

Families of dependable systems: A model checking approach

S. Gnesi

joint work with

P. Asirelli, M.H. ter Beek, A. Fantechi, F. Mazzanti

ISTI-CNR

Marktoberdorf Summer School 2012

31July- 12 August 2012

Product Line Engineering (PLE)

Paradigm

To develop a family of products using a common platform and mass customization

Aim

To lower production costs of the individual products by

- letting them share an overall reference model of the product family
- allowing them to differ w.r.t. particular characteristics to serve, e.g., different markets

Production process

Organized so as to maximize commonalities of the products and at the same time minimize the cost of variations

Feature modelling

Provide compact representations of all the products of a product family in terms of their features. Feature Models (FM) are used to model sets of software systems in terms of features and relations among them. A feature can be defined as an increment in product functionality.

Variability modelling

FMs are commonly used as a compact representation of all the products of an SPL in terms of features. A FM is visually represented as a tree-like structure in which nodes represent features, and connections illustrate the relationships between them.

Feature models

Feature models are a special type of information model widely used in software product line engineering. A feature model is represented as a hierarchically arranged set of features composed by:

- Relationships between a parent (or compound) feature and its child features (or subfeatures).
- Cross-tree (or cross-hierarchy) constraints that are typically inclusion or exclusion statements in the form: if feature F is included, then features A and B must also be included (or excluded).

Feature diagrams

- A family of popular modeling languages used for engineering requirements in SPL
- Represented as the nodes of a tree, with the product family being the root and have the following features:

How to explicitly define the features or components of a product family that are **optional**, **alternative**, **mandatory**, or **or-Relation**

optional features may be present in a product only if their parent is present

mandatory features are present in a product if and only if their parent is present

alternative features are a set of features among which one and only one is present in a product if their parent is present

or one or more of a set of features can be included in the products if their parent is present

Feature models

- With additional inter-feature constraints a feature diagram results in a feature model:

requires is a unidirectional relation between two features indicating that the presence of one feature requires the presence of the other

excludes is a bidirectional relation between two features indicating that the presence of either feature is incompatible with the presence of the other

FeatureModels were first introduced as a part of the Feature-Oriented Domain Analysis method (FODA) by Kang back in 1990

Running example: Coffee machine family

- The only accepted coins are the 1 euro (1€), exclusively for the European products and the 1 dollar (1\$), exclusively for the Canadian products: 1€ and 1\$ are exclusive (**alternative**) features
- The choice of beverage (coffee, tea, cappuccino) varies, but coffee must be offered by all products of the family, while cappuccino may be offered solely by European products: **excludes** relation between features
- The user has to choose whether or not (s)he wants sugar, by pressing one of two buttons, after which (s)he may select a beverage.
- Optionally, a ringtone may be rung after delivering a beverage . However, a ringtone must be present in all products offering cappuccino: **requires** relation between features.

Running example: Coffee machine family

Feature model:

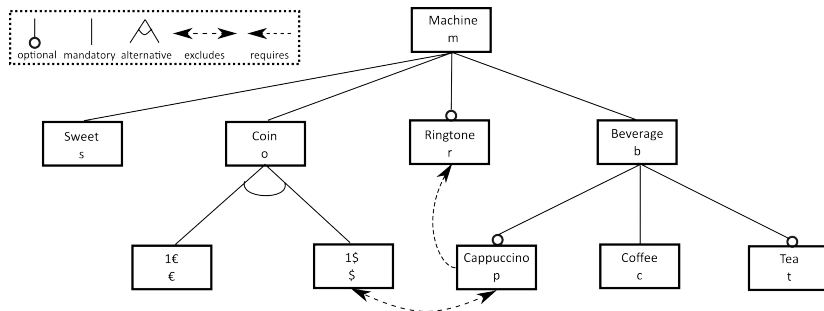


Figure: Feature model of the family of coffee machines (with shorthand names)

Running example: Coffee machine family

This family consists of the following 10 products (coffee machines defined by their features):

$$\{\{m, o, b, c, \epsilon\}, \{m, s, o, b, c, \epsilon, r\}, \{m, o, b, c, \epsilon, t\}, \\ \{m, o, b, c, \epsilon, t, r\}, \{m, o, b, c, \epsilon, p, r\}, \\ \{m, o, b, c, \$\}, \{m, o, b, c, \$, r\}, \{m, o, b, c, \$, t\}, \\ \{m, o, b, c, \$, t, r\}, \{m, o, b, c, \epsilon, p, r, t\}\}$$

Managing variability formally

Show that a certain product belongs to a product family or, instead, derive a product from a family by means of a proper selection of the features or components

A feature model can be characterized by a propositional logic formula:

$$(m \iff true) \wedge (o \iff m) \wedge ((\epsilon \iff (\neg \$ \wedge o)) \wedge (\$ \iff (\neg \epsilon \wedge o))) \wedge (r \implies m) \\ \wedge (b \iff m) \wedge ((p \implies b) \wedge (c \iff b) \wedge (t \implies b)) \wedge (p \implies r) \wedge (\neg (\$ \wedge p))$$

Suppose we have defined two coffee machines with the following sets of features:

$$CM1 = \{m, o, b, c, \epsilon\} \quad \text{and} \quad CM2 = \{m, o, b, c, \epsilon, p\}$$

S.P.L.O.T.: <http://www.splot-research.org>

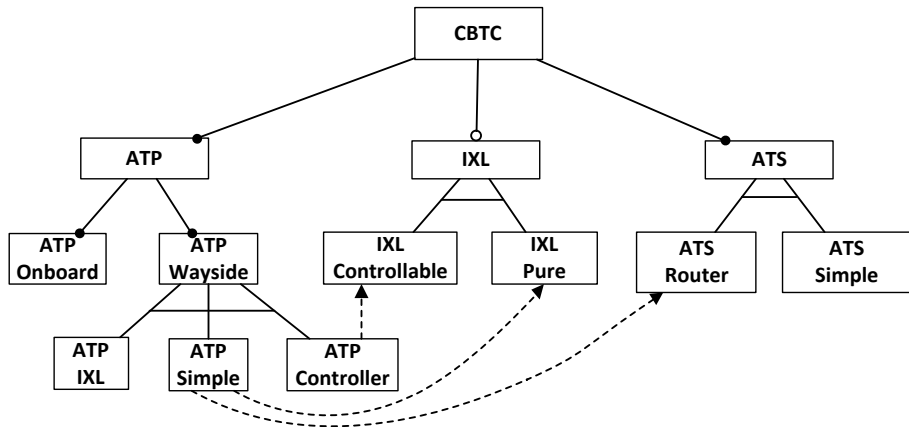
Generative Software Development Lab / Computer Systems Group,
University of Waterloo, Canada, 2009-2010.

- Online Feature Model Editor
- Automated Analysis
- Feature-based Interactive Configuration
- Feature Models Repository

Static requirements of CBTC

- A CBTC system **shall include** an ATP (Automatic Train Protection) system and an ATS (Automatic Train Supervision) system. The ATS system **may be** of two types: ATS Simple, and ATS Router.
- An IXL system may include an interlocking (IXL) system, which can be either an IXL Pure or an IXL Controllable.
- The ATP system **shall be composed** of an ATP Onboard system and an ATP Wayside system. The former is placed on board of every train. The latter is placed along the line.
- The ATP Wayside system can be of three types: ATP Wayside IXL, if it embeds an IXL; ATP Wayside Controller, which **requires** an IXL Controllable; ATP Wayside Simple, which **requires** an IXL Pure and an ATS Router to work properly.
- An ATP IXL does not exclude the usage of an additional independent IXL, normally employed for redundancy.

A real system: Communication Based Train Control



Static & behavioural requirements of product families

Static requirements identify the **features** constituting different products and *behavioural requirements* the **admitted sequences of operations**

Static requirements of product families

- The only accepted coins are the 1 euro coin (1€), exclusively for the European products and the 1 dollar coin (1\$), exclusively for the US products (1€ and 1\$ are exclusive (**alternative**) features)
- A cappuccino is only offered by European products (**excludes** relation between features)

Behavioural requirements of product families

- After inserting a coin, the user has to choose whether or not (s)he wants sugar, by pressing one of two buttons, after which (s)he may select a beverage
- The machine returns to its idle state when the beverage is taken

- Deontic logic provides a natural way to formalize concepts like violation, obligation, permission and prohibition
- Deontic logic seems to be very useful to formalize product families specifications, since they allow one to capture the notions of **optional**, **mandatory** and **alternative** features
- Deontic logic seems to be very useful to formalize feature constraints such as **requires** and **excludes**.

