Statistical Model Checking for Software Product Lines

Maurice H. ter Beek
ISTI–CNR, Pisa, Italy

joint work with

Axel Legay
Inria Rennes, France

Alberto Lluch Lafuente
DTU, Lyngby, Denmark

Andrea Vandin
IMT Lucca, Italy

ISoLA 2016
Corfu, Greece
10 October 2016
Outline

1. Background: Software Product Lines
2. Model checking behavioural variability
3. Probabilistic Feature-oriented Language with Quantitative constraints
4. Case study: A product line of bikes (from PisaMo / Bicincittà)
5. Statistical Model Checking with Maude/Z3/MultiVeStA
6. Conclusions and future work
Software Product Lines (SPLs)

Software Product Line Engineering (SPLE)

Develop and maintain a (software) product line using a shared architecture or platform (commonalities) and mass customisation (variabilities) to serve, e.g., different markets, thus allowing for (software) reuse.

Aim: Maximise commonalities whilst minimising cost of variations (i.e. of individual products)

Product: a valid combination (configuration) of features

Product line or family: a set of valid feature combinations of a domain
**Variability modelling** (Lego example by S. Apel (U Passau, Germany))

**Variability in terms of features**

- stakeholder visible pieces of functionality of a system
- which may be optional and/or may have alternatives
- only specific feature combinations lead to valid products

---

**Feature model:** compact representation of all products of a product line
Scalability is a major issue! (slide by C. Kästner, CMU, USA)

33 optional, independent features

a unique product for every person on this planet
Model checking behavioural variability

Formal analysis, testing and verification in the presence of variability

⇒ Lift success stories from single product/systems to families/SPLs

Ongoing research presented at FMSPLE + SPLC

‘12 S. Gnesi & M. Petrocchi: Towards an executable algebra for product lines

‘13 M.H. ter Beek, A. Lluch Lafuente & M. Petrocchi: Combining declarative and procedural views in the specification and analysis of product families

‘15 M.H. ter Beek, A. Legay, A. Lluch Lafuente & A. Vandin: Quantitative analysis of probabilistic models of software product lines with statistical model checking

‘15 M.H. ter Beek, A. Legay, A. Lluch Lafuente & A. Vandin: Statistical analysis of probabilistic models of software product lines with quantitative constraints

Related work

- Feature-oriented models: UML, Petri nets, MTSs, CCS, FSMs, FTSs, ...

- Both off-the-shelf and dedicated (software, probabilistic, statistical) model checkers: SPIN, SNIP, VMC, NuSMV, mCRL2, JPF, PRISM, ...
**QFLan**: Probabilistic Feature-oriented Language with Quantitative constraints

### Concurrent constraint programming paradigm
- Probabilistic modelling: uncertainty, failure rates, randomisation, ...  
- Quantitative constraints: Quality of Service, reliability, performance, ...

### Combine declarative and procedural specification
- Constraint store allows to specify all ordinary feature constraints, as well as quantitative constraints over features’ attributes  
- Rich set of process-algebraic operators allows to specify both the configuration and the (probabilistic) behaviour of products

### Declarative and procedural views closely related
- 1. process execution is constrained by the store to avoid inconsistencies  
- 2. process can query the store to resolve configuration / behavioural option  
- 3. process can update the store by adding new features, also at runtime
QFLan: Syntax

\( f, g \in \mathcal{F}, r \in \mathbb{R}^+, a \in A, p \in \mathcal{P}, \preceq \in \{\leq, <, =, \neq, >, \geq\}, \) and \( \pm \in \{+, -, \div, \times\} \)

- **(fragments)** \( F ::= [S \mid P] \)
- **(constraints)** \( S, T ::= K \mid f \triangleright g \mid f \otimes g \mid S \triangleright T \mid \top \mid \bot \)
- **(processes)** \( P, Q ::= \emptyset \mid X \mid (A, r).P \mid P + Q \mid P; Q \mid P \parallel Q \)
- **(actions)** \( A ::= a \mid \text{install}(f) \mid \text{uninstall}(f) \mid \text{replace}(f, g) \mid \text{ask}(K) \)
- **(propositions)** \( K ::= p \mid \neg K \mid K \lor K \mid E \otimes E \)
- **(expressions)** \( E ::= r \mid \text{attribute}(f) \mid E \pm E \)

Specify quantitative constraints of SPL models

- Universe \( \mathcal{P} \) of propositions: predicates \( \text{has}(f) \), \( \text{do}(a) \), \( \text{in}(\text{context}) \), \ldots
- Arithmetic constraints: e.g. \( \sum_{f \in \mathcal{P}_\mathcal{F}} \text{weight}(f) \leq 15 \)
- Action constraints: e.g. \( \text{do}(\text{sell}) \rightarrow \sum_{f \in \mathcal{P}_\mathcal{F}} \text{price}(f) \geq 250 \)
  i.e. a guard to allow/forbid executing an action

Specify probabilistic aspects of SPL models

- \( (A, r).P \): perform action \( A \) with rate/weight \( r \), then behave as \( P \)

\( \Rightarrow \) Likelihood of user behaviour or of installation of a certain feature
QFLan: Semantics

\[ \to \subseteq \mathbb{N}^F \times \mathbb{R}^+ \times F \], with \( F \) set of all terms generated by \( F \)

**DTMC semantics**

- normalising rates into \([0..1]\) such that \( \forall \) states \( s : \sum s \xrightarrow{\ell} = 1 \)
- transition \( \ell \) label corresponds to probability of transition being executed

\[ \Rightarrow \] use SMC because, in general, the DTMC is too large to generate

1-1 correspondence with rewrite rules in Maude implementation

\[ \Rightarrow \] Implementation compact, easy to read / extend, efficiently executable
<table>
<thead>
<tr>
<th>Model Checking (MC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatically check whether a model satisfies a temporal logic property (LTL, CTL) and provide a counterexample if it doesn’t</td>
</tr>
<tr>
<td>Exhaustive, but suffers from state space explosion problem</td>
</tr>
<tr>
<td>BLAST, CADP, JPF, mCRL2, (Nu)SMV, SPIN, UPPAAL, ...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Probabilistic Model Checking (PMC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model check whether a stochastic model satisfies a temporal logic property (PCTL, CSL) with a probability greater than a certain threshold</td>
</tr>
<tr>
<td>Model uncertainty/performance; do quantitative analysis (QoS, ...)</td>
</tr>
<tr>
<td>CADP, LiQuor, MRMC, PARAM, PRISM, UPPAAL-PRO, ...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Statistical Model Checking (SMC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation-based technique to statistically approximate (P)MC</td>
</tr>
<tr>
<td>Highly parallelisable and automatable; tunable preciseness via CI</td>
</tr>
<tr>
<td>APMC, PLASMA, PRISM, UPPAAL, (P)VeStA, MultiVeStA, YMER, ...</td>
</tr>
</tbody>
</table>
Structural constraints of product line of bikes

(C1) \( \sum_{f \in P_F} \text{price}(f) \leq 600 \): a bike may cost at most 600 €

(C2) \( \sum_{f \in P_F} \text{weight}(f) \leq 15 \): a bike may weigh up to 15 kg

(C3) \( \sum_{f \in P_F} \text{load}(f) \leq 100\% \): a bike's total computational load may not exceed 100%
Behavioural constraints of product line of bikes

Additional action constraints

(C4) \( \text{do(sell)} \rightarrow \sum_{f \in \mathcal{P}_F} \text{price}(f) \geq 250 \)

(C5) \( \text{do(irreparable)} \rightarrow \sum_{f \in \mathcal{P}_F} \text{price}(f) \leq 400 \)
QFLan specification

\[ FR \doteq [S \mid F] \]

\[ S \doteq DS \ PS \ QS \ AS \ IS \]

\[ DS \doteq \ldots \quad PS \doteq \ldots \quad QS \doteq \ldots \quad AS \doteq \ldots \quad IS \doteq \ldots \]

\[ F \doteq (sell, 7).D \quad \text{// Installing optional features:} \]
\[ + (install(s), 6).F + (install(m), 10).F + (install(n), 6).F + (install(u), 3).F + (install(c), 20).F \]
\[ + (install(g), 4).F + (install(a), 5).F + (install(o), 10).F + (install(i), 10).F + (install(k), 8).F \]

\[ D \doteq (deploy, 10).P \quad \text{// Installing optional features:} \]
\[ + \ldots \text{same as } F \]

\[ \text{// Replacing mandatory and exclusive features:} \]
\[ + \ldots \text{same as } F, \text{ but Frame cannot be changed} \]

\[ P \doteq (book, 10).M + (maintain, 1).D \]

\[ M \doteq (stop, 5).H + (break, 1).B + (c, 20).M + (i, 20).M \]

\[ H \doteq (start, 5).M + (break, 1).B + (park, 5).P + (c, 20).H + (i, 10).H + (s, 10).H + (u, 10).H \]

\[ B \doteq (assistance, 10).D + (irreparable, 1).T \]

\[ T \doteq (install(trashed), 1).\emptyset \]
FR is composed of process F and store S of constraint sets:

**DS** Constraints from the feature diagram (incl. cross-tree constraints)

\[ DS \doteq y \otimes r \otimes w \ d \otimes h \ \ldots \ g \triangleright a \ n \triangleright m \ \ldots \]

**PS** Predicates for attributes of concrete features in feature diagram

\[ PS \doteq \text{price}(y) = 100 \ \text{weight}(y) = 0.3 \ \ldots \ \text{price}(c) = 100 \ \text{load}(c) = 5 \ \ldots \]

**QS** Quantitative constraints

\[ QS \doteq \text{price}(b) \leq 800 \ \text{weight}(b) \leq 20 \ \text{load}(b) \leq 100 \]

**AS** Action constraints

\[ AS \doteq \text{do}(\text{sell}) \rightarrow (\text{price}(b) \geq 250) \ \ldots \ \text{do}(c) \rightarrow \text{has}(c) \ \ldots \]

**IS** Initially installed feature set, i.e. AllYear Wheels and Diamond Frame

\[ IS \doteq \text{has}(y) \ \text{has}(d) \]
A process in the specification

\[ FR = [S | F] \]

\[ S = DS PS QS AS IS \]

\[ DS = \ldots \quad PS = \ldots \quad QS = \ldots \quad AS = \ldots \quad IS = \ldots \]

\[ F = (sell, 7).D \quad // Installing optional features: \]

+ (install(s), 6).F + (install(m), 10).F + (install(n), 6).F + (install(u), 3).F + (install(c), 20).F
+ (install(g), 4).F + (install(a), 5).F + (install(o), 10).F + (install(i), 10).F + (install(k), 8).F

\[ \quad // Replacing mandatory and exclusive features: \]

+ (replace(y, r), 5).F + (replace(y, w), 5).F + (replace(r, y), 10).F + (replace(r, w), 5).F
+ (replace(w, y), 10).F + (replace(w, r), 5).F + (replace(d, h), 3).F + (replace(h, d), 3).F

\[ F \] implements \textsc{Factory}\textsc{’}s behaviour as a \textit{weighted} choice among:

(1) With rate 7, the bike is sold and sent to the depot. This action can only be executed if C4 \((\text{do}(sell) \rightarrow (\text{price}(b) \geq 250))\) is respected

(2) Install optional features and iterate on \(F\). The installations are only performed if \(DS\) and \(QS\) are preserved

(3) Replace pre-installed mandatory exclusive features \(IS\), i.e. Wheels or Frame. Again, \(DS\) and \(QS\) must be preserved

\textsc{QFLan}\textsc{’}s semantics forbids re-installing (installed) features
MultiVeStA and its property language MultiQuaTEx

**MultiVeStA extends (P)VeStA**
- Extends discrete-event simulators with distributed SMC capabilities
  - [http://sysma.imtlucca.it/tools/multivesta/](http://sysma.imtlucca.it/tools/multivesta/)
- Used so far to analyse transportation systems, volunteer clouds, crowd-steering, swarm robotic scenarios, software product lines, ...

**MultiQuaTEx extends QuaTEx**
- Statistical estimations of quantitative properties in MultiQuaTEx
- Expected values of properties that can take on any value from \( \mathbb{R} \):
  - “Average cost of products generated from an SPL specification?”
  - Computed as mean value of \( n \) samples taken from \( n \) simulations, with \( n \) large enough such that size of the \( (1 - \alpha) \times 100\% \) CI is bounded by \( \delta \)
    (i.e. if an expression is estimated as \( \bar{x} \), then with probability \( (1 - \alpha) \) its actual expected value belongs to the interval \( [\bar{x} - \delta/2, \bar{x} + \delta/2] \))
- Parametric properties to perform many analyses, reusing simulations

M.H. ter Beek et al. (INRIA/CNR/DTU/IMT)
Statistical Model Checking for SPLs
ISO LA 2016 19 / 27
Quantitative analyses of case study

Example properties

\((P_1)\) Average price, weight or load of a bike when it is first deployed

\((P_2)\) For each of the 15 primitive features (leaves), the probability to have it installed when a bike is first deployed

\(P_1 + P_2\) evaluated at bike’s first deployment (two versions of C1 and C2)

<table>
<thead>
<tr>
<th>Attributes ((P_1))</th>
<th>Features ((P_2))</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>C2</td>
</tr>
<tr>
<td>600</td>
<td>15</td>
</tr>
<tr>
<td>800</td>
<td>20</td>
</tr>
</tbody>
</table>

Constraints in disagreement with quantitative attributes of features

- Probability of installing an engine \((g)\) is very low, estimated at 0 (i.e. with probability 0.9 it belongs to \([0, 0.05]\), according to the used CI).

- This is likely because original C1 and C2 (first row) are too strict: Estimated average price and weight of bikes when first deployed is 391.91 € and 7.8 kg, while engine costs 300 € and weighs 10 kg.
Quantitative analyses of case study

Example properties

\((P_1)\) Average price, weight or load of a bike when it is first deployed

\((P_2)\) For each of the 15 primitive features (leaves), the probability to have it installed when a bike is first deployed

**MultiQuaTEx expression for \(P_1+P_2\)**

```
ObsAtFD(obs) = if \{s.rval("first-deploy") == 1.0\} then s.rval(obs) else #ObsAtFD(obs) fi;

eval E[ObsAtFD("price")]; eval E[ObsAtFD("weight")]; eval E[ObsAtFD("load")];
 eval E[ObsAtFD("y")]; eval E[ObsAtFD("r")]; ...; eval E[ObsAtFD("h")];
```

**MultiQuaTEx in a nutshell**

- **ObsAtFD**: a user-defined parametric recursive temporal operator
- **first-deploy**: fictitious feature installed at end of first deployment
- **s.rval(obs)**: evaluates observation \(obs\) in current simulation state (\(obs\): “is a feature installed”, “number of simulation steps”, ...)
- **#**: triggers the execution of a simulation step
- Each `eval` is a property (18 for \(P_1+P_2\)) analysed on same simulations
Parametric quantitative analyses of case study

Example properties

\((P_1)\) Average price, weight or load of a bike when it is first deployed

\((P_2)\) For each of the 15 primitive features (leaves), the probability to have it installed when a bike is first deployed

\((P_3)\) The probability for a bike to be disposed of (irreparable)

Parametric expression for \(P_1-P_3\) w.r.t. simulation steps \((19 \times 251\) properties\)

\[
\text{ObsAtStep}(\text{obs}, \text{st}) = \begin{cases} 
\text{s.rval}(\text{"steps"}) == \text{st} & \text{then s.rval(\text{obs})} \\
\#\text{ObsAtStep}(\text{obs}, \text{st}) & \text{else}
\end{cases}
\]

\text{eval parametric}(E[\text{ObsAtStep}(\text{"price"}, \text{st})], E[\text{ObsAtStep}(\text{"weight"}, \text{st})], E[\text{ObsAtStep}(\text{"load"}, \text{st})], E[\text{ObsAtStep}(\text{"y"}, \text{st})], E[\text{ObsAtStep}(\text{"r"}, \text{st})], \ldots, E[\text{ObsAtStep}(\text{"h"}, \text{st})], E[\text{ObsAtStep}(\text{"trashed"}, \text{st})], \text{st, 0, 2, 500});

Parametric MultiQuaTEx

- \text{ObsAtFD}: modified temporal operator for evaluation w.r.t. a specific step given as parameter (i.e. \text{st})
- \text{trashed}: fictitious feature installed after a bike has been disposed of
- \text{parametric}(..., \text{st}, 0, 2, 500): specify a range of values for the parameter, i.e. steps from 0 to 500, with an increment of 2
Parametric quantitative analyses of case study

Example properties

\((P_1)\) Average price, weight or load of a bike at varying of time

Initial configuration \((y + d = 200\,\€)\), pre-configuration (FACTORY) and customisation (DEPOT)
Example properties

\((P_2)\) For each of the 15 primitive features (leaves), the probability to have it installed \textit{at varying of time}.

Initial configuration \((P(y) = P(d) = 1)\), pre-configuration (FACTORY) and customisation (DEPOT).
Example properties

\((P_3)\) The probability for a bike to be disposed of at varying of time

Bikes rarely break, are often reparable and (C5) prohibits dumping even averagely priced bikes
Conclusions and future work

**FLan family (FLan, PFLan, QFLan)**
- Specify and verify both declarative and procedural aspects of SPLs
- Semantics neatly unifies static and dynamic feature configuration
- Implementation in Maude/Z3/MultiVeStA for SMT solving and SMC

**SPL challenges offered by QFLan**
- Model quantitative constraints over features (conditioning actions)
- Explicitly uninstall/replace a feature, e.g. due to its malfunctioning or the need to replace it with a new/better (version of the) feature

**First application of SMC to SPLs**
+ Combine simplicity of testing with formality and precision of (P)MC
± From prototype to tool: user-friendly editor for QFLan models and improved performance: Java implementation QFLan/Z3/MultiVeStA
? Evaluate the scalability of our approach (on SPL benchmark models)
QFLaner: reimplementation as Eclipse-based tool

M.H. ter Beek et al. (INRIA/CNR/DTU/IMT)  Statistical Model Checking for SPLs  ISoLA 2016 27 / 27