Introduction to Software Product Lines and variability analysis
(towards analysing quantitative aspects of a bike-sharing product line)

Maurice ter Beek

QUANTICOL plenary meeting, ISTI-CNR, Pisa, Italy, 12-02-2014

These slides are partly based on slides by Sven Apel, Slinger Jansen and Klaus Pohl
Outline

- (Software) Product Line Engineering
- Product variability and features
- Variability modelling and analysis

- Towards analysing quantitative aspects of a bike-sharing product line
Product line engineering in theory...
...and in practice
Software Product Line Engineering

- Developing a family of products from a common reference model (usually in terms of features) and mass customisation (to serve different markets)

Two main differences w/ classical software engineering

- Two distinct development processes
  - Domain engineering: develop reusable domain artefacts
  - Application engineering: develop individual products by reusing domain artefacts

- Variability
  - Common features that are part of all products
  - Variable features that can be selected for a product
Variability in requirements

R12: The navigation system must allow the user to make inputs using a control panel or by voice entry

- R12 comprises the following three realisations
  1. A navigation system that allows the user to make inputs only via the control panel
  2. A navigation system that allows the user to make inputs only via voice entry
  3. A navigation system that allows the user to make inputs only via the control panel and by voice entry

- Conjunction is a logical “or” or an exclusive “or”? 
- Is only one system asked for, or two or three different systems?
Features

- What is a feature?
  - End-user visible behaviour or property of a system...
  - ...that may be optional and/or may have alternatives

- Features represent **commonalities and variabilities** of (software) systems
# What’s in a feature?

<table>
<thead>
<tr>
<th>Reference</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kang <em>et al.</em> [3]</td>
<td>“a prominent or distinctive user-visible aspect, quality or characteristic of a software system or systems”</td>
</tr>
<tr>
<td>Kang <em>et al.</em> [8]</td>
<td>“distinctively identifiable functional abstractions that must be implemented, tested, delivered, and maintained”</td>
</tr>
<tr>
<td>Eisenecker and Czarnecki [6]</td>
<td>“anything users or client programs might want to control about a concept”</td>
</tr>
<tr>
<td>Chen <em>et al.</em> [10]</td>
<td>“a product characteristic from user or customer views, which essentially consists of a cohesive set of individual requirements”</td>
</tr>
<tr>
<td>Batory [11]</td>
<td>“an elaboration or augmentation of an entity(s) that introduces a new service, capability or relationship”</td>
</tr>
<tr>
<td>Batory <em>et al.</em> [12]</td>
<td>“an increment in product functionality”</td>
</tr>
<tr>
<td>Apel <em>et al.</em> [13]</td>
<td>“a structure that extends and modifies the structure of a given program in order to satisfy a stakeholder’s requirement, to implement and encapsulate a design decision, and to offer a configuration option.”</td>
</tr>
</tbody>
</table>

(Classen *et al.* @ FASE’08)
Feature selection

Typically, only a subset of feature combinations is valid
Products and product lines

- Product $\Leftrightarrow$ valid feature combination (configuration)
- Product line $\Leftrightarrow$ set of valid feature combinations of a domain
Variability models: explicit decisions

- Compact representations of all products of a product family in terms of their features
- **Feature diagram/model**: hierarchical tree structure, w/ the family as root, features as nodes and possibly further cross-tree constraints (Kang et al. ’90)
- **Common Variability Language (CVL)**: OMG's effort to standardise variability modelling as a separate and generic language to define variability on base models
- **Orthogonal Variability Model** (Pohl et al. ’05), etc., etc.
Advantages of dedicated conceptual model

- **Improved communication** with different stakeholders (e.g. communicate to customers which variants can be selected at which variation points)

- **Transparent decisions** i.e. the originator of a variation point is forced to state the rationale for introducing variability in a specific domain artefact

- **Relationships between requirements and variants** become **traceable** (e.g. stakeholders can document which requirements, design, implementation and test artefacts are influenced by a variant)
Mobile phone feature diagram

- GPS
  - Basic
- Screen
  - Colour
- Calls
  - High resolution
- Media
  - Camera
  - MP3
A product line of Lego minifigures (16 valid feature combinations)
‘Feature diagram’ for configuring the 16 valid products (allowing more…)}
Configure your 11-inch MacBook Air

Processor
Enjoy incredible performance from fourth-generation Intel Core processors. Choose the speed and processor you want.

