Highlights

White-box validation of quantitative product lines by statistical model checking and process mining

Roberto Casaluce, Andrea Burattin, Francesca Chiaromonte, Alberto Lluch Lafuente, Andrea Vandin

- Effective, scalable, and multi-domain methodology for the analysis of product line models
- Integration of statistical model checking and process mining applied to product line and security domains
- From black-box model validation to white-box novel validation and enhancement
- Mining of product line models

White-box validation of quantitative product lines by statistical model checking and process mining

Roberto Casaluce[a], Andrea Burattin[b], Francesca Chiaromonte[a,c], Alberto Lluch Lafuente[b], Andrea Vandin[a,b,*]

[a]*Institute of Economics and L'EMbeDS, Sant'Anna School of Advanced Studies, Pisa, Italy.*
[b]*DTU Technical University of Denmark, Lyngby, Denmark.*
[c]*Dept. of Statistics and Huck Institutes of the Life Sciences, Penn State University, USA*

Abstract

We propose a novel methodology to validate software product line (PL) models by integrating Statistical Model Checking (SMC) with Process Mining (PM). We consider the feature-oriented language QFLan from the PL engineering domain. QFLan allows to model PL equipped with rich cross-tree and quantitative constraints, as well as aspects of dynamic PLs such as the staged configurations. This richness allows us to easily obtain models with infinite state-space, calling for simulation-based analysis techniques, like SMC. For example, we use a running example with infinite state space. SMC is a family of analysis techniques based on the generation of samples of the dynamics of a system. SMC aims at estimating properties of a system like the probability of a given event (e.g., installing a feature), or the expected value of quantities in it (e.g., the average price of products from the studied family). Instead, PM is a family of data-driven techniques that uses logs collected on the execution of an information system to identify and reason about its underlying execution process. This often regards identifying and reasoning about process patterns, bottlenecks, and possibilities for improvement. In this paper, to the best of our knowledge, we propose, for the first time, the application of Process Mining (PM) techniques to the byproducts

*Corresponding author

Email addresses: roberto.casaluce@santannapisa.it (Roberto Casaluce), andbur@dtu.dk (Andrea Burattin), francesca.chiaromonte@santannapisa.it (Francesca Chiaromonte), albl@dtu.dk (Alberto Lluch Lafuente), andrea.vandin@santannapisa.it (Andrea Vandin)

of Statistical Model Checking (SMC) simulations. This aims to enhance the utility of SMC analyses.

Typically, if SMC gives unexpected results, the modeler has to discover whether these come from actual characteristics of the system, or from bugs in the model. This is done in a *black-box* manner, only based on the obtained numerical values. We improve on this by using PM to get a white-box perspective on the dynamics of the system observed by SMC. Roughly speaking, we feed the samples generated by SMC to PM tools, obtaining a compact graphical representation of the observed dynamics. This *mined PM model* is then transformed into a *mined QFLan model*, making it accessible to PL engineers. Using two well-known PL models, we show that our methodology is effective (helps in pinpointing issues in models, and in suggesting fixes), and that it scales to complex models. We also show that it is general, by applying it to the security domain.

Keywords: Software product lines, Product line engineering, Probabilistic modeling, Statistical model checking, Process mining, Attack-defense trees

---

1. Introduction

Software product lines (SPL), and feature models in general, as well as Product Line Engineering (PLE), play a very important role in modern society, where customization capabilities are expected even for commodity products. Very often, these products are equipped with software that is expected to follow the customization of the product itself. As a consequence, it becomes necessary to ensure that the product lines are properly designed and that the models indeed capture the intentions of the modelers. This paper presents a novel methodology to validate the behavior of SPL models by offering simple tools to "see and compare" the *actual* behavior of a model with the *expected* one.

To validate models that present quantitative aspects in their behavior, we often use exact or statistical analysis techniques. The formal verification of the dynamics of a system via exact techniques provides precise values of the (quantitative) properties being analyzed. These typically require reasoning upon the whole behavior of the system, which might not be feasible for complex models. Indeed, as the possible dynamics of the system increase, these techniques tend to suffer from the well-known state-space explosion problem, rendering them inapplicable when the state-space becomes infinite

(see, e.g., [1]). On the other hand, statistical analysis techniques, such as Statistical Model Checking (SMC) [2], rely only on limited but statistically relevant samples of executions of a model: simulations. Therefore, statistical analysis techniques can be used to analyze complex dynamical systems, potentially with infinite state spaces, at the cost that analysis results are not *exact* anymore but are only statistically reliable estimations, e.g., equipped with confidence intervals.

When the overarching behavior of a system is unknown, and it is impossible to make assumptions about its transition structure, the system is referred to as a black-box system. An SMC that analyzes the dynamics of a black-box system without prior knowledge of the system is referred to as a black-box SMC [3]. These simulation-based approaches return numerical estimates, plots, and occasionally counterexamples of the studied properties. However, they typically do not provide behavioral explanations for the results obtained. Without clear explanations, the modeler can only make informed guesses about how to adjust the model to fix unwanted behaviors. For example, let us assume that we consider an SPL model for a family of vending machines, a classic PLE model (see, e.g., [4, 5, 6, 7, 8, 9, 10]). Let us further assume that we use SMC to study the probability that machines from the family contain dispensers for cappuccino and that we get 0. Interesting questions about this analysis are:

- *What is the reason behind such an extreme value?*,
- *Was the model intended to express this dynamic, or is there a bug?*

In our view, the numeric value 0 is a black-box analysis result. Meaning that we do not know *why* we got 0, we do not know if it comes from an issue in the model, nor how to fix it. The core of our proposal is to enrich the analysis results obtained by SMC to study this query by automatically adding explicit visual information pinpointing any misalignment between the model and the two bullet points above. This approach not only facilitates model refinement but also serves as a method for conducting comprehensive testing. Experimenting with diverse settings within the same model structure enables the evaluation of the correctness of the simulated model.

Our proposal involves in enriching SMC analyses by conducting additional post-processing and analyzing the byproducts of SMC, i.e., log files on the computed simulations, using popular data-driven techniques known

as Process Mining (PM). Given that SMC is able to handle models with infinite state-space, our approach can similarly address such scenarios. PM is a process-oriented data-driven technique that analyzes the executions (i.e., traces) of activities generated by information systems, allowing the identification of process patterns, bottlenecks, and other issues in a model [11]. Visualizing the flow of activities helps identify opportunities for improvement (e.g., unexpected loops or unexpected dependencies).

This paper, which extends a preliminary work [12] where we sketched the potential of enriching SMC techniques with PM, presents a white-box technique to enable the evaluation of behavioral aspects of a feature model. The technique leverages the application of process mining techniques on event logs produced by (simulations generated by) statistical model checking. The goal is to provide insights into the system behavior, such as discovering new patterns, identifying bottlenecks, and improving the general model accuracy. Therefore, integrating SMC with PM paves the way to a more comprehensive understanding of the overall behavior of the model and can help identify issues or suggest actionable improvements to the modeler. To the best of our knowledge, our methodology is the first attempt in automatically *explaining* the results of SMC using PM-based approaches. In our preliminary previous work [12], we exemplified the capabilities of analyzing traces generated by SMC with PM using a preliminary version of our methodology. However, the identification of issues was entirely delegated to the modeler's visual skills and hence highly subjective. Furthermore, the preliminary methodology had low accessibility, as the mined model was given using a PM formalism different from the one used to create the original model. This had the additional disadvantage that, in the presence of a large model, mining the simulations would result in a complex mined PM model, making it difficult for the modeler to locate issues. In this paper, we overcome these limitations by fully developing the methodology to *automatically* discover and visualize undesirable behaviors. Such findings are then shown directly in the model specification itself. This is accomplished by highlighting the differences between the expected behavior of the model, the model specification, and the actual behavior discovered by mining its simulations. In fact, by highlighting the specific behaviors causing issues in the model, the modeler can make more targeted and effective adjustments to the model to fix those issues. Furthermore, in [12] we did not tailor the SPL domain, but only the cybersecurity one, while we now explicitly target the SPL domain by applying the approach on the feature-oriented language QFLan [13, 4].

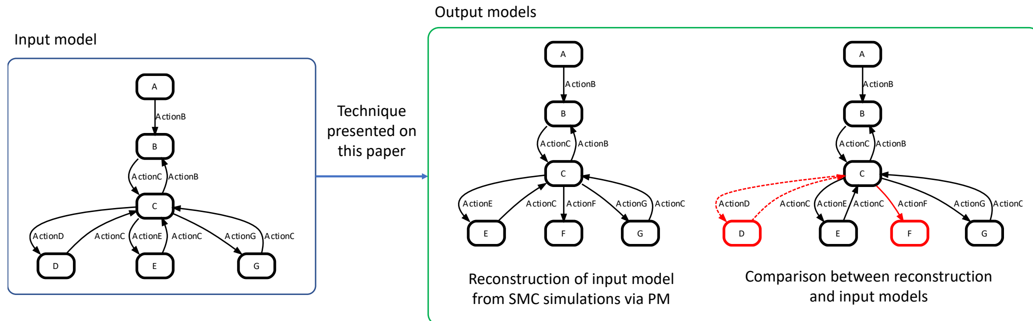

Figure 1: Example of the input and output produced by the technique presented in this paper. The input model is simulated with SMC techniques and corresponding traces are used to synthesize a new model which is then compared to the original one and an easy-to-read output is returned to the modeler emphasizing the differences.

Figure 1 illustrates an abstract example of a model validated using our method. The input model is a representation of an abstract model. This abstract model includes different states and actions used to move between those states. In this example, the simulator would start from the node `A` and with the `ActionB` move to the node `B`, and from there can move to another node by choosing the corresponding action. Without our methodology, an SMC user would need to validate the input model using only the obtained numerical results (e.g., if interested in "estimating the probability of reaching a specific node in the model"). With our approach, however, the modeler can inspect the results of the simulation once they are synthesized into new models, thus using the same language (i.e., models-to-model rather than models-to-numbers). The first graph in the "output models" box of Figure 1 depicts the reconstructed model after applying Process Mining to the simulations. Instead, the rightmost model depicts the graph obtained by comparing the input model and the reconstructed one. This final representation, in particular, highlights the differences between the input model and the simulated behavior, allowing the modeler to quickly identify issues, such as unexpected or missing behaviors. The exact semantics of the different colors of the edges is explained in Section 3. It is worth mentioning that this method can, in principle, be applied to any discipline where discrete-state simulation models are used, enhancing the capabilities of related modeling and analysis tools. On the other hand, the methodology is particularly useful for domains where the complexity of models is high, as in the case of highly-parametric models from PLE. In particular, SMC, and therefore our

approach, which post-processes its results, is particularly useful for models with very large or infinite state space models.

It is important to stress that the methodology depends on the input model, but also on the chosen studied property, as this will drive the simulation process. This is somehow reminiscent of the so-called *CEGAR* (counterexample guided abstraction and refinement [14]) approaches from qualitative model checking (and applied only in very limited way to probabilistic settings [15]). In classic qualitative model checking, if the studied property does not hold, we get counterexamples of systems' dynamics that showcase executions that falsify the formula. CEGAR involves the use of such counterexamples to refine the model. In our methodology, we proceed similarly, by using mined process models as counterexamples. This is orthogonal to static analysis approaches like, e.g., [16], which aim at identifying issues of the model in general, not tailored to the verification of a single property.