⇒ Deontic logic seems to be a natural candidate for expressing the conformance of products with respect to variability rules

A deontic logic consists of the standard operators of propositional logic, i.e. negation (\neg), conjunction (\wedge), disjunction (\vee) and implication (\implies), augmented with deontic operators (O and P in our case)

The most classic deontic operators, namely *it is obligatory that* (O) and *it is permitted that* (P) enjoy the duality property

Informal meaning of the deontic operators

- $O(a)$: action a is *obligatory*
- $P(a) = \neg O(\neg a)$: action a is *permitted*
if and only if its negation is not obligatory

Construction of deontic characterization of FM

- If A is a feature and A_1 and A_2 are subfeatures, add the formula:

$$A \implies \Phi(A_1, A_2), \quad \text{where } \Phi(A_1, A_2) \text{ is defined as:}$$

$\Phi(A_1, A_2) = (O(A_1) \vee O(A_2)) \wedge \neg(P(A_1) \wedge P(A_2))$ if A_1, A_2 **alternative**,
and otherwise:

$\Phi(A_1, A_2) = \phi(A_1) \wedge \phi(A_2)$, in which A_i , for $i \in \{1, 2\}$, is defined as:

$$\phi(A_i) = \begin{cases} P(A_i) & \text{if } A_i \text{ is } \mathbf{optional} \\ O(A_i) & \text{if } A_i \text{ is } \mathbf{mandatory} \end{cases}$$

- If A **requires** B , add the formula $A \implies O(B)$

- If A **excludes** B , add the formula $(A \implies \neg P(B)) \wedge (B \implies \neg P(A))$

Expressing feature models with deontic logic

Characteristic formula of Coffee machine family

$$O(\text{Coin}) \wedge O(\text{Beverage}) \wedge P(\text{Ringtone})$$
$$\wedge$$
$$\text{Coin} \implies (O(1\$) \vee O(1\text{€})) \wedge \neg(P(1\$) \wedge P(1\text{€}))$$
$$\text{Beverage} \implies O(\text{Coffee}) \wedge P(\text{Tea}) \wedge P(\text{Cappuccino})$$
$$\wedge$$
$$\text{Cappuccino} \implies O(\text{Ringtone})$$
$$(1\$ \implies \neg P(\text{Cappuccino})) \wedge (\text{Cappuccino} \implies \neg P(1\$))$$

Two example coffee machines

$$\text{CM1} = \{\text{Coin}, 1\text{€}, \text{Beverage}, \text{Coffee}\}$$
$$\text{CM2} = \{\text{Coin}, 1\text{€}, \text{Beverage}, \text{Coffee}, \text{Cappuccino}\}$$

CM1 in family, but CM2 not: $\text{Cappuccino} \implies O(\text{Ringtone})$ false

Running example: Coffee machine family-SPLIT

 Your model has been exported to SXFM format! [Click here to open a new page](#) containing your saved model.

 All information will be lost if you exit this page. Hence, make sure you export or save your model regularly.

Feature Diagram



Feature Information Table

Id:

Name:

Description:

Type:

#Children:

Tree level:

Cross-Tree Constraints

(ringtone \vee \neg cappuccino)

(\neg dollar \vee \neg cappuccino)

[Click to create a constraint](#)

Additional Information

Name your feature model: (*)

Short description of feature model: (*)

Primary Author: (*)

Author's Address:

Author's Email: (*)

Author's Phone Number:

Author's Website:

Author's Organization: (*)

Author's Organization Department:

Date model was created:

Where was model published? (if applicable):

(*) Mandatory fields if you wish to add your model to SPLIT's feature model repository

Feature Model Statistics

#Features	8
#Mandatory	2
#Optional	3
#XOR groups	1
#OR groups	0
#Grouped	2
#Cross-Tree Constraints (CTC)	2
CTCR (%)	0.38
#CTC distinct vars	3
CTC clause density	0.67

Feature Model Analysis

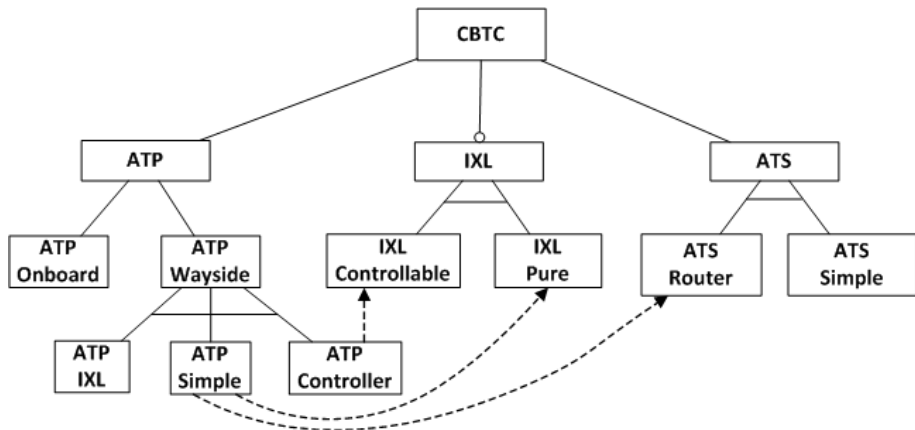
<input checked="" type="checkbox"/> Consistency	Consistent
<input checked="" type="checkbox"/> Dead Features	None
<input checked="" type="checkbox"/> Core Features	4 feature(s)
<input checked="" type="checkbox"/> Valid Configurations	10

Run Analysis every

Static requirements of CBTC

- A CBTC system **shall include** an ATP (Automatic Train Protection) system and an ATS (Automatic Train Supervision) system. The ATS system **may be** of two types: ATS Simple, and ATS Router.
- An IXL system may include an interlocking (IXL) system, which can be either an IXL Pure or an IXL Controllable.
- The ATP system **shall be composed** of an ATP Onboard system and an ATP Wayside system. The former is placed on board of every train. The latter is placed along the line.
- The ATP Wayside system can be of three types: ATP Wayside IXL, if it embeds an IXL; ATP Wayside Controller, which **requires** an IXL Controllable; ATP Wayside Simple, which **requires** an IXL Pure and an ATS Router to work properly.
- An ATP IXL does not exclude the usage of an additional independent IXL, normally employed for redundancy.

A real system: Communication Based Train Control



Behavioural modelling

- We are now interested to the dynamic behaviour: giving a behavioural model of a family of products requires the ability to say which behavioural elements (states, transitions, actions, messages,...) are required in a product and which are optional.
- Common solution: annotating behavioural descriptions of the family with aside notations that describe variation points, and which parts of the behaviour are variable, that is, are optional or required for the different products of a family. (*e.g. annotated UML state diagrams or sequence diagrams*).
- Such annotations often lack a formal definition \Rightarrow no uniform framework in which to reason formally on families .

... our final aim is to be able to apply formal verification techniques to a family definition, so that what is proved for a family is consequently proved for a product of a family

First product of the family: an audible coffee and cappuccino machine for the European market



Second product of the family: A silent tea and coffee machine for the Canadian market

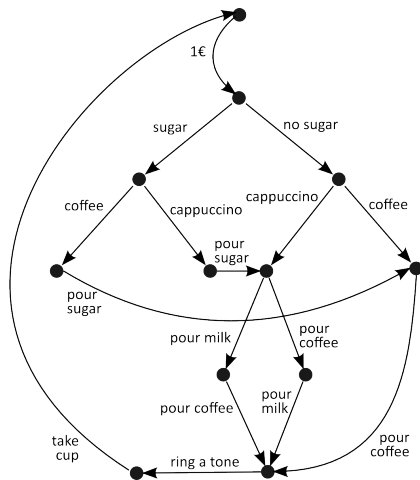


Quest for an expressive modelling formalism for families based on the choice of a fundamental model, equipped with formal tools to reason over formal specifications. Labeled Transition Systems (LTS) are one of the most common formal frameworks for modeling and reasoning over the behaviour of a system.