- 1.3GHz Dual-Core Intel Core i5, Turbo Boost up to 2.6GHz
- 1.7GHz Dual-Core Intel Core i7, Turbo Boost up to 3.3GHz [+ £130.00]

Memory
More memory (RAM) increases overall performance and enables your computer to run more applications at the same time.

- 4GB 1600MHz LPDDR3 SDRAM
- 8GB 1600MHz LPDDR3 SDRAM [+ £80.00]

Storage
Your MacBook Air comes as standard with flash storage. Flash storage has no moving parts and provides faster responsiveness and enhanced durability.

- 256GB Flash Storage
- 512GB Flash Storage [+ £240.00]
FIND YOUR BMW.

Filter
> Reset filter

- Budget
- Vehicle type
  - All
  - Petrol
  - Diesel
  - Hybrid
  - Electric Vehicle

- Body type

- Number of seats

30 Vehicles (465 Model variants)

1. BMW 1 Series 3-door Sports Hatch (34)
   from £17,775.00

2. BMW 1 Series 5-door Sports Hatch (39)
   from £18,305.00

2. BMW 2 Series Coupé (14)
   from £24,265.00

3. BMW 3 Series Saloon (56)
   from £23,550.00

4. BMW 3 Series Touring (54)
   from £24,885.00

5. BMW 3 Series Gran Turismo
   (39)
   from £29,200.00
Variability dependencies (1/2)

Allowed choices of variants at a specific variation point

- **Mandatory variant**: if the variation point is selected, this variant *must* always be selected
- **Optional variant**: if the variation point is selected, this variant *may* be selected but it does not have to be
- **Alternative variants** i.e. a collection of at least two optional variants together with a [min...max] notation to indicate the permissible number of variants to be selected: if the variation point is selected, at least “min” variants *must* be selected while at most “max” variants *may* be selected
Variability dependencies (2/2)

Cross-tree constraints

- **Requires**: indicates that the presence of one feature requires the presence of another feature
- **Excludes**: indicates that the presence of two features is mutually exclusive

Quantitative constraints

- **Feature attributes (non-functional)**: $\text{cost(feature)} = 7$, etc.
- $\text{cost(product)} = \sum\{\text{cost(feature)} \mid \text{feature} \in \text{product} \}$
Attributed feature model

Quantitative constraint: cost(Coffee Machine) ≤ 30
Elicitation of requirements variability

Two additional tasks wrt requirements engineering for single systems:

1. **Binding** the defined variability
2. **Documenting** the variability binding
So why have variability?

- **Flexibility to deliver**
  - Different versions of products based on the same trunk of code (Windows XP professional, Windows XP Home edition, …)
  - Onto different platforms (a linux, a mac and a windows version)
  - With different components (Windows Media Player + Divx, iTunes, …)
  - Onto different plug-in products, etc., etc., etc.

- **Is it an advantage then? Yes, but…**
  - Requirements engineering becomes more complex
  - Deployment becomes more complex
  - Sales becomes more complex
  - Updates become WAY more complex
  - Testing becomes more complex (all configurations need to be tested)

  “We always have 126,000,000 different bicycles in store!
  (but only the parts for 1,000…)”
Weakness: feature interactions

- Features need to be combined, but have restrictions on each other
- Consider e.g. a phone switching system with the following features
  1. Call forwarding
  2. Do not disturb
  3. 3-way calling
  4. No interaction between the features
- Scenario 1: Bob forwards his calls to Alice; Alice sets “Do not disturb”; Bob receives a call, which is forwarded to Alice; Alice’s phone rings!
- Scenario 2: Bob still forwards his calls to Alice; Bob invites Alice for a 3-way call; Carolina calls Bob to become part of the 3-way call; either
  1. Carolina is forwarded to Alice, who cannot accept any other calls then; or
  2. Carolina is accepted into the 3-way call
Weakness: complexity/scalability w/ 33 optional, independent features

A unique product for every person on this planet
320 optional, independent features

more possible products than estimated

atoms in the universe
(Behavioural) variability analysis

Rigorously establish critical system requirements (for quality assurance) w/ formal methods and automated analysis tools

- For decades successful in single product/system engineering
- Not exploited broadly in SPLE, while correctness of artifacts for reuse and correctness of developed products is of crucial importance (many massively produced (embedded) systems and safety- or business-critical applications)

Traditionally

- Mainstream formal methods do not consider variability directly
- Formal methods that have been applied in SPLE mainly focus on structural rather than behavioural properties (feature model analysis, product line testing, etc.)
Variability analysis
(type checking, static analysis, model checking, theorem proving, testing, etc.)