To validate our methodology we provide positive answers to the following research questions:

RQ1. *(effectiveness)* Can the developed techniques be employed for a comprehensive evaluation aimed at thoroughly studying the behavior of models and identifying any errors within them? To answer this question, we apply our methodology to the feature-oriented quantitative modeling and analysis framework QFLan [13, 4]. We demonstrate the effectiveness of the proposed method by applying it to a model describing a family of beverage Vending machines product line from [4], a classic case study in PLE. Our experiments, in more challenging settings of previous experiments [4], demonstrate the effectiveness of our method: We can automatically conduct a comprehensive evaluation of the behavior of the model.

RQ2. *(scalability)* Are the developed techniques scalable to large models considered challenging by the SPL community? To answer this question we apply our methodology to a case study of an Elevator product line from [13], initially proposed by [17], which is a well-known case study used to test the scalability in PLE. Furthermore, it is worth noting the variant of Vending machines product line considered in this paper is infinite state space. Our experiments show that our methodology tends to have a runtime in the same order of magnitude of SMC analysis, and it never exceeds more than 5 times its runtime.

RQ3. *(multi-domain)* Can the developed techniques generalize to further domains beyond software product lines? To answer this question we consider an additional domain, namely cybersecurity, using the framework RisQFLan [1]. It is an incarnation of QFLan to the cybersecurity domain. Thanks to this, we successfully validate our approach on an example of a threat model from [1], demonstrating its applicability to different domains. Indeed, we show how we automatically discovered unwanted and unexpected behaviors. In addition, we also got hints on how to fix such issues by obtaining a refined model that does not show the issues. Notably, as emphasized in [1], the considered model has infinite state-space, providing additional insights for addressing RQ2.

All models and replication material for this paper are available at `https://doi.org/10.5281/zenodo.8362717`.

Synopsis. The remainder of the paper is structured as follows. Section 2 introduces necessary background material, as well as the Vending machine as a running example. After this, Section 3 presents our methodology, while Sections 4- 6 validate it on three case studies, answering our research questions. Section 7 discusses related works, while Section 8 concludes the paper and drafts future works.

Further discussion regarding the relationship with [12]. this paper expands upon the preliminary research presented in that work. Specifically, within this paper: We generalized the approach from the security domain to the SPL one (we added native support for QFLan); We complete the methodology by computing automatically a *diff* model, given in the original model specification language, to highlight the differences between the reference and mined models; We evaluate the scalability, effectiveness, and generality of the approach via proper experiments; We consider a more complex security model; We added a related work section and an actual artifact to be used by third parties.

2. Background

This section presents the fundamental notions needed throughout the rest of the paper.

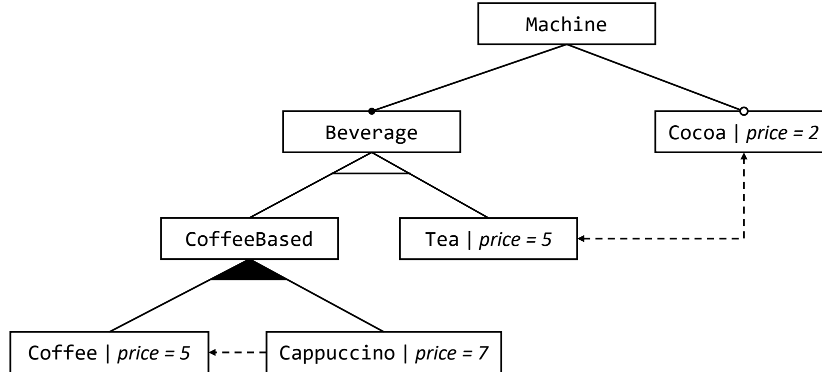

Figure 2: Feature model for hot beverage vending machines, figure adapted from [4]

2.1. Modeling product lines with QFLan

QFLan (Quantitative Feature-Oriented Language) is a feature-oriented language member of the FLan (Feature-Oriented Language) family [4, 13, 18]. It is based on the principles of concurrent constraint programming and is used to specify the configuration and behavior of product lines mixing procedural and declarative aspects. To achieve this, QFlan employs a constraint store to separate the declarative aspects of the model, e.g., the constraints imposed by a feature diagram, from procedural reconfiguration aspects typical of dynamic SPLs. In fact, QFLan can deal with aspects of dynamic SPLs such as the staged configurations known from dynamic SPLs [19, 20] (e.g., adding and removing features as well as activating and deactivating features at runtime). This allows the modeler to express typical constraints from feature models in a declarative manner. These two aspects are unified by the formal semantics of QFlan.

QFlan supports quantitative analysis, via the statistical analyzer MultiVeStA discussed in Section 2.2. QFLan has been recently recast for the security risk modeling domain, obtaining the language RisQFLan [1]. In this paper we consider both QFLan, to show that our methodology can be of interest to the PLE community, and RisQFLan, to show the multi-domain nature of our approach.

To ease the presentation of our ideas, we use as a running example a classic case study in PLE presented in [4]. This is an adaptation of several proposals from the literature, commonly used to present novel methodologies for PLE (e.g., [5, 6, 7, 8, 9, 10]). This is a classic example of a product line of vending machines that offer a selection of tea and coffee-based bev-

```
begin action constraints
// Machines serving Cappuccino and Cocoa dispensers can serve chocaccino
do(chocaccino) -> (has(Cappuccino) and has(Cocoa))
end action constraints
```

Figure 3: Action constraints

```
begin quantitative constraints
//The price of generated products must never exceed 10 (or 15)
//{ price(Machine) <= 10 }
{ price(Machine) <= 15 }
end quantitative constraints
```

Figure 4: Quantitative constraints

erages, including `Coffee`, `Cappuccino`, `Tea`, and `Chocaccino` (`Cappuccino` with `Cocoa`). Each product of this family is a concrete vending machine with an *admissible* subset of beverages. Figure 2 depicts graphically the *feature model* of our example. It describes the structural constraints among the *features* that may be present or not in valid instances of vending machines. Each node represents a feature, and the edges between nodes represent constraints that define the admissible combinations of features. QFLan has two types of features: concrete and abstract. The four leaves represent concrete features. These can be *installed* or *uninstalled* explicitly. Instead, abstract features are internal nodes. These are not explicitly (un)installed, rather they are implicitly added or removed when their children nodes are. Abstract features are mainly used to group related features. The root node represents a complete product, which in this case is a specific vending machine. The feature diagram imposes that concrete instances of vending machines may or may not contain the optional feature `Cocoa` (empty circle over it), but its presence excludes the possibility of serving teas (and vice-versa, the dashed line connecting the two). A concrete vending machine is required to have a beverage feature, which can be either coffee-based or `Tea` (the edges with an empty triangle connecting the three features). Coffee-based beverages can be `Coffee` or `Cappuccino`, with the *cross-tree constraint* that the latter need `Coffee` (the dashed arrow from `Cappuccino` to `Coffee`). QFLan models are actually given using a textual representation in a specific domain-specific language. The full model specification can be found in [4]. Here we provide the minimum information to make the paper self-contained.

We note that `Chocaccino` does not belong to the feature diagram in Figure 2. This is because it is not a feature to be (un)installed. Instead, it is a sort of implicit feature available when the machine can serve both `Cappuccino` and `Cocoa`. This is encoded by the *action constraint* in Figure 3. QFlan allows to consider further classes of constraints, including quantitative ones. E.g., the price of a sold vending machine could be limited to a maximum cost computed by summing the cost of currently installed concrete features (see Figure 4). As we will see in Section 4, these constraints considerably impact the dynamics of the model.

In addition to the discussed constraints, the declarative part, QFLan models come with a procedural part. This specifies the dynamic behavior of the model. Specifically, Figure 5 lists the probabilistic process of the Vending machine. This includes different states and transitions among them. Transitions must be labeled with weights, used to compute the probability of executing a transition and actions. Actions can be feature names, which signal the use of installed features, or custom actions (listed in Figure 6). We also note that transitions might be further labeled with *side-effects* which change the value of variables (see, e.g., line 10 where we set variable `sold` to 1). Variables are declared in the `variables` block, see Figure 6, and implicitly might have an infinite domain. Notably, in [1] we show that the fact that variables can take infinite values easily leads to a model with infinite state spaces. As discussed in [1], this limits the application of exact analysis techniques, and required to consider SMC. As we can see from the `init` block, the first state is the `factory`, and the machine is initialized with only the `Coffee`. According to the transitions of state `factory`, e.g., `Coffee` can be replaced with `Tea`, and the other two beverages can be installed. When the dispenser is sold (line 10), it moves to state `deposit`. Notably, every time a `deploy` action is performed, the variable `deploys` is increased by one, leading potentially to an infinite state space. The end user can customize the dispenser by installing or uninstalling one of the beverages. When ready, the dispenser is deployed (line 16) moving into `operating` state where the installed beverage can be served. Finally, the dispenser can be sent back to the `deposit`.

This procedural specification can be graphically depicted as in Figure 7, automatically generated by QFLan, edited by hand to improve readability, e.g., some *loop* edges have been removed. For instance, the graph is missing `replace(Coffee,Tea)` and `install/uninstall(Cocoa)` in `factory` and in `deposit`, despite these being present in the actual model.

```
begin processes diagram
begin process dynamics
    states = factory , deposit , operating , prepareCoffee ,
        prepareCappuccino, prepareTea , prepareChocaccino
    transitions =
    // Factory state
    factory -(replace(Coffee,Tea),20)->factory,
    factory -(install(Cocoa),10)->factory,
    factory -(install(Cappuccino),10)->factory,
    factory -(uninstall(Cappuccino),10)->factory,
    factory -(sell,1,{sold=1})-> deposit,
    //Deposit state
    deposit -(install(Cappuccino),2.0)->deposit,
    deposit -(uninstall(Cappuccino),2.0)->deposit,
    deposit -(install(Cocoa),2.0)->deposit,
    deposit -(uninstall(Cocoa),2.0)->deposit,
    deposit -(deploy,2,{deploys=deploys+1})-> operating ,
    // Operating state
    // Serving Coffee
    operating -(Coffee,3)-> prepareCoffee,
    prepareCoffee -(serveCoffee,1) -> operating,
    // Serving Cappuccino
    operating -(Cappuccino,3)-> prepareCappuccino,
    prepareCappuccino -(serveCappuccino,1) -> operating,
    // Serving Chocaccino
    operating -(chocaccino,2)-> prepareChocaccino,
    prepareChocaccino -(serveChocaccino,1) -> operating,
    // Serving Tea
    // ...
    operating -(reconfigure,1) -> deposit
end process
end processes diagram

begin init
    installedFeatures = { Coffee }
    initialProcesses = dynamics
end init
```

Figure 5: Probabilistic process of the model in QFLan

```
begin actions
    sell deploy reconfigure
    serveCoffee serveCappuccino serveChocaccino serveTea chocaccino
end actions

begin variables
    sold = 0    deploys = 0
end variables
```

Figure 6: Actions and variables

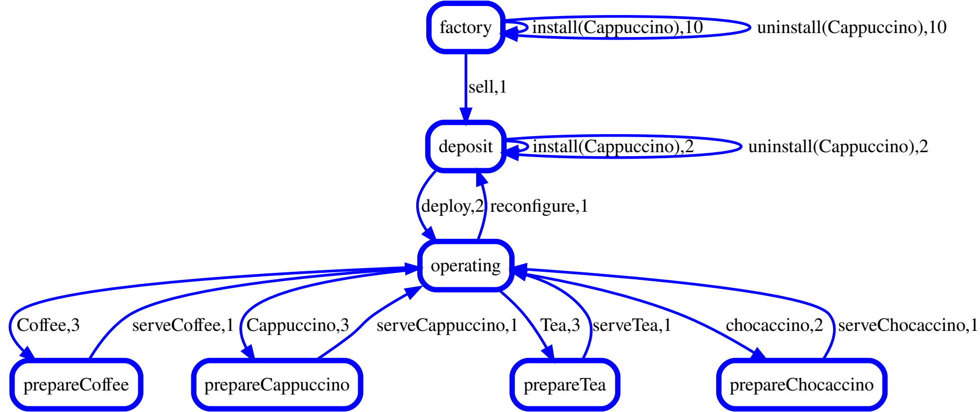

Figure 7: Graphical representation of procedural part of the QFLan model of the vending machine. Automatically generated in dot format by QFLan (edited by hand to improve readability).

