Summary of: A framework for quantitative modeling and analysis of highly (re)configurable systems

M. H. ter Beek[1][0000−0002−2930−6367], A. Legay[2], A. Lluch Lafuente[3][0000−0001−7405−0818], and A. Vandin[3][0000−0002−2606−7241]

[1] ISTI–CNR, Pisa, Italy
[2] UCLouvain, Louvain-la-Neuve, Belgium
[3] DTU, Lyngby, Denmark

Abstract. This short paper summarises the contributions published in [4], where we introduce QFLan, a framework for quantitative modeling and analysis of highly (re)configurable systems, like software product lines. We define a rich domain specific language (DSL) for systems with variability in terms of features, which can be dynamically installed, removed or replaced, capable of modeling probabilistic behavior, possibly subject to quantitative feature constraints. High-level DSL specifications are automatically encoded in a process algebra whose operational behavior interacts with a store of constraints, which allows to separate a system's configuration from its behavior. The resulting probabilistic configurations and behavior converge seamlessly in a semantics based on discrete-time Markov chains, thus enabling quantitative analysis. An accompanying Eclipse-based tool offers a modern integrated development environment to specify such systems and to perform analyses that range from the likelihood of specific behavior to the expected average cost, in terms of feature attributes, of specific system variants. Based on a seamless integration with the statistical model checker MultiVeStA, QFLan allows to scale to larger models with respect to precise probabilistic analysis techniques. We provide a number of case studies that have driven and validated the development of the QFLan framework. In particular, we show the versatility of the QFLan framework with an application to risk analysis of a safe lock system from the security domain.

1 Extended Abstract

In [1, 2], we presented various facets of the probabilistic modeling language QFLan, which is capable of describing a wide spectrum of aspects of (software) product lines (SPL) or configurable (software) systems in general. After a survey in the invited contribution [3], we provide a comprehensive presentation of the QFLan framework, consisting of a high-level modeling language with advanced tool support, in [4] and an accompanying tool paper [7]. Moreover, to illustrate the versatility of the QFLan framework, [4] contains case studies from different application domains. One of them concerns risk assessment in a security scenario with high variability that makes use of so-called attack trees. We show how to apply QFLan to the seminal example from that area: Schneier's Safe Lock [5, 6].

Figure 1 depicts the attack tree from [5]. It specifies a risk assessment for a safe lock system. An attack tree is essentially an and/or tree, whose nodes represent goals, and sub-trees represent sub-goals. In this case, the root represents the main threat under analysis, namely the lock being opened by an attacker. Each of its children are possible ways of enacting such threat. The sub-goal Eavesdrop has two sub-goals that need to be accomplished (thus their combination as *and*-children). Nodes are decorated with an estimation of the cost that the attacker would have to pay to succeed in enacting the corresponding action. The classical analysis of such trees is to compute the minimal cost for an attacker to succeed.

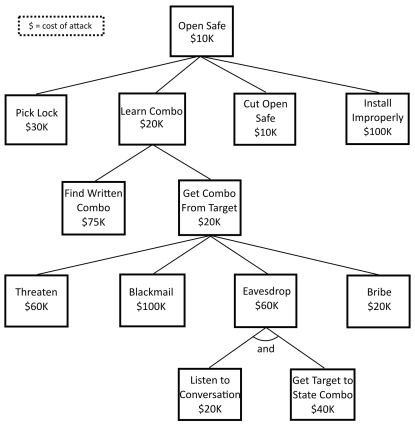

Fig. 1. Schneier's attack tree from [5]

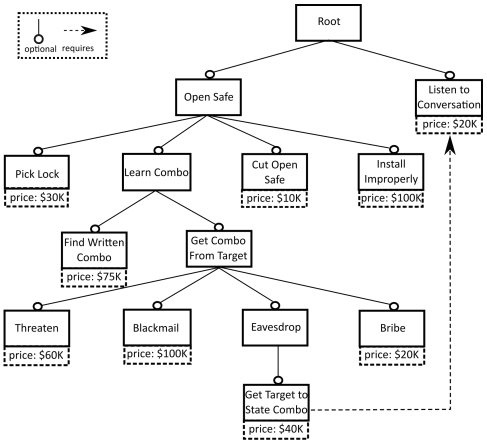

Fig. 2. Feature model representation

Attack trees can easily be modelled as so-called feature models, with the following rationale: a node, representing a goal, can be modeled as a feature of the system which the attacker tries to activate. The sub-goal relation is modeled by the feature hierarchy. The attack tree of Fig. 1 can be modeled as in Fig. 2.

