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)

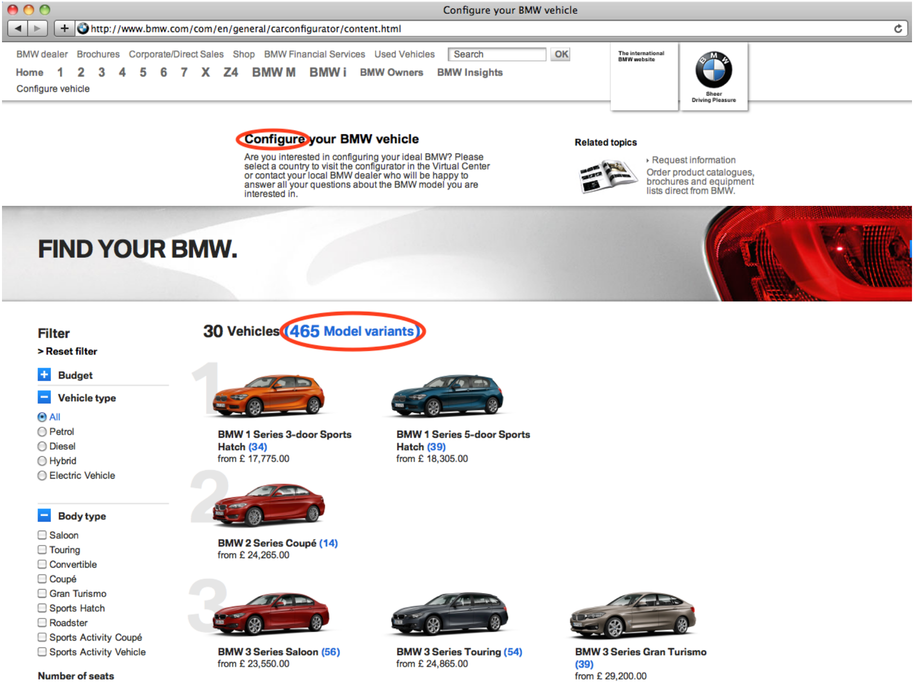
Software Product Line Engineering (SPLE)

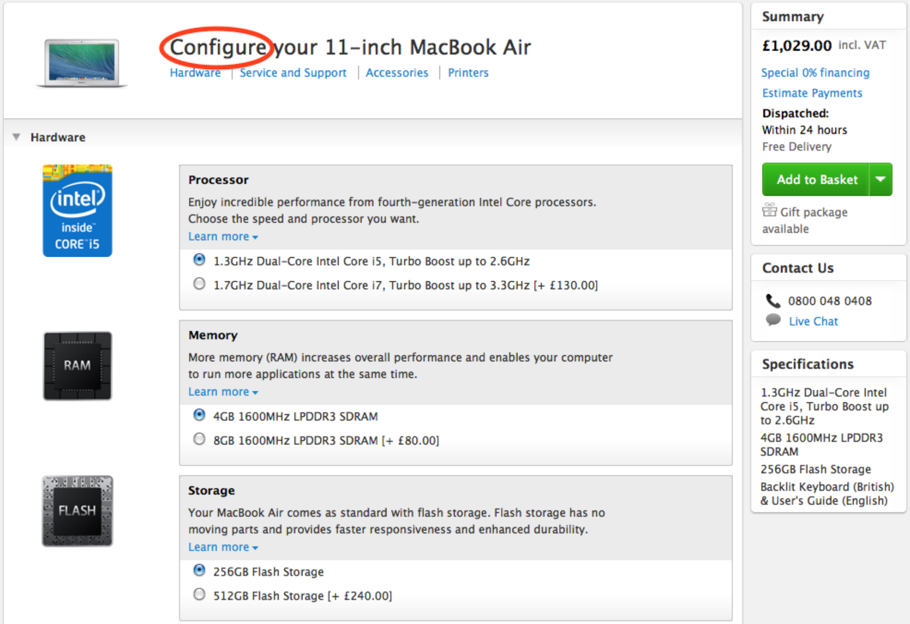
Develop and maintain a (software) product line using a shared architecture or platform (commonalities) and mass customisation (variabilities) to serve, e.g., different markets, thus allowing for (software) reuse