2.2. Black-box analysis of simulation models with Statistical Model Checking

QFLan models can be analyzed by black-box SMC [2] using the tool MultiVeStA which can be plugged into existing simulators [21]. Given a quantitative property of interest, e.g., the probability of installing a feature or the average price of sold vending machines, MultiVeStA performs *enough* probabilistic simulations of the model to obtain statistically reliable estimations of the property. Black-box SMC is a simulation-based approach where only probabilistic simulations of the model are performed, with no assumption about the overarching behavior of the model. MultiVeStA is an example of a black-box SMC tool that can perform statistical analyses over multiple properties simultaneously and is highly scalable [21, 22]. The tool enables the user to query for one or more properties of the model they want to estimate and returns the estimation of those properties within confidence intervals. For instance, if X is a random variable giving the price of sold bikes in a simulation, then MultiVeStA will estimate its expected value $E[X]$ as the mean $\overline{x}$ of n independent simulations, with n large enough but minimal, to build a $(1-\alpha)*100\%$ of width at most δ centered on $\overline{x}$. In other words, MultiVeStA guarantees that $E[X]$ belongs to the interval $[\overline{x}-\delta/2, \overline{x}+\delta/2]$ with statistical confidence of $(1-\alpha)\cdot 100\%$. The generation of new samples ends when the confidence interval size is less than or equal to δ (α and δ are user-specified parameters). MultiVeStA has been successfully applied to various domains, including security risk modeling [1], economic agent-based

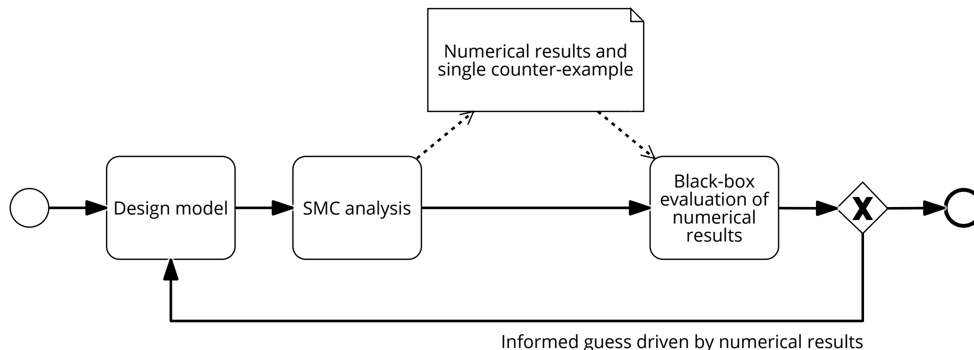

Figure 8: Sketch of SMC-based black-box validation

models [21], highly-configurable systems [13, 4], public transportation systems [23, 24], lending pools in decentralized finance [25], business process modeling [26], robotic scenarios with planning capabilities [27], and crowd steering scenarios [22]. Classic qualitative model checking, where properties are either satisfied or not, is able to provide counterexamples whenever a checked property does not hold. This is an example of system execution that falsifies the formula. Unfortunately, counterexamples are not common in quantitative variants of model checking, and even less in SMC. A common downside to most SMC approaches is that it does not provide behavioral explanations about why a property is estimated to a given value. This is what we want to solve with our methodology.

We remark that QFLan allows to perform analyses on the overall family, and not of single products. For example, we can study the probability of having a given feature installed in products of a family, the average cost of products of a family, etc. QFlan does not directly allow analyses of properties specific to an individual product within that family. However, studying properties of a single product could be accomplished by constraining the probabilistic process to focus solely on that product rather than the entire product range within the family. Alternatively, defining the set of installed features in the `init` block of the QFLan model in a manner that permits the analysis of a single product is another approach.

State-of-the-art SMC-based validation process. Figure 8 illustrates the state-of-the-art process adopted in a traditional SMC setting. The process begins with the modeler, who creates the model and then instructs the SMC to

estimate properties of interest for the system being modeled. SMC returns estimations of the properties without providing any additional information on why the results were obtained. SMC might provide single counter-examples, e.g., an interesting simulation, but to the best of our knowledge, there are no SMC approaches that try to combine simulations to obtain a representation of the dynamics that led to and explain a given estimation. In case the estimates are inconsistent with the expectations of the modeler, the modeler must make an informed guess on how to modify and correct the model. We call this process *SMC-guided black-box validation*. This is because any decisions to alter the model are made in a black-box manner without knowing the reasons behind the results of SMC. From this discussion, it emerges the need for a methodology like ours that aims at identifying unwanted behaviors and highlighting their origin.

2.3. Synthesis of models from their executions using Process Mining

Process Mining (PM) is an interdisciplinary field that seeks to extract insights from the actual executions of a process by bridging the gap between data science and process science [11]. The main activities of PM include discovery, enhancement, and conformance checking. Discovery involves identifying an abstract representation of the executed process by combining all the observed instances into a single model. Enhancement enriches the model with additional information, such as the frequency of executed activities or paths. Finally, conformance checking assesses the extent to which a normative model deviates from actual executions.

In this work, we are interested in the discovery and enhancement tasks of PM. Specifically, we aim to use execution traces (simulations) obtained from SMC analyses to synthesize new models that capture the behavior of the model as observed in the simulations, even for models with infinite states. To accomplish this goal we employ the Heuristics Miner (HM) algorithm [28, 11]. The HM algorithm can provide an accurate and comprehensive understanding of complex process dependencies, facilitating the alignment (the comparison) of the generated and the expected behavior of a model. In addition, the HM algorithm allows the user to adjust some parameters to control the trade-off between model fitness and the inclusion of infrequent paths, such as the noise threshold and dependency threshold parameters. Adjusting these parameters increases the likelihood of including infrequent paths in the discovered process model. In this regard, we follow a conservative approach that preserves any behavior observed during the simulations.

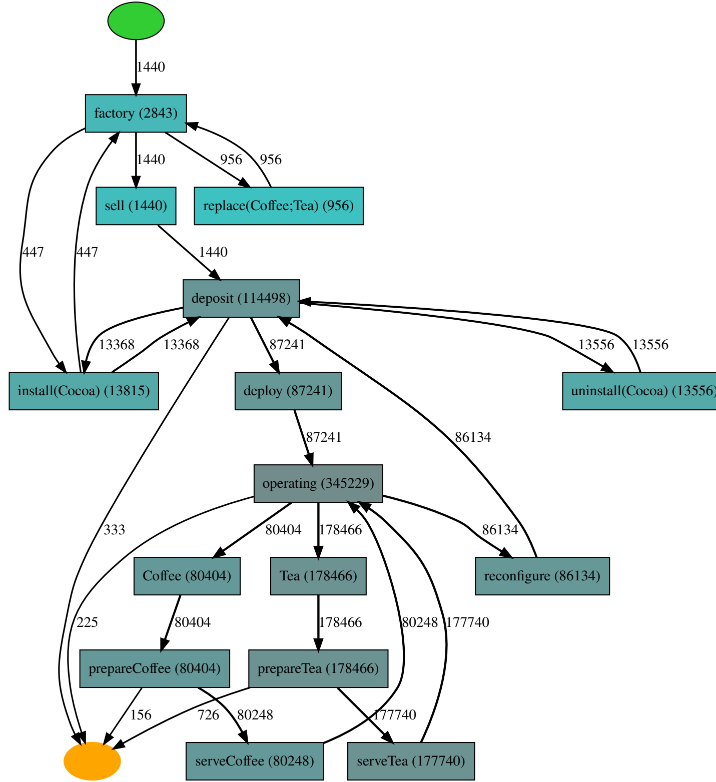

Figure 9: Heuristic Net simulation Vending machine model

Figure 9 depicts a Heuristic Net (HN) obtained by applying HM on the simulations of our running example (for the constraint of the maximum price of sold machines set to 10). As we can see, an HN consists of nodes connected by edges labeled with frequencies. In HN jargon, states are known as *activities*. In our methodology, QFLan activities and states get flattened in the same notion of activity in an HN. As we can see, the HN has two additional states, the green and red circles, that represent the *start* and *end* state, respectively, of the mined process model.

3. Method

We propose a method that enriches SMC techniques with PM techniques to overcome the limitations of the classic SMC black-box validation seen in Section 2.2. In our previous study [12], we have demonstrated the effectiveness of combining SMC analyses with PM techniques in identifying undesired

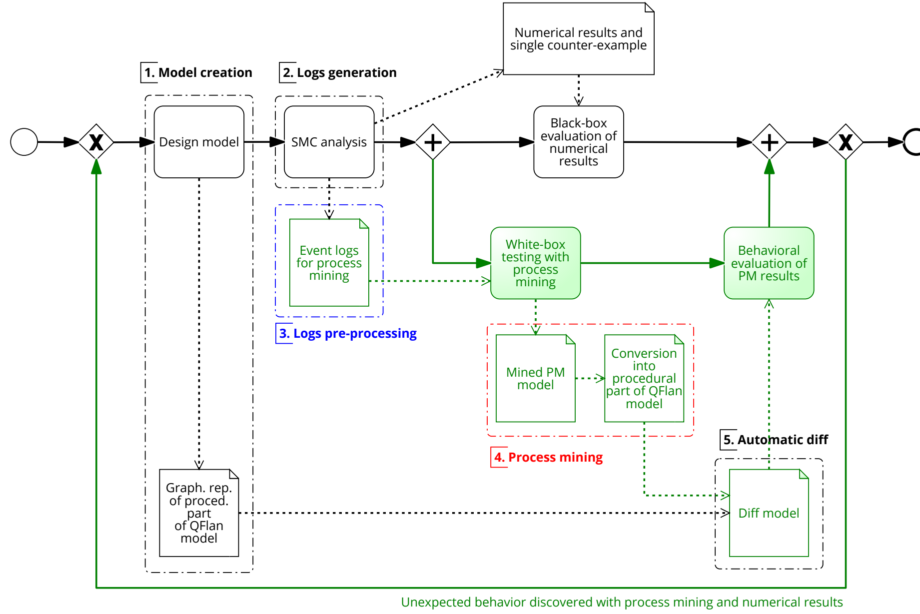

Figure 10: The new methodology presented in this paper, combining SMC and PM for *automatic* white-box behavioral validation. Green activities/data-objects/sequences identify novel aspects introduced in this paper. The main steps of the procedure are numbered.

behaviors in formal models. The integration of PM allows us to incorporate a white-box analysis of the model's behavior by leveraging mining techniques on simulated model executions. However, our earlier work was preliminary and not automated: it relied primarily on the modeler to visually identify undesirable behaviors as depicted in the PM output. Furthermore, the PM output was given in a formalism different from the one model specification language, making our approach less accessible. To solve these issues, in this current study, we go beyond a simple discovery algorithm applied to event logs. Instead, we establish an integration between SMC and PM techniques through a graphical component given in the model specification language. This component takes SMC logs as input, applies PM techniques to analyze them, and then visualizes the results using the original model specification language. This new approach facilitates the automatic discovery of missing or undesirable behaviors in the model.