We introduce a slight variation to overcome a well-known limitation of the original attack trees, namely the inability to encode the ordering of events. Indeed, Listen to Conversation should occur before Get Target to State Combo, which we can model with a *requires* cross-tree constraint. A feature model defines which configurations are valid, but not how (i.e., in which order) to configure them. QFLan does model (re)configuration: features can be dynamically installed, removed or replaced as long as at any point in time all constraints are satisfied, including those imposed by the feature model. As noted in [4], the requires cross-tree constraint from Get Target to State Combo to Listen to Conversation implies an order: whenever QFLan tries to install (i.e., the attacker tries to activate) Get Target to State Combo, it fails to do so unless Listen to Conversation was installed (i.e., activated) before. As a matter of fact, the flexibility of the way feature models are specified in QFLan allows us to specify richer relations among sub-goals. For instance, we can specify that Eavesdrop is only successful if the attacker first listens to a conversation and then gets the target to state the combo, thus refining the original *and*-relation among such sub-goals.

A notable advantage of using QFLan for such scenarios is that we can model attacker behavior and study the system's robustness against them. Consider the attackers sketched in Fig. 3 (cf. [4] for their process specifications). An attacker is specified in terms of states and transitions among them, each labeled with the performed action, a weight to probabilistically choose the transition to be executed, and optional variable updates.

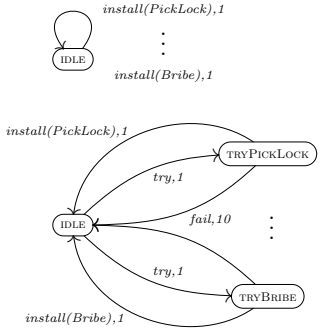

Fig. 3. PowerfulAttacker (top) and FailingAttacker (bottom)

Powerful attacker always succeeds in trying to achieve a goal and has unlimited resources. Failing attacker can fail, may need several attempts (modeled via weights) and has limited resources. Clearly, reasonable attackers stop attacking after a successful attack. This can be expressed in QFLan using two action constraints:

```
begin action constraints
  do(tryAction) -> !has(OpenSafe)        do(install(...)) -> !has(OpenSafe)
end action constraints
```

QFLan's rich specification language allows to express further constraints on the accepted classes of attacks. Consider the following two quantitative constraints:

```
begin quantitative constraints
  { cost(Root) <= 100 }                  { cumul_cost <= 20 }
end quantitative constraints
```

The first restricts to (successful) attacks that cost less than \$100K (i.e., install features with less than that price, cf. the feature model in Fig. 2). The second constraint instead restricts to attacks (independently of their success) that cumulated less than 20 attempts. Noteworthy, using the first constraint we restrict the family of admissible products, while the latter constraint regards only the behavioral part of the model. In fact, `cumul_cost` is not an attribute but a variable, which can be changed through a memory update in the behavior. We use it as a counter to record the number of times that an attack is tried.

For both attackers, we want to know the probability for an attack to succeed in a given amount of time and its average cost. QFLan can run such analyses by querying, at each of the first 40 simulation steps (`eval from 0 to 40 by 1`), the probability of installing the feature `OpenSafe`, the cost of the corresponding variant (cost(`Root`)) and the attempts cumulated by the failing attacker (`cumul_cost`) while trying to install the features corresponding to the sub-goals:

```
begin analysis
  query = eval from 0 to 40 by 1 : { OpenSafe[delta = 0.05] , cost(Root) ,
      cumul_cost }                       default   delta = 1   alpha = 0.05
end analysis
```

QFLan estimates these properties as the mean of n samples obtained from n independent simulations, with n large enough (but minimal) to grant that the size of the $(1-\alpha)\times 100\%$ *confidence interval* for the expected value is bounded by δ, i.e., a confidence interval is specified in terms of two parameters: α and δ.