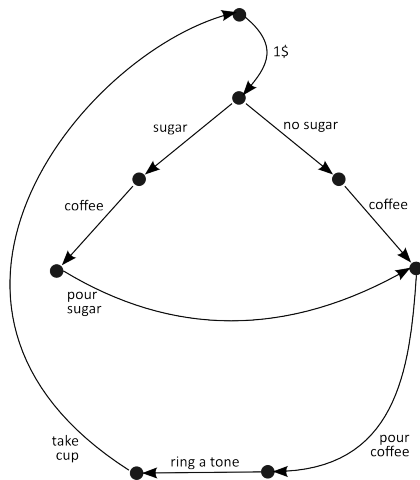
Definition

A *Labeled Transition System* (LTS) is a quadruple: $(S, Act, s_0, \rightarrow)$, where S is a set of states, Act is a set of actions used as transition labels, $s_0 \in S$ is the initial state, and $\rightarrow \subseteq S \times Act \times S$ is the transition relation. If $(s, t, s') \in \rightarrow$, we write $s \xrightarrow{t} s'$.

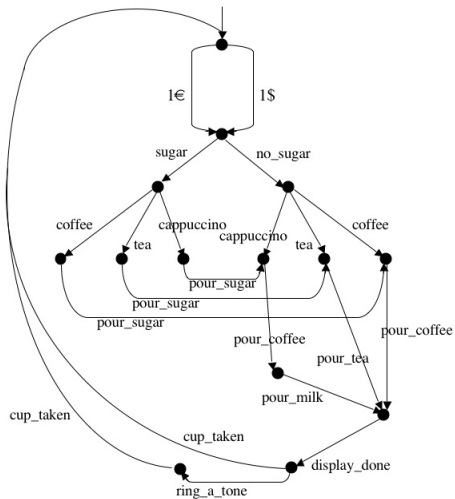
LTS model of the EU product



LTS model of the Canadian product



Attempt to model the family with a LTS



Deriving a product from a family

Given a LTS T , visit the graph of T in a breadth-first fashion, applying at each state s the step $\text{visit}(s)$, as defined below:

$\text{visit}(s)$:

```
select-transitions(s);           USER SELECTION
while( removable transitions exist )
  { if a transition  $t$  is removable:
    remove  $t$  from the graph
    if the state  $s'$ , target of the transition  $t$ ,
    is no more reachable by any other state:
      remove the state  $s'$ ;
      mark all its outgoing transitions
      as "removable";
  }
```

$\text{select-transitions}(s)$:

request to the user, for each outgoing transition t from s , to tell whether he/she considers the transition t present in the derived product. If not, the transition is marked as *removable*.

The choice needs to be guided by the family requirements, but this is not enforced by the algorithm, since no knowledge about variation points is embedded in the model.

Any LTS associated to a product derived by the previous algorithm has a *simulation* relation with the LTS representing the family.

Definition

Let $T_1 = (\mathcal{S}_1, Act, s_{0_1}, \rightarrow_1)$ and $T_2 = (\mathcal{S}_2, Act, s_{0_1}, \rightarrow_2)$ We say that $s_2 \in \mathcal{S}_2$ *simulates* $s_1 \in \mathcal{S}_1$ (written $s_1 \preceq_s s_2$) if there exists a *strong simulation* that relates s_1 and s_2 .

$\mathcal{R} \subseteq \mathcal{S}_1 \times \mathcal{S}_2$ is a strong simulation if $\forall (s_1, s_2) \in \mathcal{R}$ (where $\sigma \in T_1 \cup T_2$),

- $s_1 \xrightarrow{\sigma}_1 s'_1$ implies $\exists s'_2 : s_2 \xrightarrow{\sigma}_2 s'_2$ and $(s'_1, s'_2) \in \mathcal{R}$.

The above definition is naturally extended to LTSs by considering their initial states: A LTS T_2 simulates T_1 (written $T_1 \preceq_s T_2$) iff $(s_{0_1} \preceq_s s_{0_1})$

The family LTS simulates the product LTS

Inadequacy of the LTS modeling of a family

In the coffee machine example,

- a product that accepts both euro and dollar coins,
- or a product that does not allow a user to ask for sugar,

can both be derived by the previous algorithm.

It is easy to notice that these products do not satisfy the given family requirements

Modal Transition Systems (MTS)

Modal Transition Systems (MTS)

[K. Larsen and B. Thomsen, *A Modal Process Logic*. LICS88]

have been proposed to capture variability:

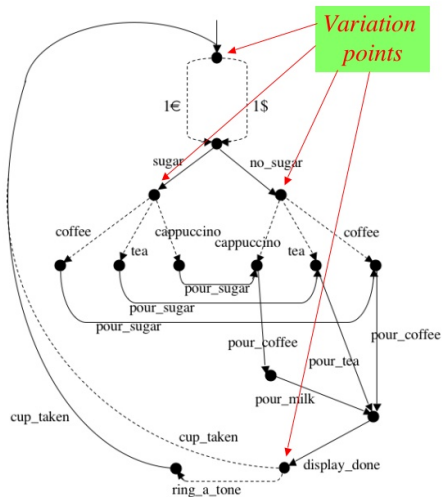
[D. Fischbein, S. Uchitel, Sebastian, V. Braberman, *A Foundation for Behavioural Conformance in Software Product Line Architectures*, 2nd ROSATEA Workshop, 2006]

In a MTS, transitions may be *possible* or *required*, that is, *optional* or *mandatory* for a product.

Definition

A MTS $F = (S, Act, s_0, \rightarrow_{\square}, \rightarrow_{\diamond})$ is defined as a LTS, having two distinct transition relations, namely $\rightarrow_{\square} \subseteq S_F \times Act \times S_F$ is the *must* transition relation, which expresses *required* transitions, $\rightarrow_{\diamond} \subseteq S_F \times Act \times S_F$ is the *may* transition relation, which expresses *possible* transitions.

Model of a family as a MTS



Conformance relation

The “*is a product of F*” relation below, called also “conformance” relation, links a MTS representing a family with a LTS representing a product.

Definition

We say that P is a product of F , written $P \vdash F$, if and only if $p_0 \vdash s_0$, where:

$p \vdash f$ if and only if

- $f \xrightarrow{a}_{\square} f' \implies \exists p' \in S_P : p \xrightarrow{a} p' \text{ and } p' \vdash f'$
- $p \xrightarrow{a} p' \implies \exists f' \in S_F : f \xrightarrow{a}_{\diamond} f' \text{ and } p' \vdash f'$

The conformance relation implies simulation.

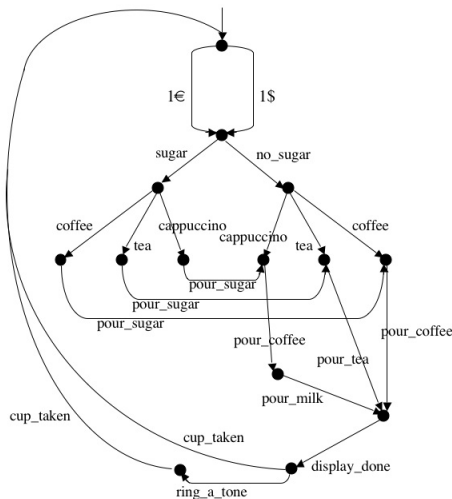
Considering our example, it is again possible to derive products that do not satisfy the requirements:

- A product that accepts both euro and dollar coins can be derived.

This is because MTSs are not completely adequate to model the case of alternative variability (i.e. the mandatory selection of at least one among several different choices).

Inadequacy of the MTS modeling of a family -Cont.

Again the following LTS (product) may be derived:



Definition

An *Extended Modal Transition System* (EMTS) is a quintuple $(S, Act, s_0, \square, \diamond)$, where S is a set of states, Act is a set of actions, $s_0 \in S$ is the initial state, $\square \subseteq S \times 2^{Act \times S}$ is the *at least 1-of- n* transition relation, and $\diamond \subseteq S \times 2^{Act \times S}$ is the *at most 1-of- n* transition relation.

We write respectively:

$s \xrightarrow{a_1, a_2, \dots, a_n} \square s_1, s_2 \dots, s_n$ and

$s \xrightarrow{a_1, a_2, \dots, a_n} \diamond s_1, s_2 \dots, s_n$

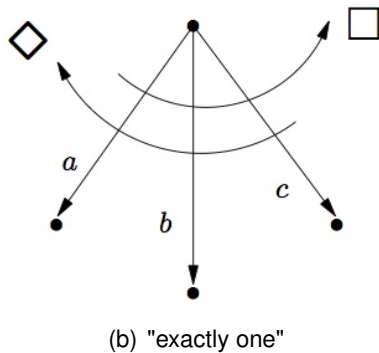
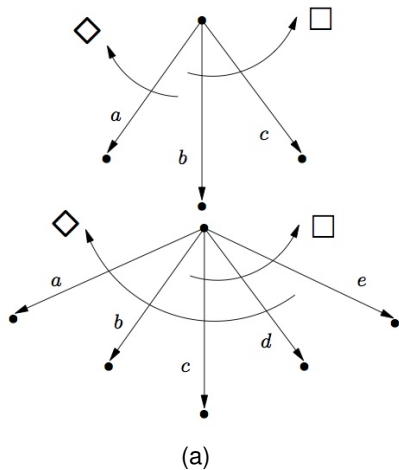
to denote elements of the two relations:

in the first case any product of the family should have at least one of the n transitions $s \xrightarrow{a_j} s_j$,

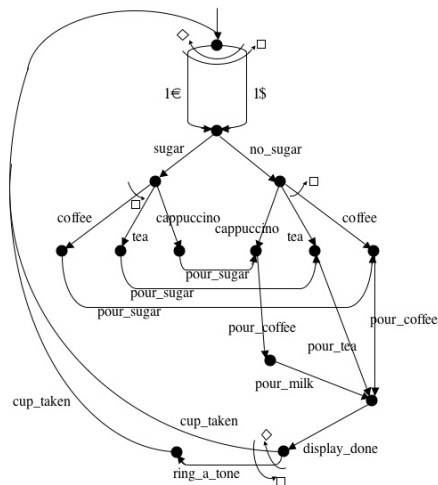
in the second case any product of the family should have at most 1 of the n transitions (that is, it can also have no transition from this set).

the number of the actions on the arrow must coincide with that of target states, and order counts as well, since each action is paired to the corresponding state

Example EMTSs

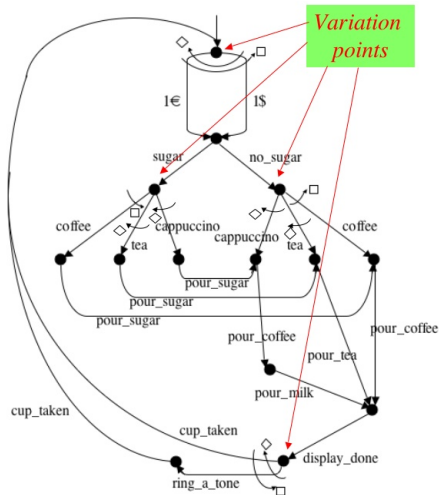


Modelling a family with an EMTS ¹



¹to avoid notation overloading modality symbols are not drawn for compulsory transitions (at least 1 of 1)

Modelling a family with an EMTS ²



²to avoid notation overloading modality symbols are not drawn for compulsory transitions (at least 1 of 1)

The above LTS with gives the choice between Dollar and Euro may not be derived now.

The two LTSs representing the Canadian and the European machine are instead correctly derived.

Inadequacy to model multiple optionality

A more general notion is actually needed if we want to model multiple optionality, that is, the fact that a product may have (at least, at most, exactly) k of the n choices proposed by the family.

Examples of multiple optionality:

energy consumption or budget considerations may give an upper bound to the number of provided features, while marketing strategies may suggest a lower bound, under which the product loses its market.

Upper and lower bounds may be used to define subfamilies on the basis of such non functional aspects.

Definition

A *Generalized Extended Modal Transition System* (GEMTS) is a quintuple $(S, Act, s_0, \square, \diamond)$, where S is a set of states, Act is a set of actions, $s_0 \in S$ is the initial state, $\square \subseteq S \times 2^{Act \times S} \times N$ is the *at least k -of- n* transition relation, and $\diamond \subseteq S \times 2^{Act \times S} \times N$ is the *at most k -of- n* transition relation.

We write respectively:

$s \xrightarrow{a_1, a_2, \dots, a_n} \square_k s_1, s_2, \dots, s_n$ and

$s \xrightarrow{a_1, a_2, \dots, a_n} \diamond_k s_1, s_2, \dots, s_n$

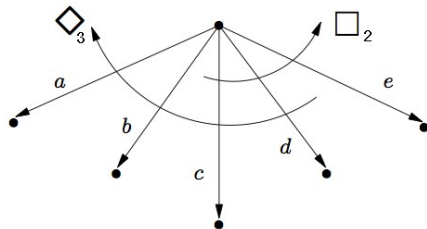
to denote elements of the two relations:

in the first case any product of the family should have at least k of the n transitions $s \xrightarrow{a_i} s_i$,

in the second case any product of the family should have at most k of the n transitions (that is, it can also have no transition from this set).

the number of the actions on the arrow must coincide with that of target states, and order counts as well, since each action is paired to the corresponding state.

An example of GEMTS



Possible transitions sets:

$\{c, d\}$ $\{c, d, e\}$ $\{a, c, d\}$ $\{b, c, d\}$ $\{c, e\}$ $\{a, c, e\}$ $\{b, c, e\}$

$\{a, b, c, e\}$ $\{d, e\}$ $\{a, d, e\}$ $\{b, d, e\}$ $\{a, b, d, e\}$ $\{a, c, d, e\}$ $\{b, c, d, e\}$

A GEMTS defines a family of LTSs according to the following definition:

Definition

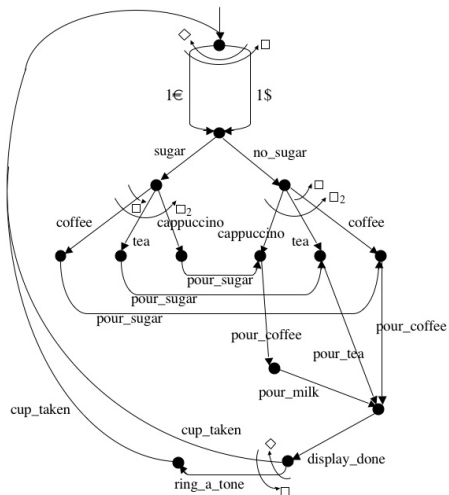
A LTS $P = (S_P, Act, \rightarrow, p_0)$ belongs to the family (GEMTS) $F = (S_F, Act, f_0, \square, \diamond)$ (we say also P is a product of F , or P conforms to F , written $P \vdash F$) if and only if $p_0 \vdash f_0$, where:

$p \vdash f$ if and only if

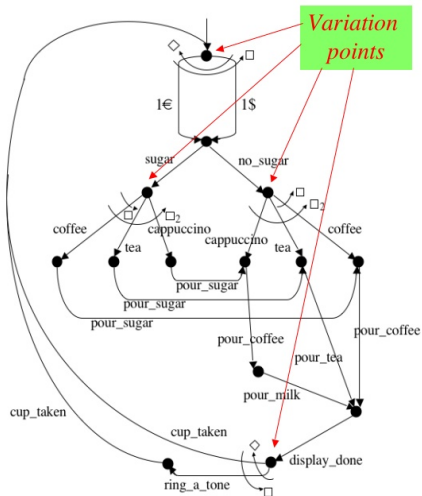
- $f \xrightarrow{a_1, a_2, \dots, a_n}_{\square_k} f_1, f_2, \dots, f_n \implies \exists I \subseteq \{1, \dots, n\}, 1 \leq |I| \leq k : \forall i \in I, p \xrightarrow{a_i} p_i \text{ and } p_i \vdash f_i$
- $f \xrightarrow{a_1, a_2, \dots, a_n}_{\diamond_k} f_1, f_2, \dots, f_n \implies \nexists I \subseteq \{1, \dots, n\}, k < |I| \leq n : \forall i \in I, p \xrightarrow{a_i} p_i \text{ and } p_i \vdash f_i$
- $p \xrightarrow{a} p' \implies \exists k, A \subseteq Act, S \subseteq S_F, f' \in S_F : (a, f') \in A \times S_F, (f, A \times S, k) \in \square \text{ or } (f, A \times S, k) \in \diamond, \text{ and } p' \vdash f'$

- the **select-transitions(s)** procedure constrains the selection of the user by enforcing him/her to respect the *at least k-of-n* and *at most k-of-n* relations applicable on the transitions outgoing from state s .
- Hence, the set of products derivable are, again, a subset of those derivable by the family LTS
- Therefore, any derivable product is still simulated by the LTS family.

Modelling a family with a GEMTS



Modelling a family with a GEMTS



Other variants of LTSs/MTSs were proposed for family modeling:

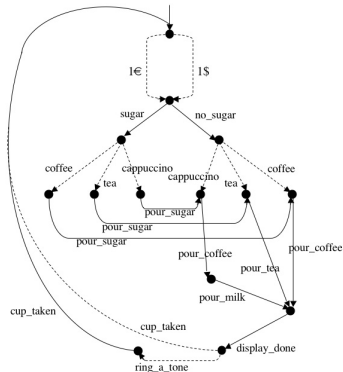
- 1-selecting MTSs (1-MTS):
H. Schmidt, H. Fecher, Comparing Disjunctive Modal Transition Systems with an One-Selecting Variant, NWPT'06
- which are a variant of Disjunctive Modal Transition Systems (DMTS)
K.G. Larsen, L. Xinxin, Equation solving using modal transition systems, LICS 1990

Meaning of modalities

modality	MTS	DMTS	1-MTS	EMTS	GEMTS
◇ (may)	at most 1-of-1	at most n-of-n	at most 1-of-n	at most 1-of-n	at most k-of-n
□ (must)	at least 1-of-1	at least 1-of-n	exactly 1-of-n	at least 1-of-n	at least k-of-n

GEMTSs show the greater generality

MTS and Constraints



Constraints :

- euro ALT dollar
- cappuccino REQ ring-a-tone
- dollar EXC cappuccino
- ring-a-tone ALT no-ring

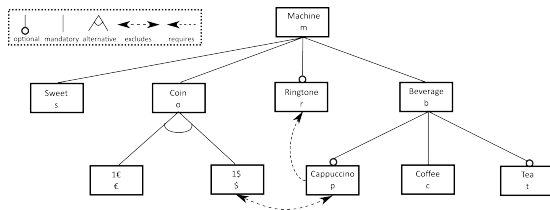
The idea will be to prune optional (may) transitions in the MTS in a counterexample-guided way, i.e. based on model-checking techniques

Definition (FTS)

A *Featured Transition System* (FTS) is a 9-tuple $(Q, A, \bar{q}, \delta, AP, L, FD, \gamma, >)$, with *underlying* L^2TS $(Q, A, \bar{q}, \delta, AP, L)$, feature diagram FD over a set \mathbb{F} of features, total function $\gamma : \delta \rightarrow \mathbb{F}$ labelling transitions with features, and partial order $> \subseteq \delta \times \delta$ defining an ordering among transitions.

A transition of an FTS is thus labelled with an action *and* a feature and multiple outgoing transitions may be ordered.

[Classen, A., Heymans, P., Schobbens, P.-Y., Legay, A., Raskin, J.-F.: **Model checking lots of systems: Efficient verification of temporal properties in software product lines**. ICSE 2010. ACM, New York (2010)]



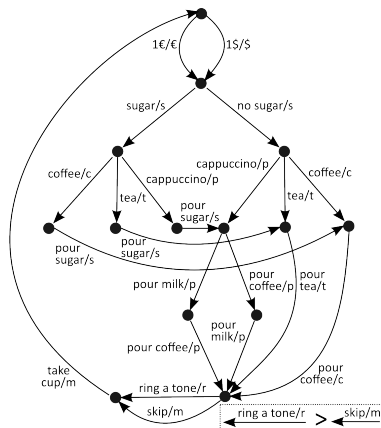
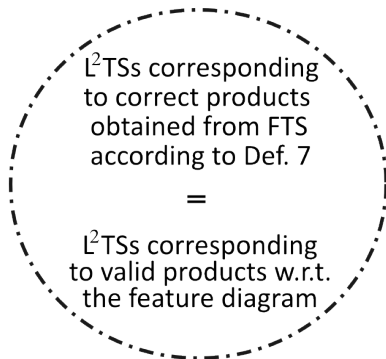
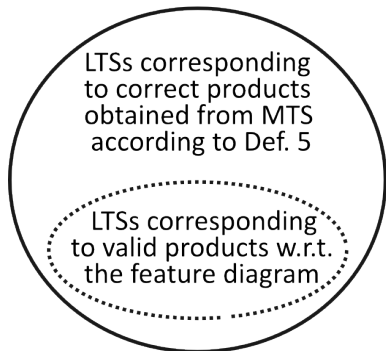


Figure: FTS modelling the family of coffee machines.

Definition (FTS product derivation)

Given an FTS $\mathcal{F} = (Q, A, \bar{q}, \delta, AP, L, FD, \gamma, >)$ that specifies a family, a set of products specified as a set of L²TSs $\{ \mathcal{P}_p = (Q_p, A_p, \bar{q}, \delta_p, AP_p, L_p) \mid p \subseteq 2^{\mathbb{F}} \}$ may be derived by considering each transition relation δ_p to be obtained from δ by pruning (i) all transitions labelled with features not in p and (ii) all transitions overridden by transitions preceding them in the partial order.

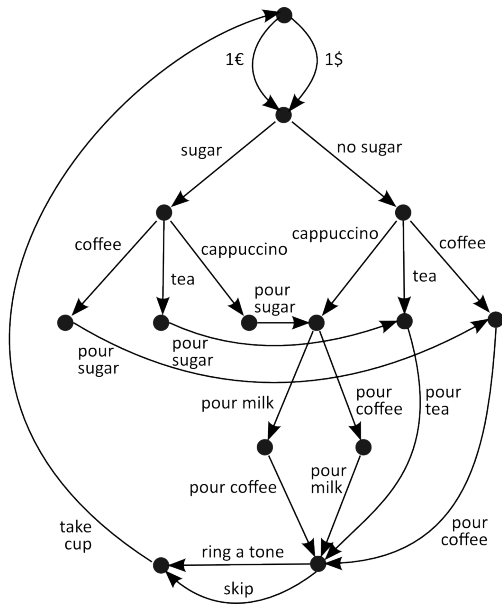
Product derivation: MTS versus FTS



FTS All and only products that are correct w.r.t. the requirements are derived (price: include a feature diagram in each FTS)

MTS Also correctly derived products may violate constraints of the type that MTSs cannot model (cf. LTS on next slide)

A correct but not valid product LTS of MTS



Action-and State-based Logics

Model checking primarily evolved over Kripke structures

Information associated to states (e.g. SMV, NuSMV, etc.)

Theory of concurrency evolved over Labelled Transition Systems

Information associated to transitions (CCS, LOTOS, π -calculus, etc.)

Many aspects of software (in particular the most interesting ones for reactive, concurrent and distributed software) are events, i.e. actions

- Ease of expressiveness w.r.t. pure action- or state-based logics
- Effective gain depends on specific system under scrutiny

HML is a branching time temporal logic interpreted over LTSs.

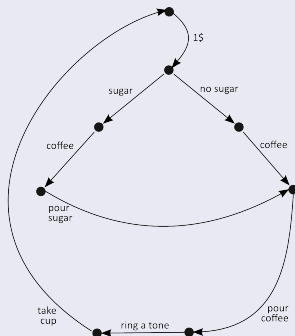
Definition

$$\phi ::= tt \mid \neg\phi \mid \phi \wedge \phi' \mid [a]\phi \mid \langle a \rangle\phi \mid E\pi \mid A\pi$$

where the informal meaning of the above operators is:

- $[a]\phi$: for all next states reachable with a , ϕ holds;
- $\langle a \rangle\phi$: a next state exists reachable with a where ϕ holds;
- $E\pi$: there exists a path on which π holds;
- $A\pi$: on each of the possible paths π holds;

LTS model of the Canadian product



The following properties may be checked over it:

$[1\$] \langle \textit{sugar} \rangle \langle \textit{coffee} \rangle \textit{true} \longrightarrow \text{TRUE}$

$\langle 1\$ \langle \textit{sugar} \rangle \langle \textit{coffee} \rangle \textit{true} \longrightarrow \text{TRUE}$

$\langle 1\$ \langle \langle \textit{coffee} \rangle \textit{true} \longrightarrow \text{FALSE}$

$[\textit{sugar}] \langle \textit{coffee} \rangle \textit{true} \longrightarrow \text{????}$

Definition of HML + UNTIL

HML has a too low expressive power to express interesting properties like Safety and liveness properties the following extension was defined:

Definition

$$\begin{aligned}\phi &::= tt \mid \neg\phi \mid \phi \wedge \phi' \mid [a]\phi \mid \langle a \rangle\phi \mid E\pi \mid A\pi \\ \pi &::= \phi \mathbf{U} \phi'\end{aligned}$$

where the informal meaning of the above operators is:

- $[a]\phi$: for all next states reachable with a , ϕ holds;
- $\langle a \rangle\phi$: a next state exists reachable with a where ϕ holds;
- $E\pi$: there exists a path on which π holds;
- $A\pi$: on each of the possible paths π holds;
- $\phi \mathbf{U} \phi'$: in the current or a future state ϕ' holds, while ϕ holds until that state (but not necessarily in that state).

Definition of MHML

Variability and action-based branching-time temporal logic

A temporal logic based on the “Hennessy-Milner logic with until”, but augmented with deontic O (*obligatory*) and P (*permitted*) operators, CTL’s path operators E and A and ACTL’s action-based Until operator, both with and without a deontic interpretation

Syntax of MHML

$$\begin{aligned}\phi &::= \text{true} \mid \neg\phi \mid \phi \wedge \phi' \mid \langle a \rangle \phi \mid [a] \phi \mid \langle a \rangle^\square \phi \mid [a]^\square \phi \mid E \pi \mid A \pi \\ \pi &::= \phi \{ \varphi \} U \{ \varphi' \} \phi' \mid \phi \{ \varphi \} U^\square \{ \varphi' \} \phi'\end{aligned}$$

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

$\langle a \rangle^\square$ and $[a]^\square$ represent the classic deontic modalities O and P , resp.

The intuitive interpretation of the nonstandard operators is:

- $\langle a \rangle \phi$: a next state exists, reachable by a *must* transition executing action a , in which ϕ holds
- $[a] \phi$: in all next states, reachable by a *may* or *must* transition executing action a , ϕ holds
- $E \pi$: there exists a full path on which π holds
- $A \pi$: on all possible full paths, π holds
- $\phi \{ \varphi \} U \{ \varphi' \} \phi'$: in a future state reached by an action satisfying φ' , ϕ' holds, while ϕ holds from the current state until that state is reached and all actions executed in the meanwhile along the path satisfy φ
- $\phi \{ \varphi \} U^\square \{ \varphi' \} \phi'$: in a future state reached by an action satisfying φ' , ϕ' holds, while ϕ holds from the current state until that state is reached and the path leading to that state is a *must* path such that all actions executed in the meanwhile along the path satisfy φ

To give the semantics of MHML the notion of path and must path should be given:

Definition

Let $(Q, A, \bar{q}, \rightarrow_{\square}, \rightarrow_{\diamond})$ be an MTS and let $\sigma = q_1 a_1 q_2 a_2 q_3 \dots$ be a full path in its underlying LTS. Then σ is a *must path* (from q_1) if

$q_i \xrightarrow{a_i}_{\square} q_{i+1}$, for all $i > 0$, in the MTS.

The set of all must paths from q_1 is denoted by $\square\text{-path}(q_1)$. A must path σ is denoted by σ^{\square} .

MHML : Semantics over MTS

- $q \models \text{true}$ always holds
- $q \models \neg \phi$ iff not $q \models \phi$
- $q \models \phi \wedge \phi'$ iff $q \models \phi$ and $q \models \phi'$
- $q \models \langle a \rangle \phi$ iff $\exists q' \in Q$ such that $q \xrightarrow{a}_{\diamond} q'$, and $q' \models \phi$
- $q \models [a] \phi$ iff $\forall q' \in Q$ such that $q \xrightarrow{a}_{\diamond} q'$, we have $q' \models \phi$
- $q \models \langle a \rangle^{\square} \phi$ iff $\exists q' \in Q$ such that $q \xrightarrow{a}_{\square} q'$, and $q' \models \phi$
- $q \models [a]^{\square} \phi$ iff $\forall q' \in Q$ such that $q \xrightarrow{a}_{\square} q'$, we have $q' \models \phi$
- $q \models E\pi$ iff $\exists \sigma' \in \text{path}(q)$ such that $\sigma' \models \pi$
- $q \models A\pi$ iff $\forall \sigma' \in \text{path}(q)$ such that $\sigma' \models \pi$
- $\sigma \models \phi \{ \varphi \} U \{ \varphi' \} \phi'$ iff $\exists j \geq 1$: $\sigma(j) \models \phi'$, $\sigma\{j\} \models \varphi'$, and $\sigma(j+1) \models \phi'$, and $\forall 1 \leq i < j$: $\sigma(i) \models \phi$ and $\sigma\{i\} \models \varphi$
- $\sigma \models \phi \{ \varphi \} U^{\square} \{ \varphi' \} \phi'$ iff σ is a must path σ^{\square} and $\sigma^{\square} \models \phi \{ \varphi \} U \{ \varphi' \} \phi'$

Abbreviations: $EF\phi = E(\text{true} \{ \text{true} \} U \{ \text{true} \} \phi)$; $EF^{\square}\phi = E(\text{true} \{ \text{true} \} U^{\square} \{ \text{true} \} \phi)$;
 $EF\{\varphi\} \text{true} = E(\text{true} \{ \text{true} \} U \{ \varphi \} \text{true})$; $EF^{\square}\{\varphi\} \text{true} = E(\text{true} \{ \text{true} \} U^{\square} \{ \varphi \} \text{true})$;
 $AG\phi = \neg EF\neg\phi$; $AG^{\square}\phi = \neg EF^{\square}\neg\phi$; etc.

MHML : Semantics over MTS

- $q \models \text{true}$ always holds
- $q \models \neg \phi$ iff not $q \models \phi$
- $q \models \phi \wedge \phi'$ iff $q \models \phi$ and $q \models \phi'$
- $q \models \langle a \rangle \phi$ iff $\exists q' \in Q$ such that $q \xrightarrow{a}_{\diamond} q'$, and $q' \models \phi$
- $q \models [a] \phi$ iff $\forall q' \in Q$ such that $q \xrightarrow{a}_{\diamond} q'$, we have $q' \models \phi$
- $q \models \langle a \rangle^{\square} \phi$ iff $\exists q' \in Q$ such that $q \xrightarrow{a}_{\square} q'$, and $q' \models \phi$
- $q \models [a]^{\square} \phi$ iff $\forall q' \in Q$ such that $q \xrightarrow{a}_{\square} q'$, we have $q' \models \phi$
- $q \models E\pi$ iff $\exists \sigma' \in \text{path}(q)$ such that $\sigma' \models \pi$
- $q \models A\pi$ iff $\forall \sigma' \in \text{path}(q)$ such that $\sigma' \models \pi$
- $\sigma \models \phi \{ \varphi \} U \{ \varphi' \} \phi'$ iff $\exists j \geq 1$: $\sigma(j) \models \phi'$, $\sigma\{j\} \models \varphi'$, and $\sigma(j+1) \models \phi'$, and $\forall 1 \leq i < j$: $\sigma(i) \models \phi$ and $\sigma\{i\} \models \varphi$
- $\sigma \models \phi \{ \varphi \} U^{\square} \{ \varphi' \} \phi'$ iff σ is a must path σ^{\square} and $\sigma^{\square} \models \phi \{ \varphi \} U \{ \varphi' \} \phi'$

Abbreviations: $EF\phi = E(\text{true} \{ \text{true} \} U \{ \text{true} \} \phi)$; $EF^{\square}\phi = E(\text{true} \{ \text{true} \} U^{\square} \{ \text{true} \} \phi)$;
 $EF\{\varphi\} \text{true} = E(\text{true} \{ \text{true} \} U \{ \varphi \} \text{true})$; $EF^{\square}\{\varphi\} \text{true} = E(\text{true} \{ \text{true} \} U^{\square} \{ \varphi \} \text{true})$;
 $AG\phi = \neg EF\neg\phi$; $AG^{\square}\phi = \neg EF^{\square}\neg\phi$; etc.

Advanced variability management

MHML can complement behavioral description of MTS by expressing constraints over possible products of a family that MTS cannot model

Template ALT: Features F1 and F2 are *alternative*

$$(EF^{\square} \{F1\} true \vee EF^{\square} \{F2\} true) \wedge \neg(EF \{F1\} true \wedge EF \{F2\} true)$$

Template EXC: Feature F1 *excludes* feature F2

$$((EF \{F1\} true) \implies (AG \neg \langle F2 \rangle true)) \wedge ((EF \{F2\} true) \implies (AG \neg \langle F1 \rangle true))$$

Template REQ: Feature F1 *requires* feature F2

$$(EF \{F1\} true) \implies (EF^{\square} \{F2\} true)$$

Define no full temporal ordering among the related features

Duty of behavioral LTS/MTS model, to be verified by MHML formulae

Verify property expressed as logical formula ψ over model T

- Decide whether $T \models \psi$, where \models is the logic's satisfaction relation
- If $T \not\models \psi$, then it is usually easy to generate a counterexample
- If T is finite, model checking thus reduces to a graph search

On the fly: Only a fragment of the overall state space might need to be generated and analysed to be able to produce the correct result. Model checking MHML formulae over MTSs can be achieved in a complexity that is **linear** w.r.t. the state space size.

Model checking temporal orderings

Property D: Always once a coffee has been selected, a coffee is eventually delivered

$$AG [coffee] AF^{\square} \{pour\ coffee\} \text{ true}$$

Property E: A coffee machine may never deliver a coffee before a coin has been inserted

$$A [true \{ \neg \text{pour coffee} \} U^{\square} \{1\$ \vee 1\text{€}\} \text{ true}]$$

Property	MTS family	European	Canadian	LTS product
A: ALT	<i>false</i>	<i>true</i>	<i>true</i>	<i>false</i>
B: EXC	<i>false</i>	<i>true</i>	<i>true</i>	<i>false</i>
C: REQ	<i>false</i>	<i>true</i>	<i>true</i>	<i>true</i>
D	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>
E	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>

Preservation of validity from families to products

A sufficient condition for the preservation of MHML properties from a product family to its products is that these properties can be expressed as formulae in the fragment MHML_{\square} of MHML:

Definition

$$\phi ::= \text{true} \mid \phi \wedge \phi' \mid \phi \vee \phi' \mid \langle a \rangle \phi \mid [a] \phi \mid EF^{\square} \{\varphi\} \mid AF^{\square} \{\varphi\} \mid$$
$$E(\phi \{\varphi\} U^{\square} \{\varphi\} \phi') \mid A(\phi \{\varphi\} U^{\square} \{\varphi\} \phi') \mid EG^{\square} \phi \mid AG^{\square} \phi \mid AG\phi$$

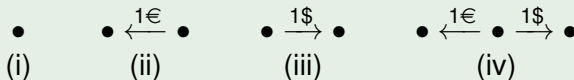
A formula of the form $[a] \phi$ is a universal formula which is preserved on the same basis as that of the classical result of preservation by simulation for CTL.

A formula of the form $\langle a \rangle \phi$ is preserved as it dictates the existence of a must path with certain characteristics will necessarily be found in all products.

The underlying idea is to construct intermediate LTSs (partially covering the original MTS to a given depth) in a step-by-step fashion as follows.

- First unfold the initial MTS according to the type of transitions, meanwhile verifying formulae of type ALT and EXC, and continue only with the LTSs satisfying all formulae.
- In the end, when no more unfolding step can be applied to the LTSs, verify formulae of type REQ and (partially) ALT over these LTSs.

A first unfolding based on its type of transitions results in:



Next, the algorithm would consider the relevant MHTML formula of type ALT (Property A: 1€ and 1\$ are *alternative*):

$$(EF \langle 1\$ \rangle true \vee EF \langle 1\text{€} \rangle true) \wedge \\ \neg(EF P(1\$) true \wedge EF P(1\text{€}) true)$$

Verifying this formula over the above intermediate LTSs implies that the algorithm would only continue to unfold the original MTS from the leaf states of the above intermediate LTSs (ii) and (iii) onwards. The final result of the algorithm would thus be a set of valid products, the behaviour of each guaranteeing that initially only a 1€ or a 1\$ can be inserted.

VMC: <http://fmtlab.isti.cnr.it/vmc/>

An on-the-fly model-checker for vACTL is defined as particularization of the FMC model checker for ACTL over a CCS-like input language

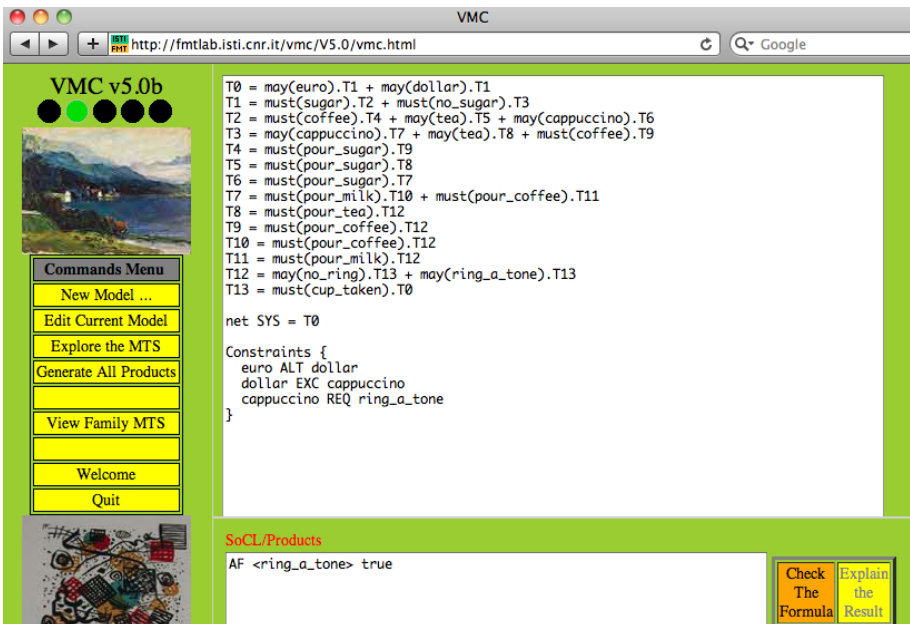
Recently implemented also our algorithm for product derivation

- Explore and verify product families (MTS)
- Generate all valid products, explore and verify products (LTS)

VMC can be used to automatically derive, from an MTS description of a product family **and** an associated set of MHML formulae expressing further constraints for this family, **all** its valid products (i.e. a set of LTS descriptions of products, each one correct w.r.t. all constraints)

VMC can also be used to experiment with products to verify further (temporal) constraints, etc.

MTS specification of coffee machine family in VMC



The screenshot shows a web browser window titled "VMC" with the URL `http://fmtlab.isti.cnr.it/vmc/V5.0/vmc.html`. The page content is divided into several sections:

- VMC v5.0b**: A header with a logo consisting of five colored circles (black, green, black, black, black).
- Commands Menu**: A vertical list of buttons: "New Model ...", "Edit Current Model", "Explore the MTS", "Generate All Products", "View Family MTS", "Welcome", and "Quit".
- MTS Specification**: A code block containing the following text:

```
T0 = may(euro).T1 + may(dollar).T1
T1 = must(sugar).T2 + must(no_sugar).T3
T2 = must(coffee).T4 + may(tea).T5 + may(cappuccino).T6
T3 = may(cappuccino).T7 + may(tea).T8 + must(coffee).T9
T4 = must(pour_sugar).T9
T5 = must(pour_sugar).T8
T6 = must(pour_sugar).T7
T7 = must(pour_milk).T10 + must(pour_coffee).T11
T8 = must(pour_tea).T12
T9 = must(pour_coffee).T12
T10 = must(pour_coffee).T12
T11 = must(pour_milk).T12
T12 = may(no_ring).T13 + may(ring_a_tone).T13
T13 = must(cup_taken).T0

net SYS = T0

Constraints {
  euro ALT dollar
  dollar EXC cappuccino
  cappuccino REQ ring_a_tone
}
```
- SoCL/Products**: A section with a text input field containing `AF <ring_a_tone> true`.
- Buttons**: Two buttons labeled "Check The Formula" and "Explain the Result" are located at the bottom right of the SoCL/Products section.

Result of 'a coffee machine always eventually rings a tone'

The screenshot shows a web browser window with the URL `http://fmtlab.isti.cnr.it/vmc/V5.0/vmc.html`. The page title is "VMC v5.0b". On the left, there is a "Commands Menu" with buttons for "New Model ...", "Edit Current Model", "Explore the MTS", "Generate All Products", "View Family MTS", "Welcome", and "Quit". Below the menu is a decorative image of a coffee machine. The main content area is titled "Evaluation of formula 'AF true' on products" and contains a list of products with their corresponding formula evaluation results:

product11.txt	Formula evaluates	FALSE
product12.txt	Formula evaluates	TRUE
product13.txt	Formula evaluates	TRUE
product20.txt	Formula evaluates	FALSE
product21.txt	Formula evaluates	TRUE
product22.txt	Formula evaluates	TRUE
product26.txt	Formula evaluates	FALSE
product27.txt	Formula evaluates	TRUE
product28.txt	Formula evaluates	TRUE
product33.txt	Formula evaluates	TRUE
product34.txt	Formula evaluates	TRUE
product39.txt	Formula evaluates	TRUE
product40.txt	Formula evaluates	TRUE
product42.txt	Formula evaluates	TRUE
product43.txt	Formula evaluates	TRUE
product44.txt	Formula evaluates	FALSE
product45.txt	Formula evaluates	TRUE
product46.txt	Formula evaluates	TRUE
product48.txt	Formula evaluates	TRUE
product49.txt	Formula evaluates	TRUE
product53.txt	Formula evaluates	FALSE

At the bottom of the page, there is a section titled "SoCL/Products" with a text input field containing the formula `AF <ring_a_tone> true`. To the right of the input field are two buttons: "Check The Formula" and "Explain the Result".

VMC v5.2



Commands Menu

New Model ...

Load Current Model

Welcome

Quit



Kandinsky 1908

```
1 T1 = may(euro).T2 + may(dollar).T2
2 T2 = sugar.T3 + no_sugar.T4
3 T3 = sugared_coffee.T5 + may(sugared_cappuccino).T6
4   + may(sugared_tea).T7
5 T4 = unsugared_coffee.T8 + may(unsugared_cappuccino).T9
6   + may(unsugared_tea).T10
7 T5 = pour_sugar.T8
8 T6 = pour_sugar.T9
9 T7 = pour_sugar.T10
10 T8 = pour_coffee.T13
11 T9 = pour_coffee.T11 + pour_milk.T12
12 T10 = pour_tea.T13
13 T11 = pour_milk.T13
14 T12 = pour_coffee.T13
15 T13 = may(ring_a_tone).T14 + may(no_ring).T14
16 T14 = take_cup.T1
17
18 net SYS = T1
19
20 Constraints {
21   euro ALT dollar
22   sugared_tea IFF unsugared_tea
23   sugared_cappuccino IFF unsugared_cappuccino
24   dollar EXC sugared_cappuccino
25   sugared_cappuccino REQ ring_a_tone
26   ring_a_tone ALT no_ring
27 }
28
29
```

VMC v5.2



Commands Menu

New Model ...

Edit Current Model

Explore the MTS

View Current Model

View Family MTS

Generate All Products

Welcome

Quit



Kandisky 1919

Valid Products of the Family

[product11-euro-ring a tone](#)

[product12-euro-no ring](#)

[product20-dollar-ring a tone](#)

[product21-dollar-no ring](#)

[product65-euro-sugared cappuccino-unsugared cappuccino-ring a tone](#)

[product77-euro-sugared tea-unsugared tea-ring a tone](#)

[product78-euro-sugared tea-unsugared tea-no ring](#)

[product89-euro-sugared cappuccino-sugared tea-unsugared cappuccino-unsugared tea-ring a tone](#)

[product95-dollar-sugared tea-unsugared tea-ring a tone](#)

[product96-dollar-sugared tea-unsugared tea-no ring](#)

Logic Formula for all Products

Check
The
Formula

Explain
the
Result

VMC v5.2



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



Kandisky 1908

Evaluation of the formula "[euro] ((EF true) and EF true)" on all family products

product11-euro-ring a tone	Formula evaluates	FALSE
product12-euro-no_ring	Formula evaluates	FALSE
product20-dollar-ring a tone	Formula evaluates	TRUE
product21-dollar-no_ring	Formula evaluates	TRUE
product65-euro-sugared cappuccino-unsugared cappuccino-ring a tone	Formula evaluates	TRUE
product77-euro-sugared tea-unsugared tea-ring a tone	Formula evaluates	FALSE
product78-euro-sugared tea-unsugared tea-no_ring	Formula evaluates	FALSE
product89-euro-sugared cappuccino-sugared tea-unsugared cappuccino-unsugared tea-ring a tone	Formula evaluates	TRUE
product95-dollar-sugared tea-unsugared tea-ring a tone	Formula evaluates	TRUE
product96-dollar-sugared tea-unsugared tea-no_ring	Formula evaluates	TRUE

Logic Formula for all Products

[euro] ((EF <sugared_cappuccino> true) and EF <unsugared_cappuccino> true)

Describe the family MTS and then use the constraints to generate the products:

- The ATS (Simple or Router) is in charge of dispatching trains from one station to another.
- An ATS Router interacts with an IXL Pure to require a root for the train that has to be dispatched. The IXL Pure monitors the status of the railway yard, and, when routing is required by the ATS Router, allows or denies the routing of the train in accordance with the railway regulations. If routing is allowed, the ATS Router sends a command to the ATP Wayside Simple, to authorize train movement. The ATP Wayside Simple interacts with the ATP Onboard to send an authorization of movement. The ATP Onboard is in charge of controlling the safe movement of the train, and brakes the train in case of dangerous operations acted by the driver.
- An ATS Simple interacts with an ATP Wayside IXL or with an ATP Wayside Controller to require a root for the train that has to be dispatched. The ATP Wayside IXL performs the same operations of the IXL pure (i.e., allows/denies routing).
- The ATP Wayside Controller interacts with an IXL Controllable. The ATP Wayside Controller is able to bypass some operations of the IXL Controllable, which, unlike the IXL Pure, allows the ATP Wayside Controller to bypass some of its controls to achieve improved performances. Both the ATP Wayside IXL and the ATP Wayside Controller implement all the operations of the ATP Wayside Simple (i.e., sending the authorization of movement to the ATP Onboard).

Conclusions and future work

- We have illustrated how to derive valid products from a product family by managing variability in a single logical framework consisting of an MTS and an associated set of MHML formulae.
- We have used the same framework to verify behavioural properties over products and families.

Several issues deserve further investigation:

- A major issue is the definition of design and verification techniques and tools that, based on the principles expressed in this paper, but hiding most of the formality, can be routinely used by product line engineers.
- Furthermore a thorough investigation of the relation between features and actions when translating feature models into transition systems.