Figure 10 illustrates the proposed methodology. It enhances SMC with automatic white-box behavioral model validation. The presented methodology incorporates several enhancements compared to our previous work. To

begin with, prior to applying the discovery algorithm to SMC logs, we now conduct a pre-processing step, which is described in Section 3.3. We then mine a PM model and convert it into a graph that represents the procedural aspect of the QFLan model. Subsequently, we compare this mined model with the original graphical representation of the procedural part of the QFLan model. The comparison yields a *diff* model, which highlights any disparities in behavior between the formal model designed by the modeler and the actually simulated model. In our previous work, we demonstrated the potential of our approach solely by applying a discovery algorithm to unprocessed SMC logs in order to extract a PM model. An example of the outcome of our previous work is the PM model depicted in Figure 9, which was obtained by mining the SMC logs of our running example with the maximum price of sold machine set to 10 using the HN algorithm. In this process model, it is not possible to visually evaluate whether those transitions represent the entire behavior of the model or merely a subgroup of the transitions. However, as discussed in Section 4.3, using our current methodology, we can thoroughly evaluate whether the model behaves correctly by identifying transitions that are absent in the simulated model when altering the quantitative constraints within the model.

More precisely, our methodology consists of five steps, numbered in Figure 10. In step *1. Model creation*, the modeler creates a model and the graphical representation of its procedural part using a model specification language. For example, QFLan, where the graphical representation is generated automatically from the model description. Then, in *2. Logs generation* we use an SMC tool to run simulations of the model to study a given property. Information on each simulation is stored as a log of events (an *event log*), containing, e.g., time stamps, actions executed, etc. In this paper, we consider the SMC tool MultiVeStA which has been extended with log generation capabilities. Once the simulated event logs are obtained, we pre-process them in step *3. Logs pre-processing*. In step *4. Process mining*, we apply PM techniques on these logs to discover the process model describing the behavior of the model as observed in the simulation. In the figure, we call this the *mined PM model*. We then post-process the PM result to convert it into (the procedural part of) a QFLan model. Finally, in step *5. Automatic diff*, we compare the graphical representation of the original model with the one discovered in our step 4. The result is a graphical representation, in terms of the source modeling language, highlighting the differences between the expected behavior of the model with the real one. We call this the *diff*

model. As we will see in our experimental section, the diff model can explain the results obtained by SMC and suggest fixes if necessary. All steps of our methodology (apart from the initial model design) are fully automated.

We refer to this methodology as *white-box behavioral model validation* because, thanks to the union of SMC and PM, we can access the internal workings of the system by shedding light on its actual behavior. Therefore, the modeler can now rely on more than just an informed guess to fix the model, the diff model. In the remaining parts of this section, we describe how we implement each of these steps.

3.1. Model creation

The first step of our methodology starts with creating the (QFLan) model where the modeler defines all the components of the system, e.g., features, variables, a list of constraints, and its procedural part. QFLan will then automatically generate a *graphical representation of the procedural part of the QFLan model*. The one for our Vending machines running example is sketched in Figure 7. This is what in Figure 1 we call the “input model”.

3.2. Logs generation

In this step, the modeler chooses the properties of interest for the model and evaluates them using MultiVeStA [21, 29]. For example, we might be interested in the average price of the sold vending machines. MultiVeStA will instruct the simulator to run the required simulations, saving information on them as event logs.

MultiVeStA has a clear interface to plug into new simulators only involving three functionalities: *reset to perform a new simulation*, *perform one step of simulation*, *evaluate an observation in the current simulation state* [21, 29]. To enable log generation, we added two new functionalities to the interface of MultiVeStA: *create an empty log file*, invoked once per SMC analysis, and *add row to log*, invoked whenever an event (of interest) is to be recorded. These functionalities have to be implemented whenever integrating MultiVeStA to a new simulator. When implementing the latter functionality, the modeler might decide to record all events, i.e., all simulation states, or only selected events of interest. In QFLan, we add a row whenever we perform one step of a simulation. The recorded information includes the incremental counter of steps (i.e., the time stamp), the unique random seed used by the current simulation (used as case ID, the unique identifier of the simulation/case),

the executed action (i.e., the activity), the target state of the executed transition, and any relevant additional information (features currently installed, and values of variables). All information is stored in separate columns and saved in a CSV file.

3.3. Logs Pre-processing

In this step, we pre-process the event logs stored before applying PM techniques. The pre-processing consists of merging the columns that record the target states and the actions used to move from one state to another. This means that states and actions will be treated as activities when we apply the PM discovery algorithm. In Section 2.3, we showed an example of the HN mined from event logs generated from our Vending machines running example, and pre-processed as discussed here. Connected to the merging of the two columns, in order to preserve the correct order to avoid losing information about the transition that executed a given action, we change the name of the actions by adding the names of the origin and target states. Renaming an action is essential because the same action can appear in different transitions across different states. Without such renaming, we would loose information on the actual executed process. For instance, in the "input model" of Figure 1, when choosing `ActionC` to move from `B` to `C`, we change the name of the action from `ActionC` to `ActionC_B_C`. Instead, in the case of execution of the action to move from `E` to `C`, we would get `ActionC_E_C`.

3.4. Process mining

We now mine the pre-processed event logs using the Heuristic Miner (HM) algorithm [28, 11] discussed in Section 2.3. We use the library PM4PY[1] [30], a versatile Python library that can help in using different PM algorithms. Once the Mined PM model is discovered, we then parse it to extract edges and nodes and use them to convert it from a *PM model* (i.e., a Heuristic net mentioned in Section 2.3) into a *mined QFLan model* (actually, the procedural part of a QFLan model). In Figure 1, this corresponds to the left graph of the "output model". In this process, we revert the names of the actions to their original ones. This helps in comparing the original QFLan model with the mined one.

[1] We use the parameters, i.e., `dependency_threshold=0.5`, `and_threshold=0.65`, `loop_two_threshold=0.5` and `dfg_pre_cleaning_noise_thresh = 0`. See https://pm4py.fit.fraunhofer.de/documentation

3.5. Automatic diff

The last step of our methodology starts by parsing the graphical representations of (the procedural part of) the original QFLan model from step 1 and of the mined one from step 4. This allows us to compare the two models, and create a *diff model* that highlights the existing differences. The diff model is built as follows: it includes the edges and nodes present in both models, without highlighting them (i.e., it uses the same color and style as in QFLan). Then, we add all edges and nodes that appear in only one of the models, this time highlighting them in red. The rightmost graph of the "output model" of Figure 1 depicts the diff model for the other two graphs in the figure. We can see that the red dashed edges (e.g., from node `C` to node `D`) denote edges present in the original model, but missing in the mined one. Vice versa, red continuous edges (e.g., from node `C` to node `F`) denote edges not present in the original model, but present in the mined one.

Therefore, dashed red edges denote transitions that the simulator has never taken, implying that the formal model includes some constraints that might *always prevent* those transitions. We remark that this information might also help the modeler when testing the effect of new constraints, by modifying the model and observing the result of our methodology. As demonstrated in the experiments in Section 4, in some cases, the modeler could be interested in intentionally varying some constraints, such as quantitative constraints, to understand their effective impact on the behavior of the model. Indeed, by using a classic SMC black-box validation approach, it might not be possible to detect the impact of a constraint on the obtained numerical values that only summarize the estimations of some properties of interest. In Appendix A, we provide an example of an analysis conducted on the Elevator model, which is discussed in Section 5. This instance emphasizes how a minor typo in the probabilistic process of the model can cause unexpected behavior which impacts the model evaluation via SMC, and how our methodology can spot this issue.

Instead, continuous red edges, such as the edge between nodes `C` and `F`, represent transitions present only in the mined model. In QFLan, this can only happen in case the simulator gets stuck in a *deadlock state*, i.e., it is in a state where there would be transitions to execute, but they are all disabled by the constraints. In other domains, instead, we might in principle have further classes of continuous red edges. Alternatively, there might be cases in which one implements the *add row to log* functionality of MultiVeStA such that it adds extra rows under predetermined conditions. The presence of continuous

red edges in the diff model might signal errors that could compromise the validity of the results obtained by SMC. This is exemplified in detail in Section 6, where we consider the security domain. There, we demonstrate that a property has a low probability (probability of an attacker succeeding in a robbery) just because of bugs in the model. This bug is identified and fixed thanks to our methodology. This gives new opportunities to improve the model that were not possible with the classic SMC black-box validation method.

In the following sections, we apply the presented methodology to answer our research questions. We consider the running example (effectiveness), a parametric SPL model (scalability), and a security model (multi-domain).

4. Experimental evaluation: RQ1 Effectiveness

To answer RQ1, we consider our running example from Section 2.1: an SPL model of vending machines from [4]. The goal of this section is to illustrate the effectiveness of our methodology.

4.1. Domain description - Coffee Vending Machine

Besides the hierarchical, cross-tree, and action constraints, QFLan allows for another essential class of constraints: quantitative constraints exemplified in Figure 4 for the considered case study. In this case, the quantitative constraints specify the maximum `price` that a vending machine might have. In particular, Figure 4 imposes that no machine will be produced that costs more than 10 or 15 euros (depending on which of the two constraints is used). Changing such constraints changes the behavior of the model. This is because some beverages cannot be installed in the dispenser if this is too low.

For instance, in [4] the authors focus on *sold machines*, i.e., those obtained after executing action `sell` from state `factory` in Figure 7. For such machines, the authors show that if the maximum price is 10, then the probability of a sold machine having a `Cappuccino` dispenser is zero. This is because the hierarchical constraints in Figure 2 impose that, in order to have `Cappuccino`, a machine must serve `Coffee`. The latter costs 5, while `Cappuccino` costs 7 (Figure 2), for a total cost of 12 that would falsify the constraint on price. It is also shown that the probability of installing a `Cappuccino` increases if the constraint on the price is relaxed to a maximum of 15.

```
//Query 1
begin analysis
    query = when sold == 1 :
    {price(Machine) [delta=0.5], Coffee, Tea, Cappuccino, Cocoa}
    default delta=0.05    alpha = 0.05    parallelism = 1
    logs ="log_name_sold.csv"
end analysis

//Query 2
begin analysis
    query = eval from 1 to 500 by 1 :
    {price(Machine) [delta=0.5], Coffee, Tea, Cappuccino, Cocoa}
    default delta=0.05    alpha = 0.05    parallelism = 1
    logs ="log_name_steps.csv"
end analysis
```

Figure 11: Two MultiVeStA queries to analyse the Vending machine model

Our experiments follow a strategy similar to that of [4], but considering a more challenging setting: we add an `uninstall(Cappuccino)` transition in the `factory` state. This has the effect that a probability 0 of having `Cappuccino` after `sell` action (Line 10 of Figure 5) does not guarantee that `Cappuccino` has not been installed at all, because it might have been uninstalled. Therefore, the black box analysis in [4] would not guarantee that the constraint has never been violated before selling the machine. Instead, we show here that our white-box approach can guarantee this.