Product-Based Analysis

Family-Based Analysis

Feature-Based Analysis
Product-based analysis

Simple approach
Standard tools available
Infeasible for large product sets

$O(2^n)$ for $n$ features
Family-based analysis

Configuration lifting
Variability encoding

Beneficial for many products with substantial similarities
Generates complex analysis tasks
Closed-world assumption

Metaproduction, product simulator, …
Feature-based analysis

Beneficial for large and cohesive features
Support for open-world scenarios
Incomplete w.r.t. feature interactions
PLA cube (von Rhein et al. @ VaMoS’13)

Sampling based on coverage criteria like pairwise or t-wise coverage (or other heuristics)
Towards analysing quantitative aspects of a bike-sharing product line

Maurice ter Beek
(joint work w/ Alessandro Fantechi, Stefania Gnesi & Franco Mazzanti)

FMT, ISTI, CNR

QUANTICOL plenary meeting, ISTI–CNR, Pisa, Italy, 12-02-2014
Outline

Introduction to SPL and variability analysis

1. Aim of our research activity... in QUANTICOL
2. Case study: A family of Bike-Sharing Systems (BSS)
   - A model-driven tool chain experience
   - Model checking value-passing modal specifications
3. Conclusions and future work
Characteristics of CAS:

- Coordination based on (local) decentralised interaction
- Large scale, heterogeneous agents, competing goals, open
- Capacity to smoothly adapt to changing circumstances
- Spatially inhomogeneous distribution influences global patterns
- Multiple scales in time and space, systems of systems
- Decentralised and centralised control

Behaviour of CAS components may exhibit variability not only in kind of features, but also in quantitative characteristics of the features

T3.3 Relating Local and Global System Views with Variability Analysis
Aim of our research activity...

Formal methods in SPLE (\textit{cf.} Intro SPL)

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

Or, in the words of Dave Clarke (Uppsala University, Sweden)

\textit{“Behaviour is what we need. Without behaviour, it’s just sticks and balls. With behaviour, you get cricket.”}

Since a decade or so several approaches

- Variants of UML diagrams (Jézéquel et al.), Petri nets (Clarke et al.), Event-B (Gondal et al.)
- Models with an LTS-like semantics: variants of MTS (Fischbein et al., Fantechi et al.), I/O automata (Larsen et al., Lauenroth et al.), CCS (Gruler et al., Gnesi et al., ter Beek et al.), FTS (Classen et al.), FSM (Millo et al.), mCRL2 (ter Beek et al.)
Develop a framework able to deal with ‘quantitative variability’

Provide tool to support this framework with formal verification

Very preliminary results (submitted)

- We established a model-driven tool chain of (academic) tools with different functionalities regarding the analysis of SPL, from feature modelling to product derivation to quantitative evaluation of the attributes of products.

- We defined a value-passing modal process algebra interpreted over Modal Transition Systems (MTS).

- We defined a deontic (variability-aware) extension of an action-based branching-time modal temporal logic able to handle data.

- We extended our Variability Model Checker (VMC) to support this setup.
Bike-sharing systems (BSS)

Finding the right BSS for a particular city poses many questions:

- How many and what kind of bikes?
- How many and what kind of stations and where to place them?
- Which features (antitheft, maintenance, smart services, etc.)?
- With or without (dynamic) redistribution?
- Incentives for users to return bikes to less popular stations?
- Costs and charging policy (credit card, keycard, etc.)?

How to evaluate the various options, costs/benefits, improvements and changes in a systematic way?

