ABSTRACT

We investigate the suitability of statistical model checking for the analysis of probabilistic models of software product lines with complex quantitative constraints and advanced feature installation options. Such models are specified in the feature-oriented language QFLan, a rich process algebra whose operational behaviour interacts with a store of constraints, neatly separating product configuration from product behaviour. The resulting probabilistic configurations and behaviour converge seamlessly in a semantics based on DTMCs, thus enabling quantitative analyses ranging from the likelihood of certain behaviour to the expected average cost of products. This is supported by a Maude implementation of QFLan, integrated with the SMT solver Z3 and the distributed statistical model checker MultiVeStA. Our approach is illustrated with a bikes product line case study.

CCS Concepts

• Theory of computation → Probabilistic computation; Process calculi; • Software and its engineering → Model checking; Software product lines;

1. INTRODUCTION

There is a lot of recent research on lifting successful high-level algebraic modelling languages and formal verification techniques known from single (software) system engineering, like process calculi and model checking, to (software) product line engineering (SPLE), cf. [1,12]. The challenge is the inherent variability: the number of possible products of a SPL may be exponential in the number of features. In [4], we contributed the feature-oriented language FLan and its implementation in Maude, allowing analyses ranging from consistency checking (by SAT solving) to model checking.

Research supported by EU project QUANTICOL (600708) and MIUR project CINA (PRIN 2010LHT4KM). We thank Biciniatta and M. Bertini from PisaMo for the case study and D. Lucanu, G. Rosu, A. Stefanescu and A. Arusoaie for sharing their Maude/Z3 integration, which we adapted.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SPLC 2015, July 20 - 24, 2015, Nashville, TN, USA © 2015 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-3613-0/15/07... $15.00

DOI: http://dx.doi.org/10.1145/2791060.2791087

In FLAN, a rich set of process algebra operators allows to specify the configuration and behaviour of products, while all common feature model constraints plus additional action constraints typical of feature-oriented software development can be specified in a ‘store’. A process’ execution is constrained by the store (to avoid introducing inconsistencies), which can also be queried (to resolve configuration options) or updated (to install new features, even at run time).

In [3], we proposed PFLAN, enriching FLAN with means to specify probabilistic models of SPLs. A clean separation between configuration and run-time aspects is a distinguishing feature of FLAN. PFLAN adds the possibility to equip actions with rates, representing uncertainty, failure rates, randomisation or preferences. An executable implementation in Maude and the distributed statistical model checker MultiVeStA [11] allow to estimate the likelihood of specific configurations and behaviour of an SPL, and thus to measure non-functional aspects like QoS, reliability or performance.

An emergent fact of our investigations is the urgent need to consider a number of further aspects in the specification and analysis of behavioural models of SPLs, like the staged configurations known from dynamic software product lines (e.g. removal of features) and quantitative constraints (e.g. price constraints). Currently, there are no approaches dealing with such aspects in a unifying framework [12]. To the best of our knowledge, [3] contains the only other application of statistical model checking in SPLE and [7] is the only approach (in addition to ours) to the model checking of SPL models with quantitative constraints over feature attributes.

Here we enrich PFLAN with the possibility to uninstall/replace features at run time and with quantitative constraints modelling the ‘cost’ of features, i.e. attributes related to non-functional aspects like price/weight. The novel options are:

1. Arithmetic relations among feature attributes (e.g. a feature set’s total cost must be less than a threshold);

2. Propositions relating a feature’s absence/presence to a constraint as in 1 (e.g. in presence of feature f, a feature set’s total cost must be less than a threshold);

3. Richer action constraints involving constraints as in 1 (e.g. an action can be performed only if the total cost of a product’s feature set is less than a threshold).