4.2. Experiments

We run two experiments for two configurations obtained by setting to 10 and 15, respectively, the maximum accepted price. We study the probability of having `Cappuccino` installed right after the `sell` action. We also study the average price of sold machines, as well as the probability of having `Tea`, `Coffee`, and `Cocoa`. To run the experiments, we invoke MultiVeStA using the QFLan GUI. We consider the two queries in Figure 11. Query 1 instructs MultiVeStA to evaluate the properties on the first simulation state in which the variable `sold` has value 1. This variable is set when `factory` performs action `sell` (see Line 10 in Figure 5). We also set the two parameters specifying the required confidence interval, α, and δ, to 0.05 (for the price we use $\delta = 0.5$). Therefore, we ask MultiVeStA to compute 95% confidence intervals of width at most 0.05 (0.5 for price).

As a second set of experiments, we study the same properties, but at the varying of the simulation steps, from 1 to 500. This is obtained using Query

	Studied properties				
Maximum Price	**Avg Price**	**Tea**	**Coffee**	**Cocoa**	**Cappuccino**
10	5.53	0.64	0.33	0.19	0.00
15	7.45	0.54	0.46	0.42	0.22

Table 1: Numerical results experiments Vending machine experiments

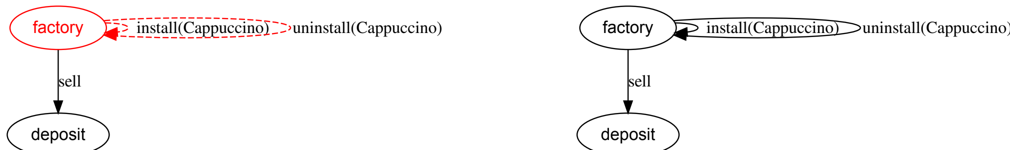

Figure 12: Diff models for Query 1 of Figure 11. (Left) Model for maximum price 10. (Right) Model for maximum price 15.

2 in Figure 11. We do this because this analysis regards a larger portion of the dynamics of the model, allowing us to further exemplify the advantages brought by our methodology.

4.3. Results

Considering Query 1, MultiVeStA instructed the QFLan probabilistic simulator to perform 1440 and 1600 simulations to estimate all properties for the case of maximum price 10 and 15, respectively.

Table 1 lists the numerical results obtained by MultiVeStA for the two considered maximum prices. As shown in the first row in Table 1, the results confirm that with maximum price 10 the probability of having `Cappuccino` installed in sold machines is zero. Instead, for the case of a maximum price of 15, the probability increases to 0.22. This is in line with the results in [4]. This is an example of potentially unexpected behavior resulting from the richness of QFLan's constraints: the model either allows or prohibits the feature `Cappuccino` depending on the strictness of a constraint. Thanks to the methodology proposed in this paper, in addition to the numerical results, we can show graphically the behavior of the Vending machine model and how a different constraint on price can change the behavior of the model.

Figure 12 depicts the diff models obtained by comparing the original QFLan model, and the ones mined using the simulation logs. Similarly to Figure 7, to improve readability we have edited the images to drop some edges irrelevant to this paper. We did not drop any red edges. Figure 12 (Left) refers to the case of maximum price 10. Here, we can see that the

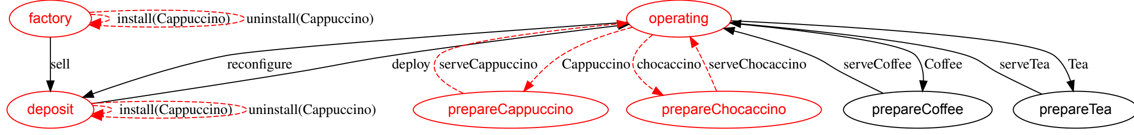

Figure 13: Diff model for Query 2 of Figure 11 for maximum price 10.

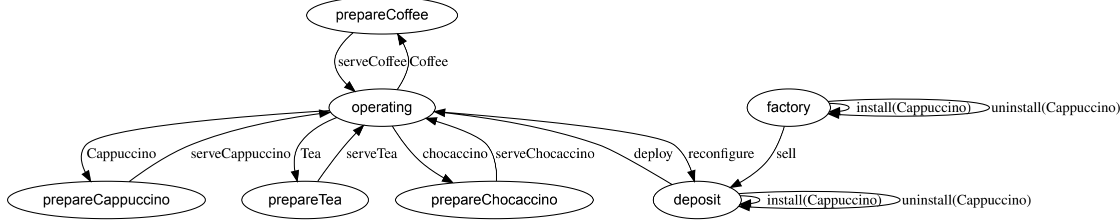

Figure 14: Diff model for Query 2 of Figure 11 for maximum price 15.

edges for `install` and `uninstall` of `Cappuccino` are marked in red (as well as node `factory`). This means that, even if those transitions are included in the model, they do not appear in its behavior (see Section 3). Figure 12 (Left) confirms our aforementioned hypothesis that the simulator never completes this transition because of the quantitative constraints.

Figure 12 (Right) considers the case in which the maximum value for the price is 15. We can see that all the edges are now black. This means that the behavior of the model now also allows to install and uninstall `Cappuccino`. We can, therefore, effectively execute all transitions present in the part of the model specification relevant to this query (i.e., the transitions from state `factory`).

We now move our attention to Query 2 from Figure 11. This time we do not focus only on transitions executed in state `factory`, but on all transitions executed in the first 500 simulation steps. We chose 500 because, from preliminary investigations, these are enough to allow the model to express all parts of its behavior. As for Query 1, we consider the two cases of maximum prices (10 and 15). For both experiments, MultiVeStA instructed the QFLan simulator to run 1440 simulations.

Figures 13 and 14 depict the diff models obtained by comparing the original model behavior, and the ones mined on the simulation logs. Figure 13 considers the case of maximum price 10, while Figure 14 15. Images were edited similarly to Figure 12 to improve readability. We can see that the case of maximum price 10 presents several red edges (and nodes). Node `factory`

presents the missing transitions related to `Cappuccino` discussed for Query 1. Furthermore, all transitions related to `Cappuccino` in other nodes are missing as well. E.g., we never enter in state `prepareCappuccino`, as we need to execute action `Cappuccino` to get there, but this action is enabled only if the corresponding feature is installed. Likewise, we cannot serve `Chocaccino`, because it requires to have both `Cappuccino` and `Cocoa` (see Figure 3).

Conversely, Figure 14 does not contain any red edge. Therefore, the more permissive constraint on maximum price does not prevent any part of the behavior of the machine.

Discussion. These experiments demonstrate the effectiveness of our method, which, thanks to the integration of SMC and PM techniques, enables us to thoroughly evaluate visually unexpected deviations between the behavior intended by the modeler, and the actual one obtained simulation the model. Such discrepancies might not be apparent in the model, as they might occur due to the richness of constraints present in QFLan. The application of PM techniques aids in exploring and uncovering these unexpected behaviors. Therefore, we can positively answer to RQ1.

5. Experimental evaluation: RQ2 Scalability

To answer RQ2, we consider a classic SPL example with parametric complexity: the elevator product line introduced in [17]. This has become a widely-used benchmark in PLE, especially as regards scalability studies for novel methodologies (see, for instance, [31, 32, 33, 34, 35, 13, 36, 37, 38, 39]). This case study is considered particularly challenging by the community. It has 9 independent and unconstrained features, yielding 512 products. More importantly, it allows to consider instances with an increasingly large number of floors. In this section we use it with the same goal: to illustrate the scalability of our methodology.

5.1. Domain description - Elevator

The elevator SPL has been provided in several *incarnations*. Here we consider the one from [13]. Product line engineers often encounter challenges when designing and developing products with numerous independent and unconstrained features. The Elevator SPL fits well for this scope. Furthermore, this case study is especially useful for product line engineers who need to consider instances of increasingly larger complexity, as it is possible to

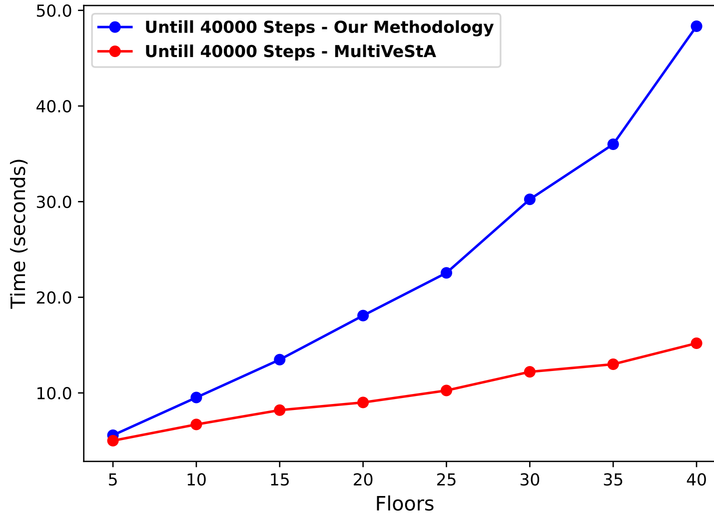

Figure 15: Runtime analysis: Runtime of our methodology (blue line). Runtime of MultiVeStA analysis (red line). Times are averaged over 10 replications of the analysis of the same MultiVeStA query over increasingly complex versions of the elevator SPL with 5, 10, . . . , 40 floors.

parameterize the model for different numbers of floors. The Elevator SPL consists of multiple platform and cabin buttons, one per floor, used to summon/direct the elevator. Once a button is pressed, it remains active until the elevator has served the corresponding floor by opening and closing its doors. The specific case study we consider focuses on nine key features that can alter the behavior of the elevator. For example, the feature `AntiPrank` makes it necessary for a button to be kept pushed, while `Park` makes the elevator return to the first floor when empty (see [13] for the full list of considered features).

In [13], the authors use this benchmark to study the scalability of QFLan at the varying of the number of floors from 5 to 40, while usual SPL approaches can scale up to 10 floors. In particular, the authors of [13] consider the Elevator SPL with all unconstrained features from [17], with a fixed maximum capacity of the elevator set to eight persons, and a maximum allowed load of four persons.

5.2. Experiments

We follow an approach similar to the one in [13]. We consider a MultiVeStA property that checks that when the `load` variable (representing the number of people in the elevator) exceeds the `capacity` variable (representing the maximum capacity of the elevator), the elevator does not move (as indicated by variable `direction` having value `0.0`). To evaluate this property, MultiVeStA checks it for all states encountered within the first `maxStep` steps. As soon as the condition is not satisfied, the current simulation terminates, otherwise, `maxStep` steps are performed. In [13], the authors considered varying numbers of `maxStep`, from $5,000$ to $40,000$. Here we consider only the latter largest case. This property is always satisfied, i.e., we always get value 1. This was on purpose, to guarantee that every simulation will always consist of $40,000$ steps. As in [13], we consider a varying number of floors from 5 to 40.

5.3. Results

The results are shown in Figure 15, providing in red the runtime of the MultiVeStA analysis, and in blue that of our methodology. The latter includes pre-processing of simulation logs, process mining, and generation of diff models. We can see that our methodology succeeded in all instances and that it can provide results in less than a minute even for the largest instances. The methodology tends to have a runtime in the same order of magnitude of the MultiVeStA analysis, and it never exceeds more than 5 times its runtime.

Discussion. These experiments demonstrate the scalability of our method. In fact, it could be successfully applied to SPL models considered particularly challenging by the PLE community, and that are regularly used as benchmarks. Therefore, we can positively answer RQ2: our techniques can indeed be applied to large SPL models considered challenging by the PLE community.