We started to collaborate with PisaMo, an in-house public mobility company of Pisa’s administration that introduced the public BSS CicloPi in Pisa (currently ± 140 bikes, ± 14 stations) in May 2013.
We first focussed on structural requirements

In the requirements engineering phase we performed text mining on a set of documents describing current BSS to extract a set of features: NLP approach based on so-called contrastive analysis to identify commonalities and variabilities from natural language documents

Ferrari, Spagnolo, dell’Orletta @ SPLC’13

We then derived an initial feature model from the features that we considered most interesting for developing a ‘good quality’ BSS as starting point for our study

Finally, we sought support in available tools for the possibility of adding attributes and quantitative constraints to our BSS model
Feature model (with S.P.L.O.T.)

Mendonça, Branco, Cowan @ OOPSLA’09

Feature Diagram

- **Bikesharing**
  - **Status**
    - [1..*]
      - RTInfoWeb
      - AllBikesNow
  - **Bike**
    - Localization
      - [1..*]
        - GPS
        - RFID
    - Antithieves
  - **DockingStation**
    - [1..1]
      - Fixed
      - FixedPortable
      - Flexible
    - Maintenance
  - **Redistribution**
    - Reward
  - **Users**

Cross-Tree Constraints

- ( ~AllBikesNow V GPS )
- ( GPS V ~Antithieves )
- ( Keycard V ~KeycardDispenser )
- ( ~Keycard V KeycardReader )
- ( ~Keycard V KeycardDispenser )

Click to create a constraint

Feature Information Table

| Id: | 
| Name: | 
| Description: | 
| Type: | 
| #Children: | 
| Tree level: |

Update Feature Model

Feature Model Statistics

- #Features: 29
- #Mandatory: 5
- #Optional: 10
- #XOR groups: 1
- #OR groups: 4
- #Grouped: 13
- #Cross-Tree Constraints (CTC): 5
- CTCR (%): 0.21
- CTC distinct vars: 6
- CTC clause density: 0.83

Feature Model Analysis

- Consistency: Consistent
- Dead Features: None
- Core Features: 6 feature(s)
- Valid Configurations: 60,840
We may turn this into an attributed feature model, e.g. to measure the costs of different products.
Attributed feature model in Clafer

Bąk, Czarnecki, Wąsowski @ SLE’10, Murashkin, Antkiewicz, Rayside, Czarnecki @ SPLC’13

abstract Feature
  customersat : integer
  cost : integer
  capacity : integer

abstract SecFeature : Feature
  security : integer

abstract BIKES
or Status : Feature ?
  [ customersat = 25 ]
  [ cost = 0 ]
  [ capacity = 0 ]
RTInfoWeb : Feature
  [ customersat = 10 ]
  [ cost = 5 ]
  [ capacity = 0 ]
AllBikesNow : Feature
  [ customersat = 20 ]
  [ cost = 10 ]
  [ capacity = 0 ]
Bike : Feature
  [ customersat = 0 ]
  [ cost = 0 ]
  [ capacity = 0 ]
or Localization : Feature ?
  [ customersat = 3 ]
  [ cost = 3 ]
  [ capacity = 0 ]
RFID : Feature
  [ customersat = 10 ]
  [ cost = 10 ]
  [ capacity = 0 ]
GPS : Feature
  [ customersat = 15 ]
  [ cost = 15 ]
  [ capacity = 0 ]
Antithieves : SecFeature ?
  [ customersat = 5 ]
  [ cost = 7 ]
  [ capacity = 0 ]
  [ security = 1 ]
xor DockingStation : SecFeature
  [ customersat = 0 ]
  [ cost = 0 ]
  [ capacity = 0 ]
  [ security = 1 ]
Fixed : Feature
  [ customersat = 17 ]
  [ cost = 30 ]
  [ capacity = 5 ]
FixedPortable: Feature
  [ customersat = 20 ]
  [ cost = 35 ]
  [ capacity = 5 ]
Flexible: Feature
  [ customersat = 23 ]
  [ cost = 40 ]
  [ capacity = 10 ]
[ Antithieves => GPS ]
[ AllBikesNow => GPS ]
total_customersat : integer = sum Feature.customersat
total_cost : integer = sum Feature.cost
total_capacity : integer = sum Feature.capacity
total_security : integer = sum SecFeature.security
Mybike : BIKES
<< max Mybike.total_customersat >>
<< min Mybike.total_cost >>
<< max Mybike.total_capacity >>
<< max Mybike.total_security >>
Multi-objective optimisation (with ClaferMOOVisualizer)
Behavioural requirements of a PL

Inspired by Fricker, Gast @ arXiv, September 2013

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, (s)he 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
The framework so far

Main ingredient: Modal Transition Systems (MTS)

- LTS distinguishing admissible *may* and necessary *must* transitions
  Larsen, Thomsen @ LICS’88

- Recognised as a useful model to describe in a compact way the possible *behaviour* of all the products (LTS) of a product family
  Fischbein, Uchitel, Braberman @ ROSATEA’06

- MTS cannot model variability constraints regarding *alternative* features, nor regarding *requires/excludes* inter-feature relations
  Asirelli, ter Beek, Fantechi, Gnesi @ iFM’10

- Our solution: add a set of *variability constraints* to the MTS to be able to decide which derivable products (LTS) are valid ones
  Asirelli, ter Beek, Fantechi, Gnesi @ SPLC’11
A behavioural model, amenable to model checking, able to formalise

1. shared behaviour: common among all variants
2. variation points: differentiate between variants

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

An MTS has two distinct transition relations

1. may transition relation \(\rightarrow\Diamond \subseteq Q \times A \times Q\): possible transitions
2. must transition relation \(\rightarrow\Box \subseteq Q \times A \times Q\): required transitions

By definition, any required transition is also possible: \(\rightarrow\Box \subseteq \rightarrow\Diamond\)

\((-\rightarrow \equiv \rightarrow\Diamond \setminus \rightarrow\Box)\)
A product LTS is obtained from a family MTS in the following way

1. include all (reachable) must transitions and
2. include a subset of the (reachable) may transitions

Each selection gives rise to a different variant

\( \nu \)-\text{ACTL} is used to complement a behavioural description by an MTS by expressing those constraints that MTS cannot model

\[
\phi ::= true \mid \neg \phi \mid \phi \land \phi \mid [\psi] \phi \mid [\psi]^{\square} \phi \\
E \pi \mid A \pi \mid \mu Y.\phi(Y) \mid \nu Y.\phi(Y)
\]

\[
\pi ::= F \phi \mid F^{\square} \phi \mid F \{\psi\} \phi \mid F^{\square} \{\psi\} \phi
\]
VMC accepts as input a model specified in the modal process algebra (+ variability constraints of form ALTernative, EXCludes, REQuires?)

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

VMS is freely usable online: http://fmtlab.isti.cnr.it/vmc/

Model checking of v-ACTL formulae over MTS can be achieved in a complexity that is linear w.r.t. the state space size
Critical point in the formalisation by MTS: lack of a possibility to model an adequate representation of the data that may need to be described when considering real systems (even for family of BSS)

Other possible approaches:

- **Parametric MTS**
  
  Beneš, Křetínský, Larsen, Møller, Srba @ ATVA’11

- **Quantitative (e.g. weighted) MTS**
  
  Larsen, Legay @ WADT’12, Bauer, Fahrenberg, Juhl, Larsen, Legay, Thrane @ FMSD’13

- **Parametric modelling w/ formal process-algebraic specification language mCRL2 and its industry-strength toolset**
  
  research in progress by ter Beek & de Vink
A value-passing modal process algebra

\[ N ::= [P] \]
\[ P ::= T(e) \mid P / L / P \]
\[ T(e) ::= nil \mid K(e) \mid A.T(e) \mid T(e) + T(e) \mid [e \otimes e] T(e) \]
\[ A ::= a(e) \mid a(may, e) \mid a(?v) \mid a(may, ?v) \]
\[ e ::= v \mid \text{int} \mid e \pm e \]

\[ (or) \quad \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \quad \text{if } \alpha \in \{a(e), a(?v)\} \]
\[ (guard) \quad \frac{[e_1 \otimes e_2] P(e_3) \rightarrow P(e_3)}{[e_1 \otimes e_2] P[\text{next} \rightarrow e_1, e_2, e_3]} \]
\[ (par) \quad \frac{P \xrightarrow{a(e_1)} P' \quad Q \xrightarrow{a(e_2)} Q'}{P / L / Q \xrightarrow{a} P' / L / Q'} \quad \text{if } a \in L, e_1 = e_2 \]
\[ (par) \quad \frac{P \xrightarrow{a(?v)} P' \quad Q \xrightarrow{a(e)} Q'}{P / L / Q \xrightarrow{a} P'[e / v] / L / Q'} \quad \text{if } a \in L \]