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

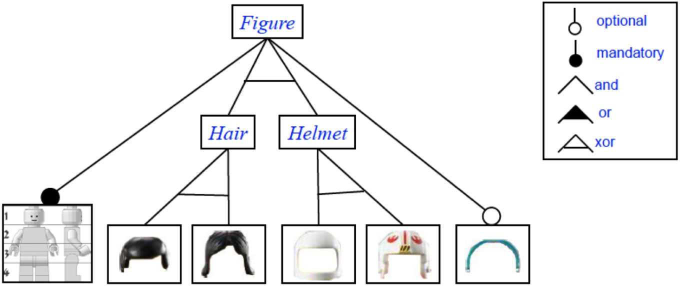

Configure your BMW vehicle
http://www.bmw.com/com/en/general/carconfigurator/content.html
BMW dealer Brochures Corporate/Direct Sales Shop BMW Financial Services Used Vehicles
Search OK
Home 1 2 3 4 5 6 7 X Z4 BMW M BMW i BMW Owners BMW Insights
Configure vehicle
The international BMW website
Sheer Driving Pleasure
Configure your BMW vehicle
Are you interested in configuring your ideal BMW? Please select a country to visit the configurator in the Virtual Center or contact your local BMW dealer who will be happy to answer all your questions about the BMW model you are interested in.
Related topics
Request information
Order product catalogues, brochures and equipment lists direct from BMW.
FIND YOUR BMW.
Filter
> Reset filter
Budget
Vehicle type
All
Petrol
Diesel
Hybrid
Electric Vehicle
Body type
Saloon
Touring
Convertible
Coupé
Gran Turismo
Sports Hatch
Roadster
Sports Activity Coupé
Sports Activity Vehicle
Number of seats
30 Vehicles (465 Model variants)
1
BMW 1 Series 3-door Sports Hatch (34)
from £ 17,775.00
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
BMW 3 Series Touring (54)
from £ 24,865.00
BMW 3 Series Gran Turismo (39)
from £ 29,200.00


Configure your 11-inch MacBook Air
Hardware | Service and Support | Accessories | Printers
Hardware
intel inside CORE i5
Processor
Enjoy incredible performance from fourth-generation Intel Core processors. Choose the speed and processor you want.
Learn more
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]
RAM
Memory
More memory (RAM) increases overall performance and enables your computer to run more applications at the same time.
Learn more
4GB 1600MHz LPDDR3 SDRAM
8GB 1600MHz LPDDR3 SDRAM [+ £80.00]
FLASH
Storage
Your MacBook Air comes as standard with flash storage. Flash storage has no moving parts and provides faster responsiveness and enhanced durability.
Learn more
256GB Flash Storage
512GB Flash Storage [+ £240.00]
Summary
£1,029.00 incl. VAT
Special 0% financing
Estimate Payments
Dispatched:
Within 24 hours
Free Delivery
Add to Basket
Gift package available
Contact Us
0800 048 0408
Live Chat
Specifications
1.3GHz Dual-Core Intel Core i5, Turbo Boost up to 2.6GHz
4GB 1600MHz LPDDR3 SDRAM
256GB Flash Storage
Backlit Keyboard (British) & User's Guide (English)