6. Experimental evaluation: RQ3 Multi-domain

So far we have shown how our methodology can be applied to well-known SPL models from the PLE community. The goal of this section is to show that our approach can easily generalize to models from other domains. In particular, we consider the cyber-security domain, using *attack-defense tress* (ADT, or just attack trees). Their use is recommended by NATO [40], and

are widely used, e.g., in aerospace [41], or safety-critical cyber-physical systems [42].

6.1. Domain description - Cyber-security Attacks and Threat Modeling

We consider the threat model presented in [1], describing the attack strategies that a thief can attempt when trying to complete a robbery in a bank. Notably, the authors of [1] emphasized that this model has infinite state space, preventing the use of exact analysis techniques based on an exhaustive exploration of such state space. The authors of [1] leveraged this aspect to advocate the utilization of SMC for analyzing this model. In the preliminary workshop version of this paper [12], we have exemplified how an embryonic version of our methodology could be applied to an extreme simplification of this model. The model has been created in RisQFLan [1], a member of the QFLan family recast to target the security domain. Similarly to QFLan, RisQFLan models consist of a declarative part, the *attack-defense tree* (ADT), and a procedural part, the *probabilistic attacker*. The intuition is that while a feature diagram describes a family of products, an ADT describes the family of possible attacks on a system. Similarly, while a probabilistic process of QFLan enables for dynamic reconfigurable SPLs, a probabilistic attacker in RisQFLan allows studying how vulnerable a system is to specific attackers (e.g., bound to stringent or permissive budget constraints). We refer to [1] for a deep presentation of RisQFLan, and of how the feature-oriented framework QFLan has been recast to this new domain.

Figure 16 shows the ADT considered in this section. The root represents the root attack goal of robbing the bank. Nodes `OpenVault` and `BlowUp` are two sub-attacks, grouped by an *OR*-relation. `OpenVault` further refines in `Learncombo` and `GetToVault`, grouped by an *AND*-relation: in order to open the vault, we need first to learn the combo and get to the vault. `LearnCombo` is further refined in three nodes, `FindCodei`, grouped by a *2-out-of-3*-relation. An attack node can succeed only if its refinements allow for it. E.g., `LearnCombo` can succeed only if at least two `FindCodei` attacks succeeded. An attack node in RisQFLan can be somehow mapped to a feature in QFLan, but these are different notions with different interpretations and therefore are handled differently in the formal semantics of the two languages. An ADT also has other types of nodes, *defense* nodes, like the defense `Memo` and the *countermeasure* `LockDown`. The former is a static defense that decreases the probability of success of the attacks to which it is connected (`FindCode2`). Instead, a countermeasure is a dynamic defense that must be

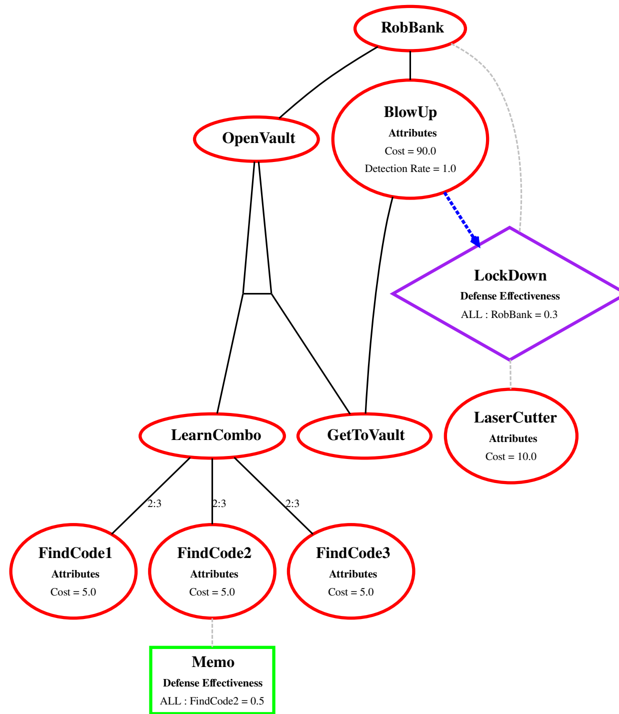

Figure 16: ADT RobBank model

activated by attack attempts that it can monitor (`BlowUp`), denoted by the blue arrow. Once activated, countermeasures behave like defenses.

Intuitively, the ADT can be read as follow: the thief can attempt the robbery by two strategies: opening the vault or blowing it up. Both strategies require that the thief gets to the vault. In addition, opening the vault requires the thief to learn the combination of the vault, which in turn requires discovering at least two codes.

A RisQFLan model is given in textual format, similar to QFLan. The complete model can be found in [1]. Here we provide only the parts relevant to the performed experiments.

A model in RisQFLan, similarly to QFLan, can be equipped with several (quantitative) predicates and constraints. For example, Figure 16 shows that attack nodes have a `Cost`, paid by the attacker every time the corresponding sub-attack is attempted. At the same time, The block `quantitative constraints` in Figure 17 depicts how we can constrain the dynamics of an attacker to finite resources, e.g., by imposing attackers to not spend more

than 100 (EUR). In other words, the sum of the costs of attempted attacks cannot be higher than 100.

The considered probabilistic attacker is illustrated in Figure 17, given in a format similar to that of the probabilistic process of QFLan. Similarly to QFLan, the RisQFLan simulator will select the next transition to execute in a simulation depending on their weights and on several types of constraints.

In the model, we can also define the prior knowledge and availability of the attacker. The block `init` in Figure 17 imposes that the thief already knows the first combination, and owns a laser cutter to disable the lockdown defense.

Similarly to Figure 7, Figure 18 provides a graphical representation of the probabilistic attacker. This plays the role of the input model in Figure 1. From Figure 18 we can see that each attack begins in the `Start` node, where the simulator can choose to blow up or open the vault. After an attack attempt, the simulator returns to the Start state, where it chooses to try another attack or to complete the robbery if this is allowed. If the root attack succeeds, the attacker will move, and terminate, in state `Complete`. Attack attempts might also fail. This is dictated by the weights added in the probabilistic attacker and by the existing defenses.

6.2. Experiments on the original model

To demonstrate that our method can automatically discover unwanted behaviors also in this domain, we use MultiVeStA to analyze the query in Figure 19. This instructs MultiVeStA to evaluate the probability of success of eight attacks in each simulation step from 1 to 100. The CI specification is as in QFLan. As in [1], we assume that the attacker owns a `LaserCutter` which disables the `Lockdown` defense, and that s/he already succeeded in obtaining the first code of the vault (Figure 17). We will show that our methodology can pinpoint issues in the model, and how it can suggest fixes.

6.3. Results on the original model

MultiVeStA instructed the probabilistic simulator to run 320 simulations. Table 2 lists the analysis results obtained for step 100. We can notice that the probability of a total lockdown is equal to zero, as expected by the presence of the `LaserCutter`.

Another critical element highlighted by the SMC analysis is that the probability of succeeding in the root attack is 0.175 despite `LockDown` is disabled. The reasons why only a few simulations (about 18%) ended with a complete

```
1     begin quantitative constraints
2         { value(Cost) <= 100 }
3     end quantitative constraints
4     begin actions
5         tryAction tryGTV choose
6     end actions
7
8     begin attacker behaviour
9     begin attack
10        attacker = Thief
11        states = Start, TryOpenVault, TryLearnCombo, TryFindCode,
              TryGetToVault, TryBlowUp, Complete
12        transitions =
13        Start - (succ(RobBank), 2, allowed(RobBank)) -> Complete,
14        Start - (fail(RobBank), 1, allowed(RobBank)) -> Complete,
15        //Get to the vault attempt
16        Start -(tryGTV, 4, !has(GetToVault)) -> TryGetToVault,
17        TryGetToVault -(succ(GetToVault) , 2, {AttackAttempts =
              AttackAttempts + 1}) -> Start,
18        TryGetToVault -(fail(GetToVault), 1, {AttackAttempts =
              AttackAttempts + 1}) -> Start,
19        //Open the vault attempt
20        Start -(choose, 4) -> TryOpenVault,
21        TryOpenVault -(succ(OpenVault) , 2, {AttackAttempts =
              AttackAttempts + 1},has(LearnCombo) and has(GetToVault)) ->
              Start,
22        TryOpenVault -(fail(OpenVault), 1, {AttackAttempts = AttackAttempts
              + 1},has(LearnCombo) and has(GetToVault)) -> Start,
23        TryOpenVault -(tryAction , 2, has(LearnCombo) and !has(GetToVault))
              -> Start,
24        TryOpenVault -(tryAction, 5, !has(LearnCombo)) -> TryLearnCombo,
25        //Learn the combinations of he vault attempt
26        ...
27        //Blow up the vault attempt
28        Start -(choose, 4) -> TryBlowUp,
29        ...
30    end attack
31    end attacker behaviour
32
33    begin init
34        // LockDown cannot be activated if we have LaserCutter
35        Thief = {FindCode1, LaserCutter }
36    end init
```

Figure 17: Probabilistic attacker in RisQFLan

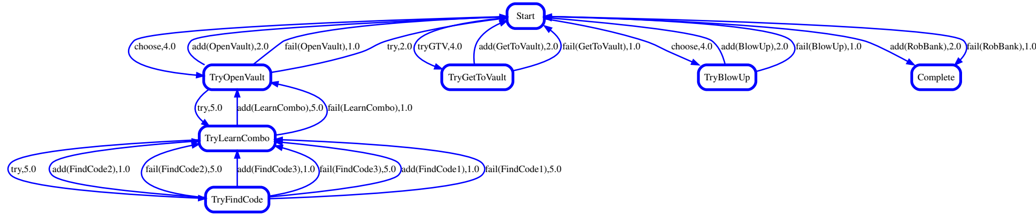

Figure 18: RisQFLan model: Attacker behavior

```
1  begin analysis
2     query = eval from 1 to 100 by 1 :
3     {RobBank, OpenVault, BlowUp,LearnCombo, GetToVault, FindCode2,
          FindCode3,LockDown}
4     default delta = 0.1   alpha = 0.1   parallelism = 1
5     logs = "log_RobBank.csv"
6  end analysis
```

Figure 19: Query to invoke MultiVeStA RobBank model

robbery of the bank are not easy to spot just by a black-box inspection of the numerical results.

Figure 20 depicts the diff model produced by our methodology. It displays two red edges and a red node; all the remaining edges and nodes are colored in black. In particular, the red node is a special node added by our methodology: a *deadlock* node. It denotes simulations that ended unexpectedly because no transitions were enabled. Thanks the two red edges, we can see that some of the simulations ended unexpectedly in the states `TryFindCode` and `TryBlowUp`, lowering the overall success probability.

Issue in the model, and fix. Figure 20 highlights issues in states `TryFindCode` and `TryBlowUp`. By looking at the original model specification, we can see that these states can only perform transitions to attempt attacks (transitions `fail` and `add` in Figure 18). The execution of these transitions increases a cost given by the cost of the attempted attacks (5 for `Findcode`, 90 for `BlowUp`). This makes us suspect that the deadlocks are due to the constraint on the maximum cost being set to only 100 (Figure 17). Therefore, if the attacker gets into these states without enough money to attempt the corresponding attack, s/he will get stuck there because no transition is enabled. To fix this unwanted behavior and to have a more reliable evaluation of the properties, the modeler needs to refine it by adding an extra *escape* transition in each of these two nodes to go back to their direct parent nodes. This transition shall

Property	RobBank	OpenVault	BlowUp	LearnCombo	GetToVault	FindCode2	FindCode3	LockDown
Probability	0.175	0.204	0.062	0.515	0.525	0.221	0.363	0.0

Table 2: Numerical results experiments on the original RobBank model

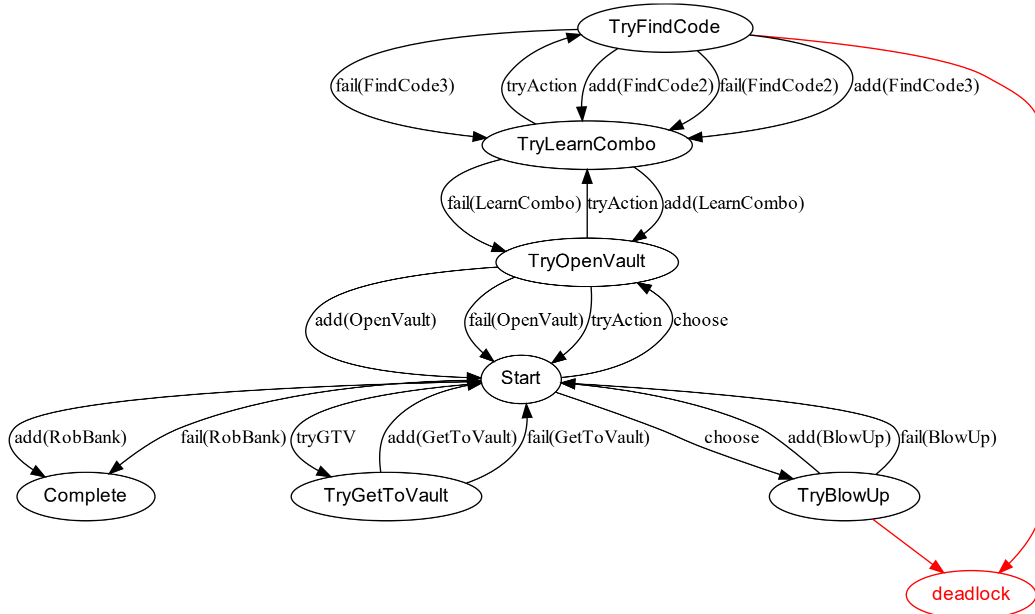

Figure 20: Diff model obtained experiments on the original RobBank model

have no cost, which is obtained by executing a custom action (see Figure 17).

6.4. Refined model

We refine our model by adding a new action `goBack` in Figure 17, and two transitions with this action from `TryBlowUp` to `Start`, and from `TryFindCode` to `TryLearnCombo`. Figure 21 depicts the refined attacker behavior with the new transitions highlighted in blue. We assign a low weight to these transitions, i.e., 0.1, to ensure that the simulator tends to choose them only when other options are not permitted.[2]

Experiments and results on the refined model. Thanks to these new transitions, the observed dynamics shall not exhibit anymore the discussed dead-

[2]The use of a low probability is a workaround. We could have used so-called *action constraints* [1], but this would have required an in depth description of RisQFLan.

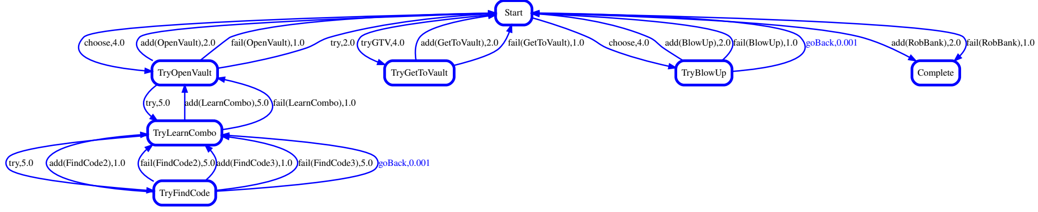

Figure 21: RisQFLan model: probabilistic attacker - Refined model

Property	RobBank	OpenVault	BlowUp	LearnCombo	GetToVault	FindCode2	FindCode3	LockDown
Probability	0.393	0.572	0.12	0.59	0.825	0.21	0.453	0.0

Table 3: Numerical results experiments on the refined RobBank model

locks. To ensure this, we use the same query from Figure 19. Also, in this case, MultiVeStA required to run 320 simulations. The results are given in Table 3. Besides `LockDown`, which is again equal to zero, all the other properties increased. The obtained diff model is given in Figure 22. No red edges or nodes are present, meaning that we fixed the issues.

Discussion. These experiments demonstrate that our methodology can be applied to domains beyond SPL, including cyber-security ones. Therefore, we can answer positively to RQ3, since we have automatically discovered unwanted and unexpected behaviors. In addition, we also got hints on how to fix such issues by obtaining a refined model that does not show the issues.

7. Related work

Since our approach is centered around the utilization of automated verification techniques, specifically probabilistic model checking combined with PM techniques, within the specific context of behavioral models of dynamic SPLs, we will give an overview of the related works that used models for specifying SPLs, providing the behavior of those along with the related verification techniques and tools. The well-known behavioral modeling languages for SPLs rely on overlaying multiple labeled transition systems (LTSs) that represent different variants of products onto a single, augmented LTS family model. However, only a few [33, 43] of these languages enable the specification of probabilistic SPL models, and even less support model-checking approaches. Featured Transition Systems (FTSs) were first introduced in [6] and later expanded upon in [44] and [45]. An FTS represents a family of

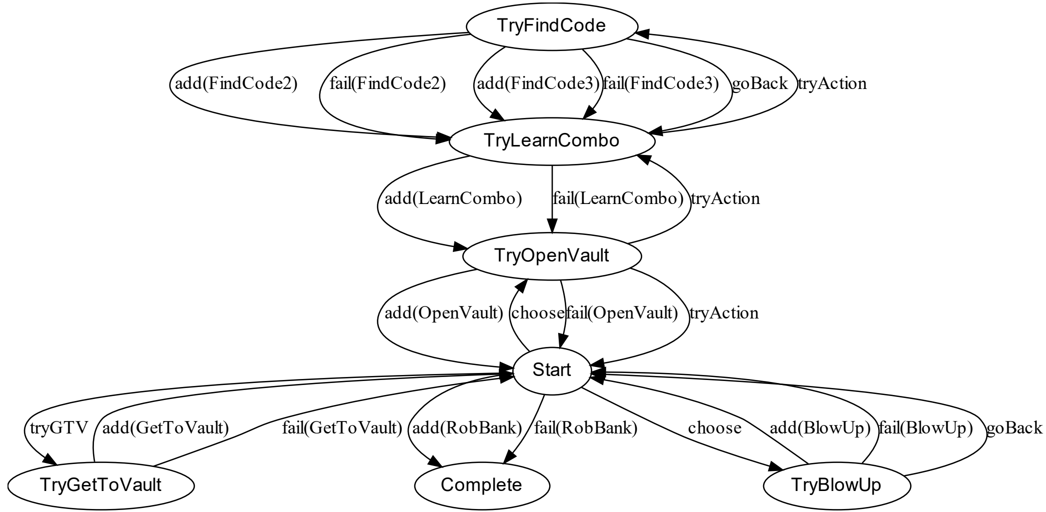

Figure 22: Diff model for the refined RobBank model

LTSs, with each LTS corresponding to a specific product. The LTSs are derived by projecting feature expressions (Boolean formulas defined over the feature set) assigned to the transitions. Transitions whose feature expressions are not satisfied by a particular product's feature set are eliminated, along with any unreachable states and transitions. In QFLan [13], the action constraints are similar to feature expressions in FTSs but they are applied to actions rather than transitions. While feature expressions offer more fine-grained specifications, action constraints in QFLan provide a more concise and declarative approach and support more general constraints and accommodate the modeling of adaptive or dynamic SPLs compared to FTS feature expressions. Modal Transition Systems (MTSs) [46, 47] represent a family of LTSs that, similar to FTSs, distinguishes between admissible transitions and necessary transitions. Nevertheless, in comparison with this family of LTSs, QFLan provides support for feature attributes and richer quantitative constraints and allows for the modeling of dynamic SPLs since the feature set is statically determined upfront. In [16], unreachable states and transitions, so-called hidden deadlocks, are made explicit through an algorithm that effectively transforms ambiguous FTSs into unambiguous ones and transforms them into a Modular Transition System that the modeler can more efficiently check. With QFLan, we are not interested in the static analysis of the formal model as in [16], but, instead, we run the model and conduct a sufficient

number of simulations to attain statistically significant outcomes regarding the particular query under investigation.

A sequence of works, summarized in [48], introduced the notion of Product Line CCS (PL-CCS). PL-CCS expands upon the CCS framework by incorporating a variant operator, which enables the representation of alternative behaviors as alternative processes. The objective is to ensure the existence of only one of these processes during runtime. Another notable approach, described in [49], is the choice calculus, which creates a common language for software variation management and aims to establish a foundational model for software variation, similar to the lambda calculus in programming languages. Additionally, DeltaCCS [50], an extension of CCS, draws inspiration from the widely used delta-modeling approach employed in automated product derivation for SPLs. The approach discussed in [51] utilizes deltas to specify incremental changes to a core product. In contrast to PL-CCS and the choice calculus, DeltaCCS follows a modular approach in which choices are applied at well-defined variation points. Model-checking algorithms have been implemented in MAUDE to verify SPLs specified in DeltaCCS against modal m-calculus formulas. Despite both PL-CCS and DeltaCCS offering fundamental mechanisms for restructuring or modifying SPLs, they are not able to effectively model dynamic SPLs. An alternative technique, known as Variant Process Algebra (VPA), is proposed in [52] for formal reasoning about SPLs but places emphasis on behavioral (bi)simulation relations rather than verification through model checking. For our work, we use FLan, a feature-oriented language designed to demonstrate how to specify both declarative and procedural aspects of product families. FLan draws inspiration from concurrent constraint programming, combining a store of constraints for declaratively expressing common constraints on features, including cross-tree constraints found in feature models. Additionally, FLan offers a comprehensive set of process-algebraic operators to procedurally specify product configurations and behaviors by supporting a wide range of constraints that can encompass quantitative aspects of feature attributes. Family-based model checking of behavioral SPL models offers a powerful approach to simultaneously verify multiple behavioral product models within a single run. This technique enables the verification of properties using specialized SPL model-checking tools such as SNIP [53], ProVeLines [54], VMC [55, 47, 56], fNuSMV [34, 57], and ProFeat [33] for probabilistic model checking. ProFeat is an example of technique that utilizes numerical computations to achieve precise outcomes when evaluating the properties of a model. Alternatively, traditional model

checkers such as SPIN [58, 59, 60], PRISM [43] for probabilistic model checking, Maude [50], mCRL2 [61, 13], or NuSMV [62] can be employed through appropriate abstractions or encodings. These classical model checkers can effectively verify properties of SPL models by leveraging suitable transformations or encodings to adapt them for SPL-specific analyses. In comparison to conventional product-based model checking approaches, QFLan's statistical model-checking features provide several noteworthy benefits. Firstly, the process of performing simulations can be effortlessly parallelized and distributed across multiple cores, clusters, or distributed computing systems, resulting in nearly linear improvements in processing speed. This parallelization capability enables significant acceleration of the overall verification process. Secondly, the same set of simulations can be utilized to evaluate multiple properties simultaneously, leading to a reduction in the computational time required for verifying each property individually. This simultaneous checking of multiple properties further enhances the efficiency of the verification process. About enhancing SMC techniques with PM techniques, besides our preliminary work [12], directed to demonstrate the potentiality of these techniques applied on a threat model, to best of our knowledge there are no other previous works that apply PM techniques to probabilistic model checking on SPLs models.