(similarly in case of may actions and for the remaining operators)
Value-passing BSS specification

Station(I,N,J,M) = request(I).
   ( [N=0] nobike(I).Station(I,N,J,M) +
     [N>0] givebike(I).Station(I,N-1,J,M) ) +
   deliver(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(s1,2,s2,2) /redistribute/ Station(s2,2,s1,2)

Users(I,J) = request(I).
   ( givebike(I).deliver(J).Users(I,J) +
     nobike(I).Users(I,J) )

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

net BSS = STATIONS /request,givebike,nobike,deliver/ USERS
MTS with parameters and values
(w/ two user groups 224 rather than 18 states)
The extended modelling and verification environment described so far has been implemented in VMC v6.0 (November 2013)

http://fmt.isti.cnr.it/vmc/v6.0

Accepts models specified in the value-passing modal process algebra

Allows model checking properties expressed in $v$-ACTL extended w/ action values
EVENTUALLY IT MUST OCCUR THAT STATION 1 HAS NO BIKES:

$EF \Diamond \{ \text{nobike}(s1) \} \text{ true} $  
TRUE

EVENTUALLY IT MAY OCCUR THAT STATION 2 HAS NO MORE BIKES:

$EF \{ \text{nobike}(s2) \} \text{ true} $  
FALSE

IT IS ALWAYS THE CASE THAT EVENTUALLY STATION 1 MUST GIVE A BIKE, POSSIBLY AFTER IT HAS FIRST RECEIVED BIKES AFTER REDISTRIBUTION:

$AG((EF \Diamond \{ \text{givebike}(s1) \} \text{ true}) \lor (EF \Diamond [\text{redistribute}(*,s1,*)] EF \Diamond \{ \text{givebike}(s1) \} \text{ true})) $  
TRUE
From modal specifications of families to products

Products may be derived according to an extension of the algorithms already present in VMC, for example:

Verification results of \(v\)-ACTL properties over an MTS are inherited by the entire family of derived product according to the following rules:

- Formulas without negation and only composed from \(false\), \(true\) and the operators \(\land\), \(\lor\), \(\langle \rangle^0\), \([\cdot]\), \(\mu\), \(\nu\), \(EF^0\), \(EF^\varnothing\{\}\), \(AF^0\), \(AF^\varnothing\{\}\) and \(AG\) that are valid for a family MTS are also valid for all its product LTSs

- Formulas without negation and only composed from \(false\), \(true\) and the operators \(\land\), \(\lor\), \(\langle \rangle\), \(\mu\), \(\nu\), \(EF\) and \(EF\{\}\) that are false for a family MTS are false for all its product LTSs
Future work

In general:

- Extend data **beyond integer** value passing
- Study and implement the **derivation** of products in the presence of both **structural** constraints (ALT, EXC & REQ from feature models) and **quantitative** constraints (attributed feature models)
- Study the **inheritence** of the result of verifying a v-ACTL formula over an MTS by its product LTS in the presence of both types of constraints

In QUANTICOL:

- Study **quantitative MTS**
- **Scalability**?
Publicity: Call for Papers

JOURNAL OF LOGIC AND ALGEBRAIC PROGRAMMING

Special Issue on

Formal Methods in Software Product Line Engineering

Submission of papers: July 15, 2014
First review decision: December 15, 2014
Revision due: February 15, 2015
Acceptance notification: April 15, 2015
Final manuscript due: June 15, 2015
Expected publication: Summer 2015

Guest editors:
- Maurice ter Beek (ISTI–CNR, Pisa, Italy)
- Dave Clarke (U Uppsala, Sweden & KU Leuven, Belgium)
- Ina Schaefer (TU Braunschweig, Germany)