The new language is called QFLAN. The uninstallation and replacement of features can be the result of malfunctioning or of the need to install a better version of the feature (e.g. a software update). We remark that the above type of quantitative constraints are significantly more complex than the ones commonly associated to attributed feature models.
As feature attributes are typically not Boolean [7], the problem of deciding whether a product satisfies an attributed feature model with quantitative constraints, requires more general satisfiability-checking techniques than SAT solving. This naturally leads to the use of SMT solvers like Z3 [8], which allow to deal with richer notions of constraints such as arithmetic ones. In fact, an important contribution of this paper is the integration of SMT solving into our approach by combining our Maude QFLAN interpreter and Z3.

Statistical model checking (SMC) offers advantages over exhaustive (probabilistic) model checking. First, there is no need to generate entire state spaces, so SMC scales better without suffering from the combinatorial state-space explosion typical of model checking. In particular in the context of SPLs, with possibly exponential numbers of products, this outweighs the disadvantage of not obtaining exact results (100% confidence) as with (probabilistic) model checking. Second, SMC scales better with hardware resources since the simulations to be run can be parallelised and distributed. Our approach is to perform a sufficient number of probabilistic simulations of an SPL model to obtain statistical evidence (with a predefined level of statistical confidence) of the quantitative properties being verified. Properties are formulated in MultiVeStA's property specification language MultiQuaTEx [11]. MultiVeStA can run on multi-core machines, clusters or distributed computers with almost linear speedup. Moreover, its unique advantage is that it can use the same set of simulations to check several properties at once, thus offering further reductions of computing time.

**Outline**: §2 contains a bikes product lines case study; §3 presents QFLAN; §4 contains the case study's QFLAN model. §5 presents analyses of the case study. §6 concludes.

2. **BIKES PRODUCT LINE CASE STUDY**

Fig. 1 shows an attributed feature model of a product line of 1,314 bikes (without attributes). Quantitative constraints over attributes reduce the number of products, but not enough to mitigate the inherent exponential explosion. Each feature is equipped with a set of non-functional attributes, like price, weight or load, representing the feature’s price, weight (kg), and computational load, respectively. The set of features is $F = \{ b, l, i, c, g, u, k, f, y, r, w, o, a, t, s, d, h, m, n, u, c \}$. A product $P$ from the product line is a non-empty subset $P \subseteq F$ that fulfils the additional quantitative constraints defined over features and attributes. These can range from simple constraints (e.g. $price(u) \leq 20$, i.e. the price of a computational unit should be less than 20 €) to complex ones (e.g. $g \notin P \rightarrow \sum_{f \in P} weight(f) \leq 10$, i.e. if a bike does not have an engine then it cannot weigh more than 10 kg). Without such constraints, deciding whether a product satisfies a feature model is efficiently computed with SAT solvers, but in this paper we specifically allow quantitative constraints, which means we need SMT solvers.

For our case study, we consider the following constraints:

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

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

(C3) $\sum_{f \in P} load(f) \leq 100\%$: a bike’s total computational load may not exceed 100%.

(C1)–(C3) are part of the constraint store of our QFLAN model of the case study: any action (e.g. the run-time (un)installation or replacement of features) that would violate these constraints is prohibited, since its execution would lead to an inconsistent store. The store also contains action constraints which specify the subset of actions that they affect:

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

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

These are used in the behavioural part of our model to forbid selling bikes that cost less than 250 € (C4) or dumping broken, irreparable bikes that cost more than 400 € (C5).

Behaviour is probabilistic in the sense that in the presence of several enabled actions some may occur with a higher likelihood than others, to model the uncertainty of the behaviour of a bike, its components, and its interacting environment (users, exploiters, road conditions, etc.). Fig. 2 sketches the behaviour of our bikes product line without the action rates. 
Some typical properties of interest on the case study are:

\( P_1 \) Average bike price, weight or load when first deployed;

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

\( P_1 \) and \( P_2 \) are useful for studying an initial scenario, to estimate the required initial investments and infrastructure. Bikes with a high price and load (i.e. with a high technological footprint) or equipped with a battery might, e.g., require docking stations with specific characteristics or have to be collected for the night for safe storage. Instead, analysing docking stations with specific characteristics or have to be collected for the night for safe storage. Instead, instead of \( P_1 \) and \( P_2 \) over time provides an indication of how those values evolve, e.g. to estimate the average value in euros of a deployed bike and the financial consequences of its loss. In general, properties like \( P_2 \) measure how often (on average) a feature is actually installed in a product, which is important information for those responsible for the production or programming of a specific feature or software module.