8. Conclusion

We presented a novel approach for the validation of simulation models, and in particular software product lines from product lines engineering. The methodology consists of a combination of simulation-based analysis techniques from statistical model checking (SMC) [2], and process-oriented data-driven techniques from process mining (PM) [11]. In particular, we use PM to explain SMC analyses, obtaining a graphical representation of the system behavior as observed in the SMC simulations. In our experimental evaluation, we demonstrate that: (1) our methodology helps in identifying issues in the model, and in getting hints on how to fix them (2) it scales to complex models, and (3) it is general because it can be applied to domains beyond product line engineering.

In future investigations, we will investigate if PM can further help in the product line engineering. For example, it might be useful to compare different products from the same family. Furthermore, we will also study whether PM can further help in solving issues of SMC, e.g., how to handle

rare events [63]. We already considered two different domains, namely product lines engineering, and risk modeling and analysis, analyzed for SMC. In the future, it might be interesting to consider further types of probabilistic models and of analysis techniques like, e.g., pGCL programs studied with used in probabilistic model checkers like STORM [3]. Finally, we plan to investigate the application of our methodology to further frameworks for dynamic PL models. A notable example is ProFeat [33]. It is built on top of the probabilistic model checker PRISM [64], and allows as well for SMC-based analyses.

References

[1] M. H. ter Beek, A. Legay, A. L. Lafuente, A. Vandin, Quantitative security risk modeling and analysis with RisQFLan, Computers & Security 109 (2021) 102381.

[2] G. Agha, K. Palmskog, A survey of statistical model checking, ACM Trans. Model. Comp. Simul. 28 (1) (2018) 6:1–6:39.

[3] H. L. Younes, Probabilistic verification for "black-box" systems, in: CAV 2015, Springer, 2005, pp. 253–265.

[4] A. Vandin, M. H. ter Beek, A. Legay, A. Lluch-Lafuente, QFLan: A Tool for the Quantitative Analysis of Highly Reconfigurable Systems, in: FM, 2018.

[5] M. H. ter Beek, F. Mazzanti, A. Sulova, VMC: A Tool for Product Variability Analysis, in: FM 2012, Vol. 7436 of LNCS, 2012, pp. 450–454. doi:10.1007/978-3-642-32759-9_36.

[6] A. Classen, P. Heymans, P.-Y. Schobbens, A. Legay, J.-F. Raskin, Model Checking Lots of Systems: Efficient Verification of Temporal Properties in Software Product Lines, in: ICSE 2010, ACM, 2010, pp. 335–344. doi:10.1145/1806799.1806850.

[3] https://www.stormchecker.org/documentation/background/languages.html#cpgcl

[7] M. H. ter Beek, E. P. de Vink, Using mCRL2 for the Analysis of Software Product Lines, in: FormaliSE 2014, ACM, 2014, pp. 31–37. doi:10.1145/2593489.2593493.

[8] M. H. ter Beek, A. Legay, A. Lluch Lafuente, A. Vandin, Quantitative Analysis of Probabilistic Models of Software Product Lines with Statistical Model Checking, in: FMSPLE 2015, Vol. 182 of EPTCS, 2015, pp. 56–70. doi:10.4204/EPTCS.182.5.

[9] R. Muschevici, J. Proença, D. Clarke, Feature Nets: behavioural modelling of software product lines, Softw. Sys. Model. 15 (4) (2016) 1181–1206. doi:10.1007/s10270-015-0475-z.

[10] M. H. ter Beek, E. P. de Vink, T. A. C. Willemse, Family-Based Model Checking with mCRL2, in: FASE 2017, Vol. 10202 of LNCS, 2017, pp. 387–405. doi:10.1007/978-3-662-54494-5_23.

[11] W. M. van der Aalst, Process Mining, 2nd Edition, Springer, 2016.

[12] R. Casaluce, A. Burattin, F. Chiaromonte, A. Vandin, Process mining meets statistical model checking: Towards a novel approach to model validation and enhancement, in: C. Cabanillas, N. F. Garmann-Johnsen, A. Koschmider (Eds.), Business Process Management Workshops, Springer International Publishing, Cham, 2023, pp. 243–256.

[13] M. H. ter Beek, A. Legay, A. Lluch-Lafuente, A. Vandin, A Framework for Quantitative Modeling and Analysis of Highly (Re)configurable Systems, IEEE Trans. Software Eng. 46 (3) (2020) 321–345.

[14] E. Clarke, O. Grumberg, S. Jha, Y. Lu, H. Veith, Counterexample-guided abstraction refinement, in: E. A. Emerson, A. P. Sistla (Eds.), Computer Aided Verification, Springer Berlin Heidelberg, Berlin, Heidelberg, 2000, pp. 154–169.

[15] H. Hermanns, B. Wachter, L. Zhang, Probabilistic CEGAR, in: A. Gupta, S. Malik (Eds.), Computer Aided Verification, 20th International Conference, CAV 2008, Princeton, NJ, USA, July 7-14, 2008, Proceedings, Vol. 5123 of Lecture Notes in Computer Science, Springer, 2008, pp. 162–175. doi:10.1007/978-3-540-70545-1_16.
URL https://doi.org/10.1007/978-3-540-70545-1_16

[16] M. H. ter Beek, F. Damiani, M. Lienhardt, F. Mazzanti, L. Paolini, Efficient static analysis and verification of featured transition systems, Empir. Softw. Eng. 27 (1) (2022) 10. doi:10.1007/s10664-020-09930-8.
URL https://doi.org/10.1007/s10664-020-09930-8

[17] M. Plath, M. Ryan, Feature integration using a feature construct, Science of Computer Programming 41 (1) (2001) 53–84. doi:https://doi.org/10.1016/S0167-6423(00)00018-6.
URL https://www.sciencedirect.com/science/article/pii/S0167642300000186

[18] M. H. ter Beek, A. Legay, A. Lluch-Lafuente, A. Vandin, Statistical analysis of probabilistic models of software product lines with quantitative constraints, in: D. C. Schmidt (Ed.), Proceedings of the 19th International Conference on Software Product Line, SPLC 2015, Nashville, TN, USA, July 20-24, 2015, ACM, 2015, pp. 11–15. doi:10.1145/2791060.2791087.
URL https://doi.org/10.1145/2791060.2791087

[19] K. Czarnecki, S. Helsen, U. W. Eisenecker, Staged Configuration Using Feature Models, in: R. L. Nord (Ed.), Proceedings of the 3rd International Software Product Lines Conference (SPLC'04), Vol. 3154 of LNCS, Springer, 2004, pp. 266–283. doi:10.1007/978-3-540-28630-1_17.

[20] J. Bürdek, S. Lity, M. Lochau, M. Berens, U. Goltz, A. Schürr, Staged Configuration of Dynamic Software Product Lines with Complex Binding Time Constraints, in: P. Collet, A. Wąsowski, T. Weyer (Eds.), Proceedings of the 8th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'14), ACM, 2014. doi:10.1145/2556624.2556627.

[21] A. Vandin, D. Giachini, F. Lamperti, F. Chiaromonte, Automated and distributed statistical analysis of economic agent-based models, Journal of Economic Dynamics and Control (2022) 104458.

[22] D. Pianini, S. Sebastio, A. Vandin, Distributed statistical analysis of complex systems modeled through a chemical metaphor, in: HPCS, 2014, pp. 416–423.

[23] S. Gilmore, M. Tribastone, A. Vandin, An analysis pathway for the quantitative evaluation of public transport systems, in: IFM, 2014.

[24] V. Ciancia, D. Latella, M. Massink, R. Paškauskas, A. Vandin, A tool-chain for statistical spatio-temporal model checking of bike sharing systems, in: ISOLA'17, 2017.

[25] M. Bartoletti, J. H. Chiang, T. Junttila, A. Lluch-Lafuente, M. Mirelli, A. Vandin, Formal analysis of lending pools in decentralized finance, in: T. Margaria, B. Steffen (Eds.), Leveraging Applications of Formal Methods, Verification and Validation. Adaptation and Learning - 11th International Symposium, ISoLA 2022, Rhodes, Greece, October 22-30, 2022, Proceedings, Part III, Vol. 13703 of Lecture Notes in Computer Science, Springer, 2022, pp. 335–355. `doi:10.1007/978-3-031-19759-8_21`. URL `https://doi.org/10.1007/978-3-031-19759-8_21`

[26] F. Corradini, F. Fornari, A. Polini, B. Re, F. Tiezzi, A. Vandin, A formal approach for the analysis of BPMN collaboration models, JSS 180 (2021) 111007.

[27] L. Belzner, R. De Nicola, A. Vandin, M. Wirsing, Reasoning (on) service component ensembles in rewriting logic, in: Spec., Alg., and Soft., 2014, pp. 188–211.

[28] A. Weijters, W. M. van Der Aalst, A. A. De Medeiros, Process mining with the heuristics miner-algorithm, Technische Universiteit Eindhoven, Tech. Rep. WP 166 (July 2017) (2006) 1–34.

[29] S. Sebastio, A. Vandin, MultiVeStA: statistical model checking for discrete event simulators, in: 7th Int. Conf ValueTools'13, ICST/ACM, 2013, pp. 310–315.

[30] A. Berti, S. J. van Zelst, W. M. P. van der Aalst, Process mining for python (pm4py): Bridging the gap between process- and data science, CoRR abs/1905.06169 (2019). `arXiv:1905.06169`. URL `http://arxiv.org/abs/1905.06169`

[31] A. S. Dimovski, A. S. Al-Sibahi, C. Brabrand, A. Wąsowski, Efficient family-based model checking via variability abstractions, International Journal on Software Tools for Technology Transfer 19 (2017) 585–603.

[32] M. Cordy, P. Schobbens, P. Heymans, A. Legay, Beyond boolean product-line model checking: dealing with feature attributes and multi-features, in: D. Notkin, B. H. C. Cheng, K. Pohl (Eds.), 35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013, IEEE Computer Society, 2013, pp. 472–481. doi:10.1109/ICSE.2013.6606593.
URL https://doi.org/10.1109/ICSE.2013.6606593

[33] P. Chrszon, C. Dubslaff, S. Klüppelholz, C. Baier, Profeat: feature-oriented engineering for family-based probabilistic model checking, Formal Aspects Comput. 30 (1) (2018) 45–75. doi:10.1007/s00165-017-0432-4.
URL https://doi.org/10.1007/s00165-017-0432-4

[34] A. Classen, M. Cordy, P. Heymans, A. Legay, P. Schobbens, Formal semantics, modular specification, and symbolic verification of product-line behaviour, Sci. Comput. Program. 80 (2014) 416–439. doi:10.1016/j.scico.2013.09.019.
URL https://doi.org/10.1016/j.scico.2013.09.019

[35] S. Apel, A. von Rhein, P. Wendler, A. Größlinger, D. Beyer, Strategies for product-line verification: case studies and experiments, in: D. Notkin, B. H. C. Cheng, K. Pohl (Eds.), 35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013, IEEE Computer Society, 2013, pp. 482–491. doi:10.1109/ICSE.2013.6606594.
URL https://doi.org/10.1109/ICSE.2013.6606594

[36] A. Classen, P. Heymans, P.-Y. Schobbens, A. Legay, