formula: true $\forall$ products

Products generated by VMC  
(not needed for verification)