3. SYNTAX AND SEMANTICS OF QFLAN

QFLAN is a process algebra promoting the separation of declarative (pre-)configuration from procedural run-time aspects. The FLAN family (FLAN [4], PFLAN [3], QFLAN) is inspired by the concurrent constraint programming paradigm of \cite{10} and its stochastic extension \cite{5}. All constraints known from feature models (and more) can be specified declaratively in a store, while product line configurations and behaviour can be specified procedurally in terms of a rich set of process algebra operators. The semantics unifies static (pre-configuration) and dynamic (run-time) feature selection.

QFLAN’s core notions are features from \( F \), constraints from \( K \) (using predicates from \( P \) and arithmetic expressions of feature attributes from \( E \)) collected in a store \( S \) and processes \( P \) (with actions from \( A \)), cf. its syntax in Fig. 3.

\[
\begin{align*}
F & ::= [S | P] \\
S,T & ::= K | S T | T | \bot \\
P,Q & ::= \emptyset | X | (A,r).P | P + Q | P;Q | P \parallel Q \\
A & ::= a | \text{install}(f) | \text{uninstall}(f) | \text{replace}(f,g) | \text{ask}(K) \\
K & ::= p | \neg K | K \lor V | E \bowtie E \\
E & ::= r | \text{attribute}(f) | E \pm E
\end{align*}
\]

Figure 3: QFLan syntax (with \( f, g \in F, r \in \mathbb{R}^+, a \in A, p \in P, \bowtie \in \{\leq, <, =, \neq, >, \geq\} \) and \( \pm \in \{+, -, \div, \times\} \).

We assume that the universe \( P \) of propositions contains a Boolean predicate \( has(f) \) that can be used to denote the presence of a feature \( f \) in a product. In our case study, e.g., \( \neg has(g) \) models a bike without an engine.

Quantitative constraints based on arithmetic relations between feature attributes are a QFLAN novelty. In our case study, e.g., a constraint \( \neg has(g) \rightarrow \sum_{f \in P} \text{weight}(f) \leq 10 \) would impose a weight bound on non-electric bikes.

As mentioned, QFLAN admits action constraints, reminiscent of featured transition systems (FTS) \cite{6}. In FTS, transitions are labelled with actions and Boolean constraints over the feature set. In QFLAN, arbitrary constraints are associated to actions rather than to transitions (together with a rate, discussed below). We assume that each action \( a \) may have a constraint \( do(a) \rightarrow p \), where \( p \in P \) is a proposition. Such constraints act as a kind of guards to allow/forbid the execution of actions (e.g. constraints (C4) and (C5) of §2).

Each action has an associated rate \( r \), used to determine the probability that it is executed. They allow to specify probabilistic aspects of SPL models like the behaviour of a user or the likelihood of installing a certain feature.

A constraint store \( S \) and a process \( P \) of a term \([S | P]\) may influence each other according to the concurrent constraint programming paradigm \cite{10}: a process may update its store which, in turn, may condition the execution of the process’ actions. For the sake of simplicity, in this paper we consider that \( S \) uniquely characterises a product of a product line (i.e. for each feature \( f \), \( S \) contains either \( has(f) \) or \( \neg has(f) \)).

QFLAN’s structural operational semantics is defined in Fig. 4, which defines a labeled transition system (LTS), with rates as labels. It is straightforward to obtain a discrete-time Markov chain (DTMC) from such an LTS by normalising the rates into \([0, 1]\). Such DTMCs are too large to be generated explicitly and that is why we prefer SMC which uses on-the-fly generated simulations of the DTMC. We advocate SMC because in general the DTMC is too large to generate.

4. BIKES PRODUCT LINE IN QFLAN

Fig. 5 sketches a QFLAN model of our bikes product line.

\( FS \) Constraints of the feature model in Fig. 1 (e.g. \( d \odot h \));

\( AS \) Action constraints as discussed in §2 and §3, like (C4);

\( QS \) Quantitative constraints affecting all actions, like (C2);

\( IS \) Pre-installed features AllYear/Diamond: \( has(g) \) \( has(d) \).

Process \( F \) implements the factory’s behaviour of Fig. 2 as a choice, weighted by the rates, among three main activities:

(1) With rate \( 7 \) the bike is sold and sent to the depot. This action can only be executed if (C4) is respected;

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

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

QFLAN’s semantics forbids re-installing (installed) features.

\( D \) (DEPOT) may execute deploy leading to \( P \) (PARKED) or uninstall features, to allow customisation. \( D \) can (un)install optional features with the same rate, except for Engine, Battery and Dynamo, which are uninstall with a lower rate to penalize their occurrence (a modelling choice justified by the fact that it is reasonable to assume that uninstalling might cost more than installing these features. We assume that a frame identifies a sold bike, so it cannot be modified in \( D \). The last action \( D \) can perform is interesting: replace Battery with a much cheaper Dynamo. QFLAN’s semantics prohibits this if a subfeature of CompUnit or Engine is still installed. This is useful to reduce costs/weight after a previously installed feature requiring a battery was uninstalled.

The remaining processes are faithful to Fig. 2 (\( T \) installs a fictitious feature \( trashed \) to express the fact that the bike has been disposed, and then evolve in the idle process).

Contrary to \( F \), \( D \) is not limited to (pre-)configuration. Parked bikes can be returned to the depot, so features can be (un)installed or replaced at run time. This is an example of staged configuration, in which some optional features are bound at run time rather than (pre-)configuration time.

The specification is completed with the features attributes depicted in Fig. 1, not shown here. The interested reader can consult the full specification in an extension of this work \cite{2}.
Figure 4: QFLan semantics in terms of $\mathbb{N}^{2\times \mathbb{R}^+\times \mathbb{R}}$, where F is the set of all terms generated by F in Fig. 3

Figure 5: QFLan specification of bikes product line

5. BIKES PRODUCT LINE ANALYSES

Our analyses were performed with MultiVeStA [9, 11], a distributed statistical model checker that was co-developed and is being maintained by A. Vandin. The integration of MultiVeStA and QFLan is available via [2] together with all files necessary to reproduce the experiments discussed here.

MultiVeStA’s property specification language MultiQuaTEx is very flexible, based on the following ingredients: real-valued observations on the system states (e.g. the total cost of installed features), arithmetic expressions and comparison operators, if-then-else statements, a one-step next operator (which triggers the execution of one step of a simulation) and recursion. Intuitively, MultiQuaTEx can associate a value from $\mathbb{R}$ to each simulation and subsequently MultiVeStA can estimate the expected value of such number (in case it is 0 or 1 upon the occurrence of a certain event, we thus estimate the probability of such an event to happen).

We obtain probabilistic simulations of a QFLan model by executing it step-by-step applying the rules of Fig. 4, each time selecting one of the computed one-step next-states according to the probability distribution resulting from normalising the rates of the generated transitions (cf. §3).

Classical SMC allows to perform analyses like “is the probability that a property holds greater than a given threshold?”. In addition, MultiVeStA allows to estimate the average costs/weight/load of products configured according to an SPL specification?”. Estimations are computed as the mean of $n$ samples obtained from $n$ independent simulations, with $n$ large enough to grant that the size of the $(1-\alpha) \times 100\%$ confidence interval (CI) is bounded by $\delta$.

In other words, if a MultiQuaTEx expression is estimated as $\tau \in \mathbb{R}$, then with probability $(1-\alpha)$ its actual expected value belongs to $[\tau - \delta/2, \tau + \delta/2]$. In our experiments we set $\alpha = 0.1$, and $\delta = 20.0$ for costs, $\delta = 1.0$ for weights, $\delta = 5.0$ for loads, $\delta = 1.0$ for steps, and $\delta = 0.1$ for probabilities.

We first evaluated $P_1$ and $P_2$ against the model of §4. The results are shown in the first row of Table 1. Notably, the 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). We guess that this is due to (C1) and (C2), imposing to bikes to cost less than 600 €, and weighing less than 15 kg. In fact, the estimated average price and weight of bikes at first deploy is 391.91 € and 7.8 kg, resp., while engine costs 300 € and weighs 10 kg. We confirmed this hypothesis by analysing $P_1$ and $P_2$ in a new model where (C1) and (C2) allow bikes to cost up to 800 € and weigh up to 20 kg, as shown in the second row of Table 1. The analysis thus reveals that the constraints are in disagreement with the quantitative attributes of the features.

Next, we evaluated variants of $P_1$ and $P_2$ at the varying of time, showing how MultiVeStA can analyse properties upon varying parameters, in this case the number of performed simulation steps. We used the model in which (C1) and (C2) bound the price and weight of a bike to 800 € and 20 kg, respectively. All analyses (19 × 251 properties) were evaluated using the same simulations. The results are shown in Fig. 6 for prices (left), for weights and loads (middle) and for the probabilities of installing features (right).

Fig. 6(left) shows that the average price (on the y-axis) of the intermediate bikes generated from the product line starts at 200 €, in line with the initial configuration (IS, with AllYear and Diamond installed). Then the price grows with respect to the number of performed simulation steps. In particular, it is possible to see an initial fast growth until reaching an average price of about 510 €, after which the growth slows down, reaching about 537 € at step 100 and 542 at step 500. This is consistent with our QFLan specification, which has a pre-configuration phase (FACTORY) during which a number of features can be installed, followed by a customisation phase (DEPOT), where features can be (un)installed and replaced. We recall that FACTORY does not perform any uninstalling, while we note that the uninstalling actions of DEPOT do not introduce decrements of the price, on average. A manual inspection of the data revealed that the phase of fast growth terminates about 19 steps. This is consistent with the analysis described in the second row of Table 1, where the average number of steps to complete the first DEPOT phase is estimated as being close to 19. In addition, the average price at the end of such a
phase is estimated to be around 510 €, as in Table 1. Note, finally, that the probability of a bike to return to the depot after its first deployment is quite low. In fact, as specified in Fig. 5, PARKED has a transition with rate 10 towards MOVING and one with rate 1 towards DEPOSIT. Thus, in average, the price of bikes is only slightly affected by (un)installations and replacements performed by successive depot phases.

As confirmed by Fig. 6(right), the probabilities (on the y-axis) for each feature to be installed evolve similarly to the average price, weight and load of the generated products. Notably, the pre-installed features AllYear (y) and Diamond (d) have probability 1 of being installed at step 0, after which the probability decreases during the first 19 steps.

6. CONCLUSIONS AND FUTURE WORK

We recently presented the probabilistic feature-oriented language PFLAN [3]. In this paper, we introduced QFLAN, which extends PFLAN with dynamic uninstallation and replacement of features and with advanced quantitative constraint modelling options, thus allowing for more involved quantitative analyses (requiring SMT solving). We achieved this by integrating an efficiently executable Maude implementation of QFLAN with Z3 and with the distributed statistical model checker MultiVeStA. We applied the resulting modelling and analysis framework to a bikes product line case study taken from companies with whom we cooperate in the context of the European project QuantiCoL. Our analyses revealed some interesting properties of the model, like the existence of a disagreement among constraints imposed on the price and weight of bikes, and prices and weights of bike components, as well as the high probability of replacing some features that tend to appear in initial configurations, which suggest to prioritise their installation in the early stages of the configuration. All in all, our detailed analysis served to validate our methodology and its tool support. Hence, our work provides a further contribution towards the adoption of formal specification and analysis techniques in SPL

In future work we plan to further develop the integration of Z3 with MultiVeStA, e.g. to equip our toolset with optimisation capabilities, so that users can not only validate configuration choices but also automatically obtain configuration options optimising their objective functions.

7. REFERENCES