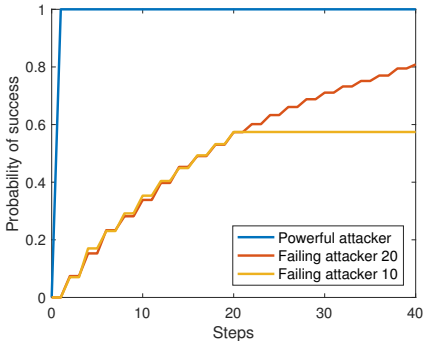

Fig. 4. Probabilities of successful attacks

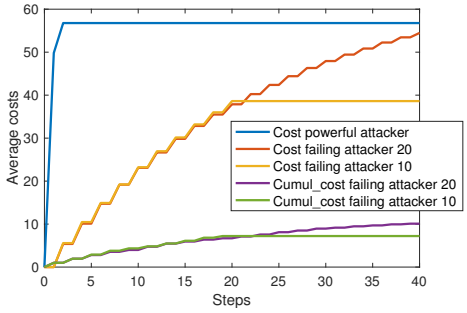

Fig. 5. Costs of successful attacks

We consider three configurations: (a) a powerful attacker with constraints as specified above; (b) an attacker that might fail with the same constraints; and (c) an attacker that might fail with less resources, obtained by changing the constraint on the `cumul_cost` to `{ cumul_cost <= 10 }`. This is obtained by running once the analysis on each model variant, each requiring about 12 seconds.

Figure 4 plots the probabilities of successful attacks. The powerful attacker succeeds with probability almost 1 after one step, but for the other attackers the probability of success increases slowly. For constraint `{ cumul_cost <= 10 }`, the probability of success stabilizes at ± 0.6 after 20 steps. Indeed, `cumul_cost` increases by 1 every two steps. Instead, for `{ cumul_cost <= 20 }`, the probability reaches 0.8 after 40 steps, but this is not due to the mentioned constraint. In fact, Fig. 5 plots the costs and cumulative attempts for the model variants. The average cumulative attempts for the failing attackers do not diverge enough to attribute the different dynamics of the two attackers to the mentioned constraint. Finally, note that costs evolve similarly to probabilities, but with different scales.

References

1. ter Beek, M.H., Legay, A., Lluch Lafuente, A., Vandin, A.: Quantitative Analysis of Probabilistic Models of Software Product Lines with Statistical Model Checking. EPTCS **182**, 56–70 (2015). https://doi.org/10.4204/EPTCS.182.5
2. ter Beek, M.H., Legay, A., Lluch Lafuente, A., Vandin, A.: Statistical Analysis of Probabilistic Models of Software Product Lines with Quantitative Constraints. In: SPLC. pp. 11–15. ACM (2015). https://doi.org/10.1145/2791060.2791087
3. ter Beek, M.H., Legay, A., Lluch Lafuente, A., Vandin, A.: Statistical Model Checking for Product Lines. In: ISoLA. LNCS, vol. 9952, pp. 114–133. Springer (2016). https://doi.org/10.1007/978-3-319-47166-2_8
4. ter Beek, M.H., Legay, A., Lluch Lafuente, A., Vandin, A.: A framework for quantitative modeling and analysis of highly (re)configurable systems. IEEE Trans. Softw. Eng. (2018). https://doi.org/10.1109/TSE.2018.2853726
5. Schneier, B.: Attack trees. Dr. Dobb's Journal (December 1999)
6. Schneier, B.: Secrets & Lies: Digital Security in a Networked World. Wiley (2000)
7. Vandin, A., ter Beek, M.H., Legay, A., Lluch Lafuente, A.: QFLan: A Tool for the Quantitative Analysis of Highly Reconfigurable Systems. In: FM. LNCS, vol. 10951, pp. 329–337. Springer (2018). https://doi.org/10.1007/978-3-319-95582-7_19