Modelling, analysing and verifying variability by means of Modal Transition Systems

M.H. ter Beek, A. Fantechi, S. Gnesi
and other FMT members

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

Dagstuhl Seminar 13091
24 February - 1 March 2013

Outline

1. Aim and achievements of our research on variability
2. Running example: a family of coffee machines
3. Our Variability Model Checker VMC
4. Discussion and future work

Aim and achievements

Aim

- To develop a logical framework capable of dealing with variability (behavioural)
- To provide tools to support this framework with formal verification (model checking)

Main ingredient: Modal Transition Systems (Larsen et al.)

- Labelled Transition Systems distinguishing **may** (admissible) and **must** (necessary) transitions

ESEC/FSE'07, SPLC'08, VaMoS'09, VaMoS'10, iFM'10, ACoTA @ ASE'10, PLEASE @ ICSE'11, FMOODS/FORTE'11, SEW-34 @ FM'11, SPLC'11, demo @ iFM'12, tool @ FM'12, FMSPLE'12, demo @ SPLC'12, ISoLA'12,...

- v-ACTL: ACTL-like branching-time temporal logic (a.k.a. MHML)
- VMC: Variability Model Checker

Aim and achievements

Aim

- To develop a logical framework capable of dealing with variability (behavioural)
- To provide tools to support this framework with formal verification (model checking)

Main ingredient: Modal Transition Systems (Larsen et al.)

- Labelled Transition Systems distinguishing **may** (admissible) and **must** (necessary) transitions

ESEC/FSE'07, SPLC'08, VaMoS'09, VaMoS'10, iFM'10, ACoTA @ ASE'10, PLEASE @ ICSE'11, FMOODS/FORTE'11, SEW-34 @ FM'11, SPLC'11, demo @ iFM'12, tool @ FM'12, FMSPLE'12, demo @ SPLC'12, ISoLA'12,...

- v-ACTL: ACTL-like branching-time temporal logic (a.k.a. MHML)
- VMC: Variability Model Checker

Aim and achievements

Aim

- To develop a logical framework capable of dealing with variability (behavioural)
- To provide tools to support this framework with formal verification (model checking)

Main ingredient: Modal Transition Systems (Larsen et al.)

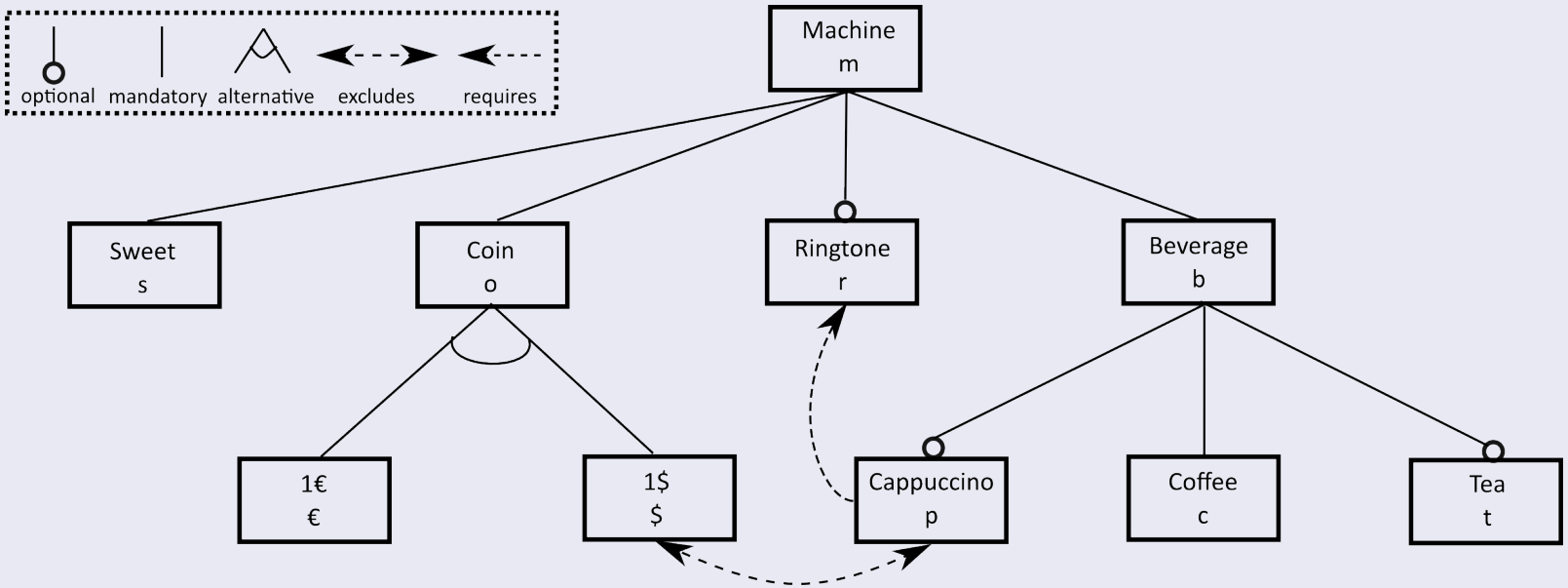
- Labelled Transition Systems distinguishing **may** (admissible) and **must** (necessary) transitions

ESEC/FSE'07, SPLC'08, VaMoS'09, VaMoS'10, iFM'10, ACoTA @ ASE'10, PLEASE @ ICSE'11, FMOODS/FORTE'11, SEW-34 @ FM'11, SPLC'11, demo @ iFM'12, tool @ FM'12, FMSPLE'12, demo @ SPLC'12, ISoLA'12,...

- v-ACTL: ACTL-like branching-time temporal logic (a.k.a. MHML)
- VMC: Variability Model Checker

Running example: family of coffee machines

Feature-based constraints

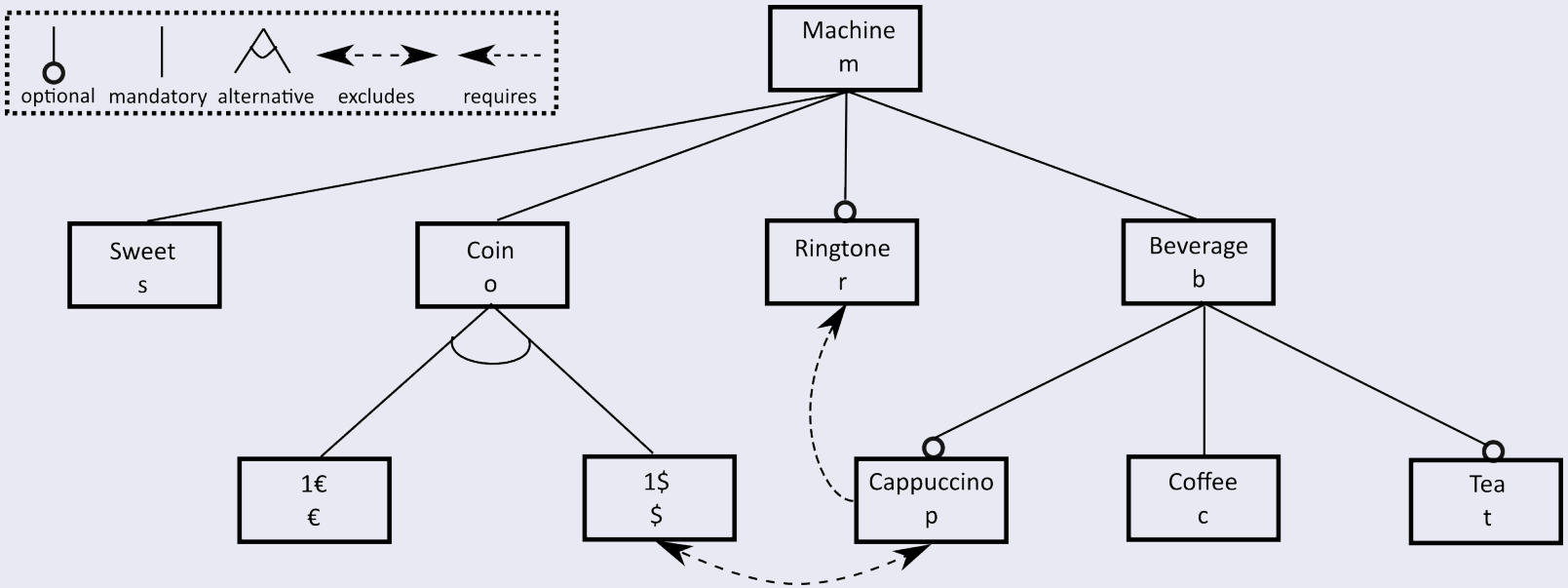


Behavioural constraints

- **After** coin insertion, user must press a button to choose whether (s)he wants sugar, **after** which (s)he may select a beverage
- The optional ringtone is rung **after** delivering a beverage
- The machine returns to its idle state **after** the beverage is taken

Running example: family of coffee machines

Feature-based constraints



Behavioural constraints

- **After** coin insertion, user must press a button to choose whether (s)he wants sugar, **after** which (s)he may select a beverage
- The optional ringtone is rung **after** delivering a beverage
- The machine returns to its idle state **after** the beverage is taken

Modal Transition Systems (MTSs)

Their use for behavioural modelling of SPLs due to Uchitel et al.

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

- *shared behaviour*, common among all variants, and
- *variation points*, differentiating between variants

An LTS is obtained by including all (reachable) must transitions and a subset of the (reachable) may transitions—each selection is a variant (a.k.a. an implementation)

Limitation

MTSs cannot model **alternative** features nor **requires**/**excludes** CTC

Our solution

Additional variability constraints to single out the valid variants (and an algorithm to derive only—and possibly all—valid ones)

Modal Transition Systems (MTSs)

Their use for behavioural modelling of SPLs due to Uchitel et al.

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

- *shared behaviour*, common among all variants, and
- *variation points*, differentiating between variants

An LTS is obtained by including all (reachable) must transitions and a subset of the (reachable) may transitions—each selection is a variant (a.k.a. an implementation)

Limitation

MTSs cannot model **alternative** features nor **requires**/**excludes** CTC

Our solution

Additional variability constraints to single out the valid variants (and an algorithm to derive only—and possibly all—valid ones)

Modal Transition Systems (MTSs)

Their use for behavioural modelling of SPLs due to Uchitel et al.

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

- *shared behaviour*, common among all variants, and
- *variation points*, differentiating between variants

An LTS is obtained by including all (reachable) must transitions and a subset of the (reachable) may transitions—each selection is a variant (a.k.a. an implementation)

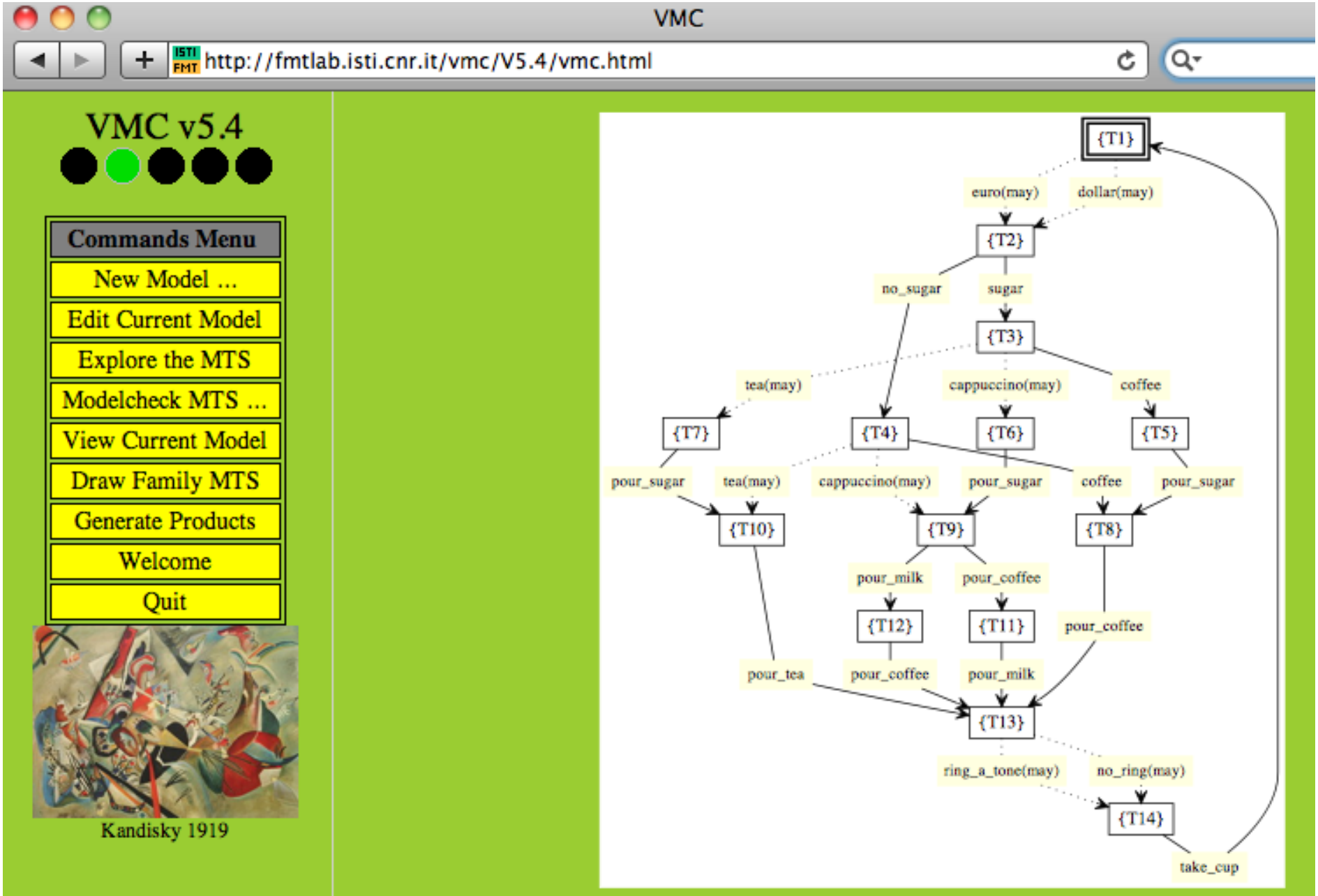
Limitation

MTSs cannot model **alternative** features nor **requires**/**excludes** CTC

Our solution

Additional variability constraints to single out the valid variants (and an algorithm to derive only—and possibly all—valid ones)

Coffee machine family: a possible MTS model



A valid European coffee machine

VMCP

http://fmtlab.isti.cnr.it/vmcp/V5.2/vmcp.html?remotemodel

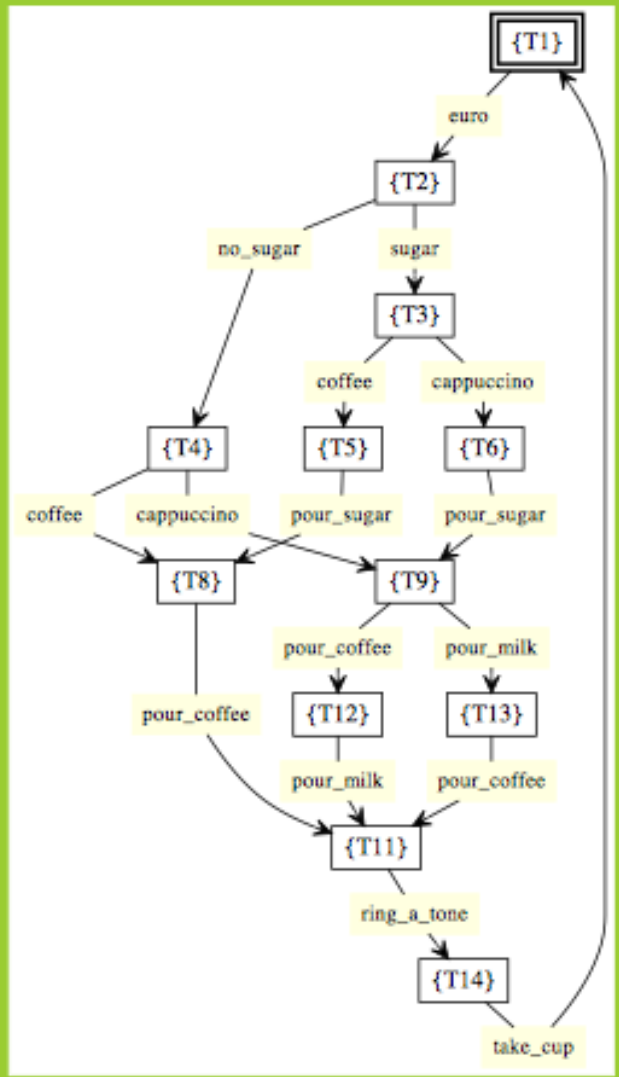
VMC | VMCP

VMCproduct
(on product)

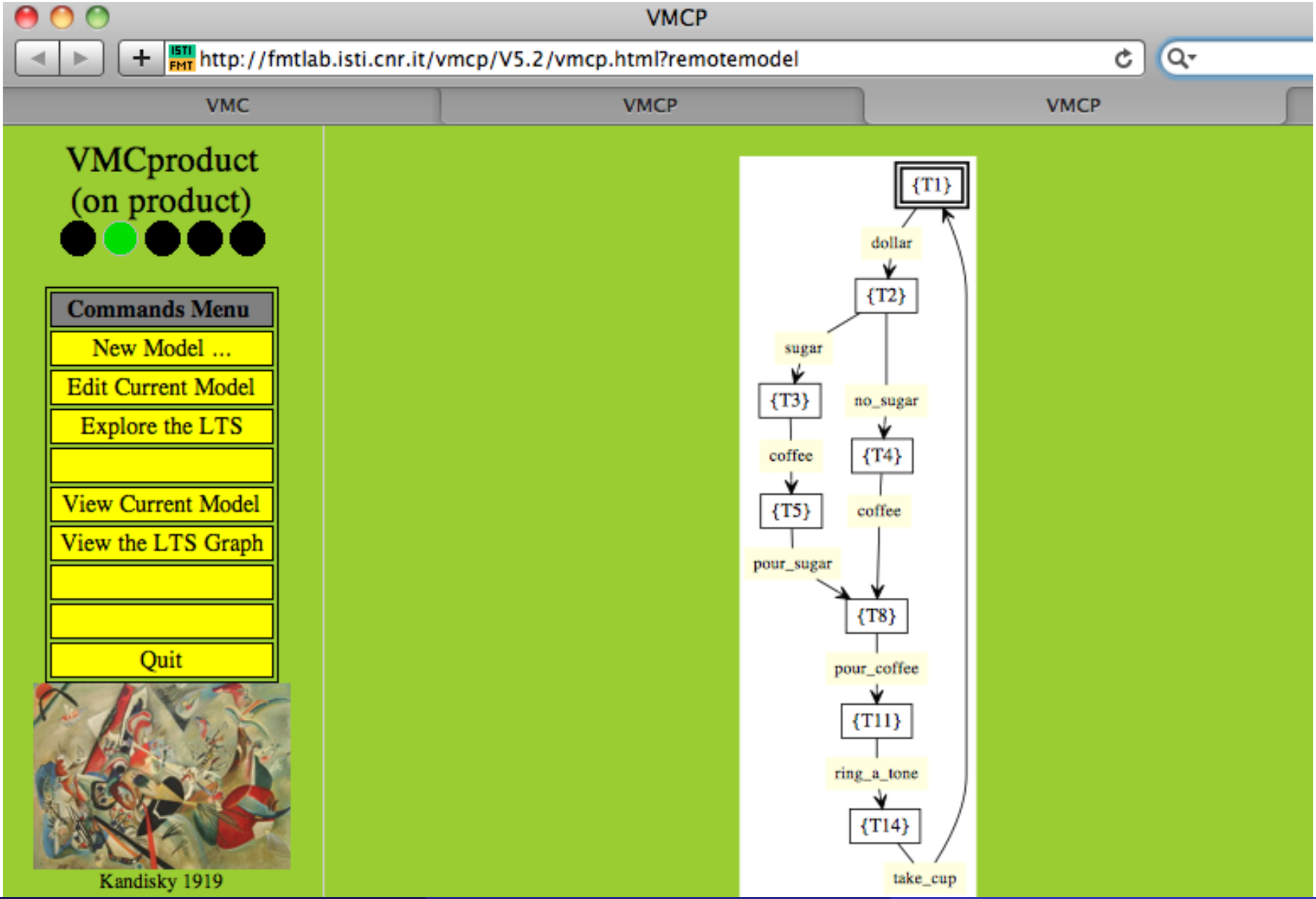
Commands Menu

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

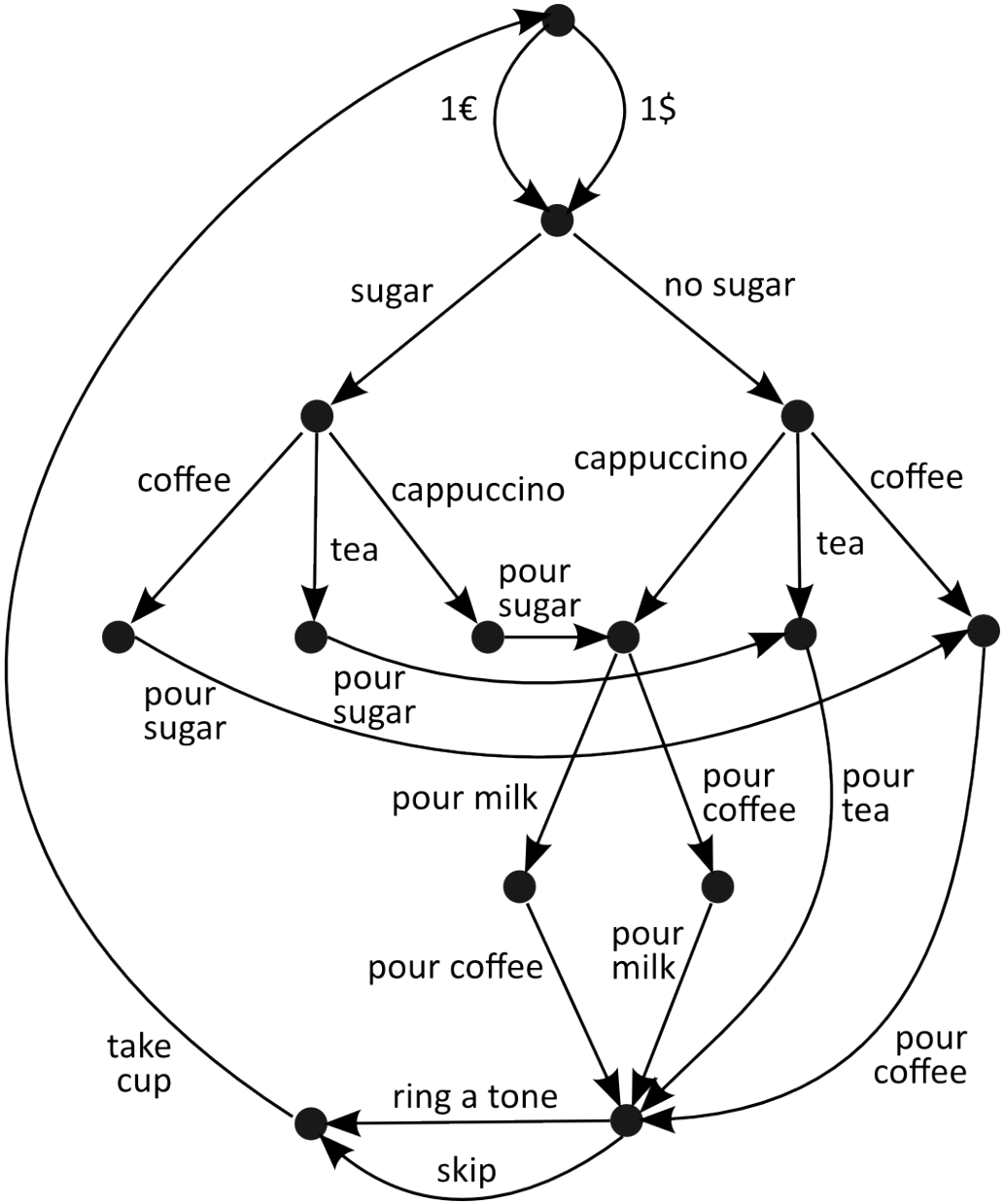
Kandisky 1919



A valid Canadian coffee machine



A variant LTS of above MTS that is not valid



v-ACTL: A logic interpreted over MTSs

Variability and action-based branching-time temporal logic

Based on action-based logic ACTL (De Nicola et al.), state-based logic CTL (Clarke et al.) and Hennessy-Milner Logic with Until (Larsen et al.)
Plus *deontic* interpretations of operators based on *must paths* in MTSs

Syntax of v-ACTL

$$\phi ::= true \mid \neg\phi \mid \phi \wedge \phi \mid \langle\varphi\rangle\phi \mid \langle\varphi\rangle^{\Box}\phi \mid [\varphi]\phi \mid [\varphi]^{\Box}\phi \mid E\,\pi \mid A\,\pi$$

$$\pi ::= \phi\ {}_{\varphi}U_{\varphi}\ \phi \mid \phi\ {}_{\varphi}U^{\Box}_{\varphi}\ \phi \mid \phi\ {}_{\varphi}W_{\varphi}\ \phi \mid \phi\ {}_{\varphi}W^{\Box}_{\varphi}\ \phi$$

Thus defines state formulae ϕ, path formulae π and action formulae φ (boolean compositions of actions) over set of atomic actions $\{a, b, \ldots\}$

Further operators

$F\phi$, $F_{\varphi}\ true$, $F^{\Box}_{\varphi}\ true$, $F^{\Box}\ \phi$, $AG\,\phi$, $AG^{\Box}\ \phi$, ...

v-ACTL: A logic interpreted over MTSs

Variability and action-based branching-time temporal logic

Based on action-based logic ACTL (De Nicola et al.), state-based logic CTL (Clarke et al.) and Hennessy-Milner Logic with Until (Larsen et al.)
Plus *deontic* interpretations of operators based on *must paths* in MTSs

Syntax of v-ACTL

$$\phi ::= \mathit{true} \mid \neg\phi \mid \phi \wedge \phi \mid \langle\varphi\rangle\,\phi \mid \langle\varphi\rangle^{\Box}\,\phi \mid [\varphi]\,\phi \mid [\varphi]^{\Box}\,\phi \mid E\,\pi \mid A\,\pi$$

$$\pi ::= \phi\ {}_{\varphi}U_{\varphi}\ \phi \mid \phi\ {}_{\varphi}U^{\Box}_{\varphi}\ \phi \mid \phi\ {}_{\varphi}W_{\varphi}\ \phi \mid \phi\ {}_{\varphi}W^{\Box}_{\varphi}\ \phi$$

Thus defines state formulae ϕ, path formulae π and action formulae φ (boolean compositions of actions) over set of atomic actions $\{a, b, \ldots\}$

Further operators

$F\,\phi$, $F_{\varphi}\ \mathit{true}$, $F^{\Box}_{\varphi}\ \mathit{true}$, $F^{\Box}\,\phi$, $AG\,\phi$, $AG^{\Box}\,\phi$, ...

v-ACTL: A logic interpreted over MTSs

Variability and action-based branching-time temporal logic

Based on action-based logic ACTL (De Nicola et al.), state-based logic CTL (Clarke et al.) and Hennessy-Milner Logic with Until (Larsen et al.)
Plus *deontic* interpretations of operators based on *must paths* in MTSs

Syntax of v-ACTL

$$\phi ::= \mathit{true} \mid \neg\phi \mid \phi \wedge \phi \mid \langle\varphi\rangle\,\phi \mid \langle\varphi\rangle^{\Box}\,\phi \mid [\varphi]\,\phi \mid [\varphi]^{\Box}\,\phi \mid E\,\pi \mid A\,\pi$$

$$\pi ::= \phi\ {}_{\varphi}U_{\varphi}\ \phi \mid \phi\ {}_{\varphi}U^{\Box}_{\varphi}\ \phi \mid \phi\ {}_{\varphi}W_{\varphi}\ \phi \mid \phi\ {}_{\varphi}W^{\Box}_{\varphi}\ \phi$$

Thus defines state formulae ϕ, path formulae π and action formulae φ (boolean compositions of actions) over set of atomic actions $\{a, b, \ldots\}$

Further operators

$F\phi$, $F_{\varphi}\ \mathit{true}$, $F^{\Box}_{\varphi}\ \mathit{true}$, $F^{\Box}\,\phi$, $AG\,\phi$, $AG^{\Box}\,\phi$, ...

Tool: Variability Model Checker (VMC)

Modelling, analysing and verifying behavioural variability

Given a textual encoding of an MTS and a set of variability constraints:

- interactively explore the model (MTS)
- derive and explore (all) the model's valid variants (LTSs)
- visualize the model/variants graphically as MTS/LTSs
- verify branching-time temporal logic properties over MTSs/LTSs
- interactively explain why a property is (not) satisfied

Freely usable online: `http://fmtlab.isti.cnr.it/vmc/`

Core is a command-line version of the model checker and of a variant generation procedure, both stand-alone executables written in Ada, so:

- easy to compile for Windows | Linux | Solaris | Mac OS X
- wrapped with a set of CGI scripts handled by a web server:
  - easy to build graphical html-oriented GUI
  - easy to integrate with graph drawing tools

Tool: Variability Model Checker (VMC)

Modelling, analysing and verifying behavioural variability

Given a textual encoding of an MTS and a set of variability constraints:

- interactively explore the model (MTS)
- derive and explore (all) the model's valid variants (LTSs)
- visualize the model/variants graphically as MTS/LTSs
- verify branching-time temporal logic properties over MTSs/LTSs
- interactively explain why a property is (not) satisfied

Freely usable online: `http://fmtlab.isti.cnr.it/vmc/`

Core is a command-line version of the model checker and of a variant generation procedure, both stand-alone executables written in Ada, so:

- easy to compile for Windows | Linux | Solaris | Mac OS X
- wrapped with a set of CGI scripts handled by a web server:
  - easy to build graphical html-oriented GUI
  - easy to integrate with graph drawing tools

Family of coffee machines specified in VMC

VMC

http://fmtlab.isti.cnr.it/vmc/V5.4/vmc.html

VMC v5.4

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

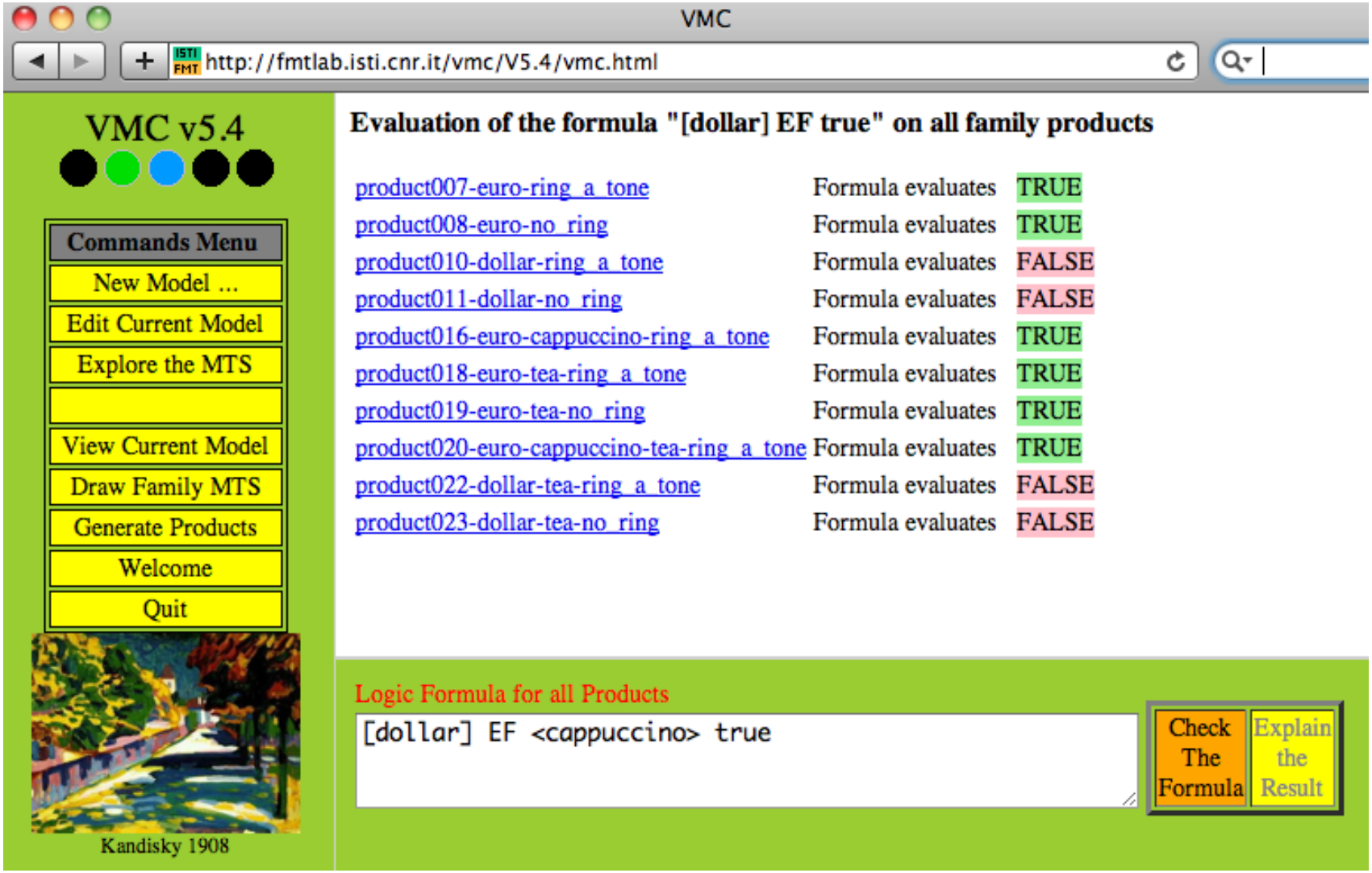
Kandisky 1908

```
T1 = euro(may).T2 + dollar(may).T2
T2 = sugar.T3 + no_sugar.T4
T3 = coffee.T5 + cappuccino(may).T6 + tea(may).T7
T4 = coffee.T8 + cappuccino(may).T9 + tea(may).T10
T5 = pour_sugar.T8
T6 = pour_sugar.T9
T7 = pour_sugar.T10
T8 = pour_coffee.T13
T9 = pour_coffee.T11 + pour_milk.T12
T10= pour_tea.T13
T11 = pour_milk.T13
T12 = pour_coffee.T13
T13 = ring_a_tone(may).T14 + no_ring(may).T14
T14 = take_cup.T1

net SYS = T1

Constraints {
   euro ALT dollar
   dollar EXC cappuccino
   cappuccino REQ ring_a_tone
   ring_a_tone ALT no_ring
}
```

A property verified over variants with VMC



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

VMC in relation to other model checkers

VMC itself is a product of a family of model checkers

- developed and highly optimized at ISTI–CNR during last decades
- on-the-fly: in general no need to generate/explore full state space
- capable of verifying properties specified in a CTL-like action- and state-based branching-time temporal logic

Products: FMC (input: CCS/CSP/LOTOS-like process algebras), UMC (input: UML state machines) & CMC (input: WS orchestration calculus)

Detailed comparison with MTSA, SNIP & SPLVERIFIER: SPLC'12

MTSA (D'Ippolito et al.): built on top of LTSA, i.e. also LTS-based, but no specific features for analysing variability

SNIP (Classen et al.): no GUI, but built-in support for feature diagrams and model-checking algorithms specifically tailored for SPLE

SPLVERIFIER (Apel et al.): does focus on features, but addressing the detection of feature interactions rather than variability

VMC in relation to other model checkers

VMC itself is a product of a family of model checkers

- developed and highly optimized at ISTI–CNR during last decades
- on-the-fly: in general no need to generate/explore full state space
- capable of verifying properties specified in a CTL-like action- and state-based branching-time temporal logic

Products: FMC (input: CCS/CSP/LOTOS-like process algebras), UMC (input: UML state machines) & CMC (input: WS orchestration calculus)

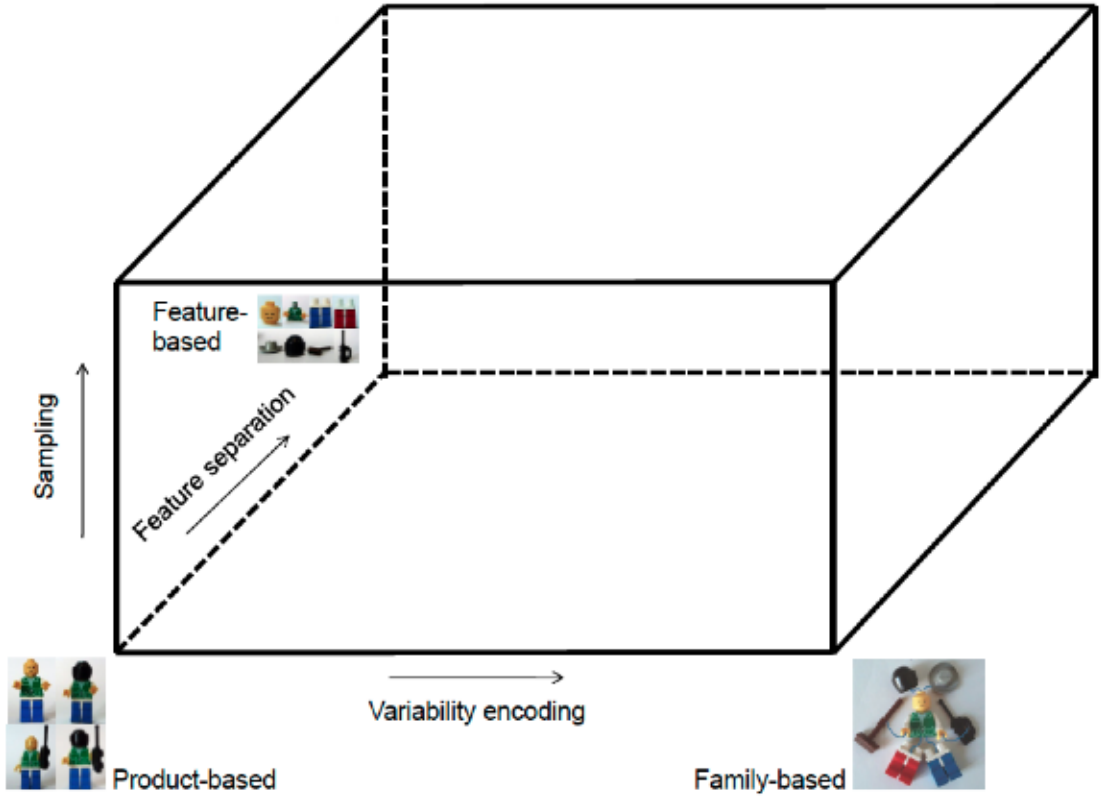
Detailed comparison with MTSA, SNIP & SPLVERIFIER: SPLC'12

MTSA (D'Ippolito et al.): built on top of LTSA, i.e. also LTS-based, but no specific features for analysing variability

SNIP (Classen et al.): no GUI, but built-in support for feature diagrams and model-checking algorithms specifically tailored for SPLE

SPLVERIFIER (Apel et al.): does focus on features, but addressing the detection of feature interactions rather than variability

VMC's position in the PLA cube (Apel et al.)



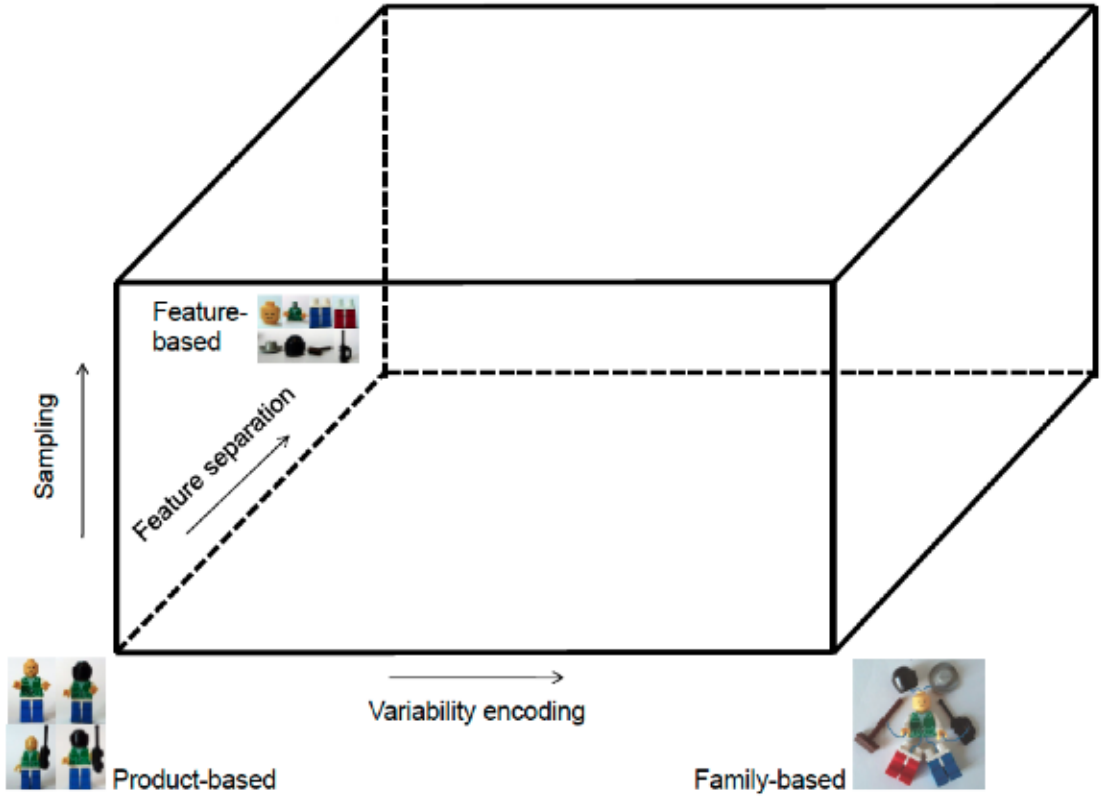
From family-based to product-based analysis

VMC can also be used to model and analyse a subfamily or a specific subset of a family's valid variants by restrictions applied via constraints

Add constraint `coffee EXC dollar` to the family specification

Only European coffee machines will be derived as valid variants, which can then be analyzed both as a subset and individually

VMC's position in the PLA cube (Apel et al.)



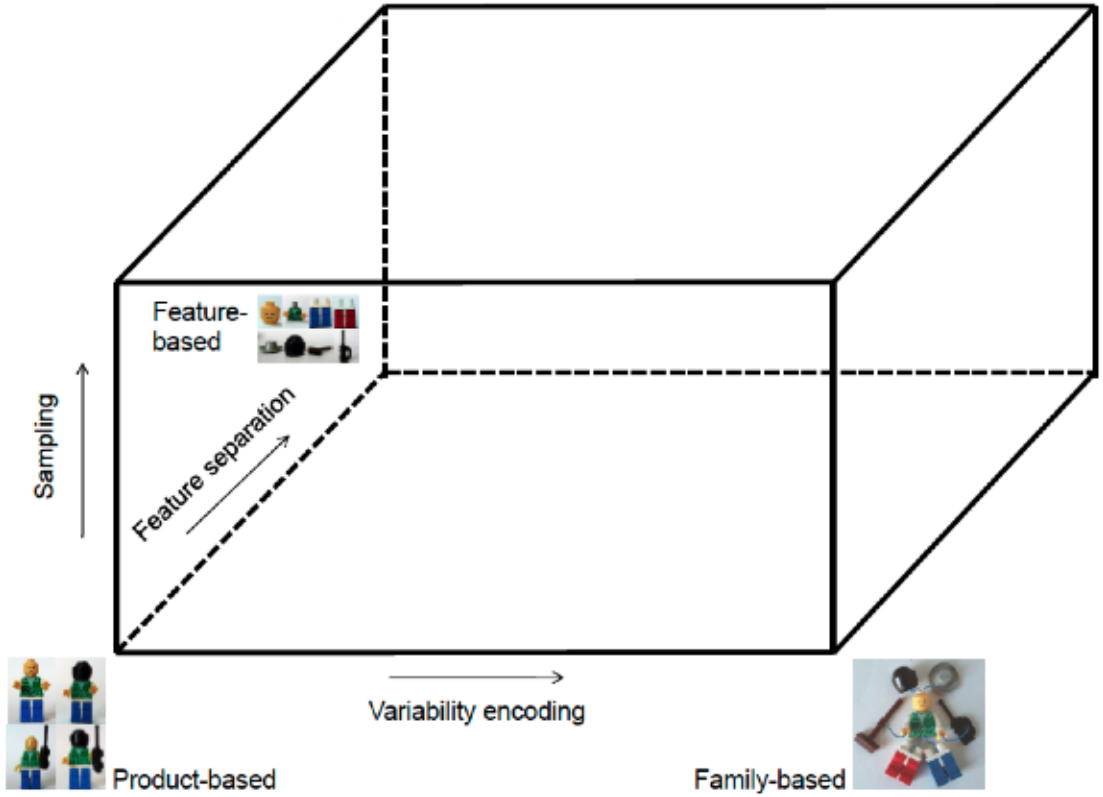
From family-based to product-based analysis

VMC can also be used to model and analyse a subfamily or a specific subset of a family's valid variants by restrictions applied via constraints

Add constraint `coffee EXC dollar` to the family specification

Only European coffee machines will be derived as valid variants, which can then be analyzed both as a subset and individually

VMC's position in the PLA cube (Apel et al.)



From family-based to product-based analysis

VMC can also be used to model and analyse a subfamily or a specific subset of a family's valid variants by restrictions applied via constraints

Add constraint `coffee EXC dollar` to the family specification

Only European coffee machines will be derived as valid variants, which can then be analyzed both as a subset and individually

Future work

1. Consider more expressive behavioural models (e.g. our (G)EMTSs)
2. Feature-based analysis, building on feature model composition (Acher et al.)
3. Confront with / learn from dynamic adaptation / reconfiguration, where all possible variants are often not known upfront