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



Visor $\Rightarrow$ *Helmet*

Feature model: compact representation of all products of a product line


LEGO
minifigures
Minifigures.LEGO.com

LEGO
minifigures
Minifigures.LEGO.com

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

$\Rightarrow$ 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

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

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

Specify quantitative constraints of SPL models

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

$\mathcal{P}_\mathcal{F}$ feature set of product $\mathcal{P}$

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

$\rightarrow \subseteq \mathbb{N}^{\mathbb{F}\times\mathbb{R}^+\times\mathbb{F}}$, with $\mathbb{F}$ set of all terms generated by F

$$(\textsc{Inst})\ \frac{\mathit{consistent}(S\ \mathit{has}(f))}{[S\ \neg\mathit{has}(f) \mid (\mathsf{install}(f), r).P] \xrightarrow{r} [S\ \mathit{has}(f) \mid P]} \qquad (\textsc{Act})\ \frac{S = (\mathit{do}(a) \rightarrow K) \qquad S \vdash K}{[S \mid (a, r).P] \xrightarrow{r} [S \mid P]}$$

$$(\textsc{Unst})\ \frac{\mathit{consistent}(S\ \neg\mathit{has}(f))}{[S\ \mathit{has}(f) \mid (\mathsf{uninstall}(f), r).P] \xrightarrow{r} [S\ \neg\mathit{has}(f)\ \mid P]} \qquad (\textsc{Ask})\ \frac{S \vdash K}{[S \mid (\mathsf{ask}(K), r).P] \xrightarrow{r} [S \mid P]}$$

$$(\textsc{Rpl})\ \frac{\mathit{consistent}(S\ \neg\mathit{has}(f)\ \mathit{has}(g))}{[S\ \mathit{has}(f)\ \neg\mathit{has}(g)\ \mid (\mathsf{replace}(f,g), r).P] \xrightarrow{r} [S\ \neg\mathit{has}(f)\ \mathit{has}(g) \mid P]}$$

$$(\textsc{Or})\ \frac{[S \mid P] \xrightarrow{r} [S' \mid P']}{[S \mid P + Q] \xrightarrow{r} [S' \mid P']} \qquad (\textsc{Seq})\ \frac{[S \mid P] \xrightarrow{r} [S' \mid P']}{[S \mid P; Q] \xrightarrow{r} [S' \mid P'; Q]} \qquad (\textsc{Par})\ \frac{[S \mid P] \xrightarrow{r} [S' \mid P']}{[S \mid P \parallel Q] \xrightarrow{r} [S' \mid P' \parallel Q]}$$

DTMC semantics

- normalising rates into $[0..1]$ such that $\forall$ states s : $\sum s \xrightarrow{\ell} = 1$
- transition 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

Statistical Model Checking

Model Checking (MC)

- Automatically check whether a model satisfies a temporal logic property (LTL, CTL) and provide a counterexample if it doesn't
- Exhaustive, but suffers from state space explosion problem
- BLAST, CADP, JPF, mCRL2, (Nu)SMV, SPIN, UPPAAL, ...

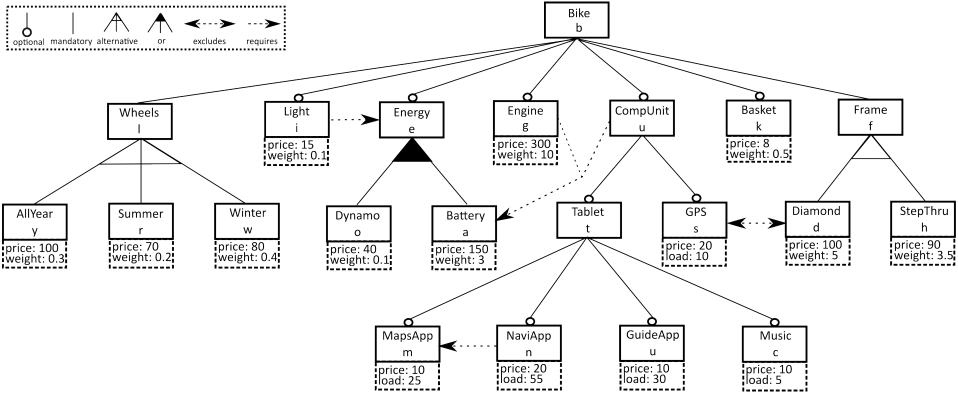
Probabilistic Model Checking (PMC)

- Model check whether a stochastic model satisfies a temporal logic property (PCTL, CSL) with a probability greater than a certain threshold
- Model uncertainty/performance; do quantitative analysis (QoS, ...)
- CADP, LiQuor, MRMC, PARAM, PRISM, UPPAAL-PRO, ...

Statistical Model Checking (SMC)

- Simulation-based technique to statistically approximate (P)MC
- Highly parallelisable and automatable; tunable preciseness via CI
- APMC, PLASMA, PRISM, UPPAAL, (P)VeStA, MultiVeStA, YMER, ...

Structural constraints of product line of bikes



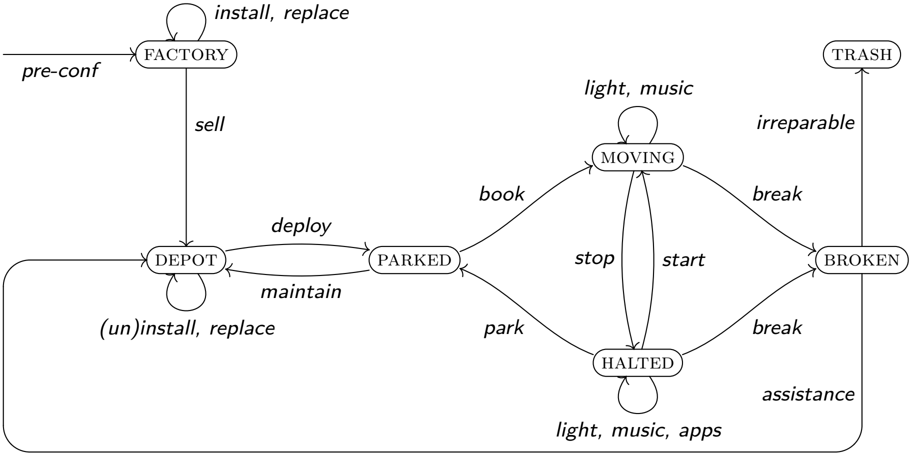
Additional arithmetic constraints

(C1) $\sum_{f \in \mathcal{P}_\mathcal{F}} \textit{price}(f) \leq 600$: a bike may cost at most 600 €

(C2) $\sum_{f \in \mathcal{P}_\mathcal{F}} \textit{weight}(f) \leq 15$: a bike may weigh up to 15 kg

(C3) $\sum_{f \in \mathcal{P}_\mathcal{F}} \textit{load}(f) \leq 100\%$: a bike's total computational load may not exceed 100%

Behavioural constraints of product line of bikes



Additional action constraints

(C4) $do(sell) \rightarrow \sum_{f \in \mathcal{P}_{\mathcal{F}}} price(f) \geq 250$

(C5) $do(irreparable) \rightarrow \sum_{f \in \mathcal{P}_{\mathcal{F}}} price(f) \leq 400$

QFLAN specification

$$
\begin{aligned}
FR &\doteq [\, S \mid F \,] \\
S &\doteq DS\ PS\ QS\ AS\ IS \\
DS &\doteq \ldots \quad PS \doteq \ \ldots \quad QS \doteq \ \ldots \quad AS \doteq \ \ldots \quad IS \doteq \ \ldots \\
F &\doteq (\mathit{sell}, 7).D \qquad \textit{// Installing optional features:} \\
&+ (\text{install}(s), 6).F + (\text{install}(m), 10).F + (\text{install}(n), 6).F + (\text{install}(u), 3).F + (\text{install}(c), 20).F \\
&+ (\text{install}(g), 4).F + (\text{install}(a), 5).F + (\text{install}(o), 10).F + (\text{install}(i), 10).F + (\text{install}(k), 8).F \\
&\quad \textit{// Replacing mandatory and exclusive features:} \\
&+ (\text{replace}(y, r), 5).F + (\text{replace}(y, w), 5).F + (\text{replace}(r, y), 10).F + (\text{replace}(r, w), 5).F \\
&+ (\text{replace}(w, y), 10).F + (\text{replace}(w, r), 5).F + (\text{replace}(d, h), 3).F + (\text{replace}(h, d), 3).F \\
D &\doteq (\mathit{deploy}, 10).P \\
&\quad \textit{// Installing optional features:} \\
&+ \ldots \textit{same as } F \\
&\quad \textit{// Uninstalling optional features:} \\
&+ \ldots \textit{same features and rates as installing, except for:} \\
&+ (\text{uninstall}(g), 1).D + (\text{uninstall}(a), 2).D + (\text{uninstall}(o), 3).D \\
&\quad \textit{// Replacing mandatory and exclusive features:} \\
&+ \ldots \textit{same as F, but Frame cannot be changed} \\
&\quad \textit{// Replacing battery by dynamo in case no features requiring a Battery are installed:} \\
&+ (\text{replace}(a, o), 1).D \\
P &\doteq (\mathit{book}, 10).M + (\mathit{maintain}, 1).D \\
M &\doteq (\mathit{stop}, 5).H + (\mathit{break}, 1).B + (c, 20).M + (i, 20).M \\
H &\doteq (\mathit{start},5).M + (\mathit{break},1).B + (\mathit{park},5).P + (c,20).H + (i,10).H + (s,10).H + (u,10).H \\
&\qquad + (m,10).H + (n,10).H \\
B &\doteq (\mathit{assistance}, 10).D + (\mathit{irreparable}, 1).T \\
T &\doteq (\text{install}(\mathit{trashed}), 1).\emptyset
\end{aligned}
$$

Constraints in the specification

$FR \doteq [\, S \mid F \,]$
$S \doteq DS\ \ PS\ \ QS\ \ AS\ \ IS$
$DS \doteq \ldots \quad PS \doteq \ \ldots \quad QS \doteq \ \ldots \quad AS \doteq \ \ldots \quad IS \doteq \ \ldots$

FR is composed of process F and store S of constraint sets:

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

$DS \doteq y \otimes r \otimes w \quad d \otimes h \ \ldots \ g \triangleright a \quad n \triangleright m \ \ldots$

PS Predicates for attributes of concrete features in feature diagram

$PS \doteq \mathrm{price}(y) = 100 \quad \mathrm{weight}(y) = 0.3 \ \ldots \ \mathrm{price}(c) = 100 \quad \mathrm{load}(c) = 5 \ \ldots$

QS Quantitative constraints

$QS \doteq \mathrm{price}(b) \leq 800 \quad \mathrm{weight}(b) \leq 20 \quad \mathrm{load}(b) \leq 100$

AS Action constraints

$AS \doteq do(sell) \rightarrow (\mathrm{price}(b) \geq 250) \ \ldots \ do(c) \rightarrow has(c) \ \ldots$

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

$IS \doteq has(y) \quad has(d)$

A process in the specification

$$FR \doteq [\, S \mid F \,]$$
$$S \doteq DS\ PS\ QS\ AS\ IS$$
$$DS \doteq \ldots \quad PS \doteq \ \ldots \quad QS \doteq \ \ldots \quad AS \doteq \ \ldots \quad IS \doteq \ \ldots$$
$$\begin{aligned}F \doteq\ & (sell, 7).D \qquad \textit{//Installing optional features:}\\ &+ (\text{install}(s), 6).F + (\text{install}(m), 10).F + (\text{install}(n), 6).F + (\text{install}(u), 3).F + (\text{install}(c), 20).F\\ &+ (\text{install}(g), 4).F + (\text{install}(a), 5).F + (\text{install}(o), 10).F + (\text{install}(i), 10).F + (\text{install}(k), 8).F\\ &\quad \textit{//Replacing mandatory and exclusive features:}\\ &+ (\text{replace}(y, r), 5).F + (\text{replace}(y, w), 5).F + (\text{replace}(r, y), 10).F + (\text{replace}(r, w), 5).F\\ &+ (\text{replace}(w, y), 10).F + (\text{replace}(w, r), 5).F + (\text{replace}(d, h), 3).F + (\text{replace}(h, d), 3).F\end{aligned}$$

F implements FACTORY's behaviour as a *weighted* choice among:

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

QFLan'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/`
- 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 δ
  (i.e. if an expression is estimated as $\overline{x}$, then with probability $(1-\alpha)$ its actual expected value belongs to the interval $[\overline{x} - {}^{\delta}/_{2}, \overline{x} + {}^{\delta}/_{2}]$)
- Parametric properties to perform many analyses, reusing simulations

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)

		Attributes (P_1)			Features (P_2)						
C1	C2	price	weight	load	y	r	$\cdots$	g	$\cdots$	d	h
600	15	391.91	7.80	33.50	0.57	0.24	$\cdots$	0.0	$\cdots$	0.61	0.39
800	20	509.83	11.98	34.45	0.54	0.23	$\cdots$	0.40	$\cdots$	0.60	0.40

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 × 251 properties)

```
ObsAtStep(obs,st) = if {s.rval("steps") == st} then s.rval(obs)
                                                 else #ObsAtStep(obs,st) fi;
eval parametric(E[ObsAtStep("price",st)], E[ObsAtStep("weight",st)],
E[ObsAtStep("load",st)], E[ObsAtStep("y",st)], E[ObsAtStep("r",st)], ...,
E[ObsAtStep("h",st)], E[ObsAtStep("trashed",st)], st, 0, 2, 500);
```

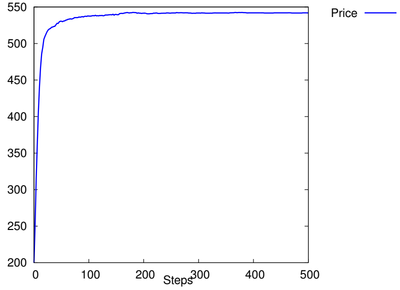
Parametric MultiQuaTEx

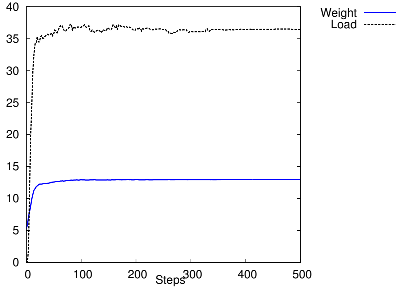
- `ObsAtFD`: modified temporal operator for evaluation w.r.t. a specific step given as parameter (i.e. `st`)
- `trashed`: fictitious feature installed after a bike has been disposed of
- `parametric(..., st, 0, 2, 500)`: specify a range of values for the parameter, i.e. steps from 0 to 500, with an increment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

Parametric quantitative analyses of case study

Example properties

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



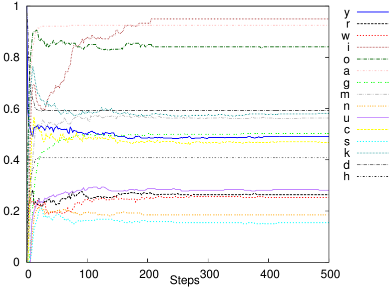


Initial configuration ($y + d = 200$ €), pre-configuration (FACTORY) and customisation (DEPOT)

Parametric quantitative analyses of case study

Example properties

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

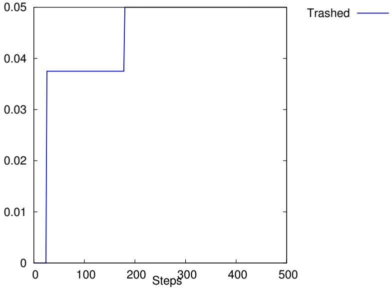


Initial configuration ($P(y) = P(d) = 1$), pre-configuration (FACTORY) and customisation (DEPOT)

Parametric quantitative analyses of case study

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

$\pm$ 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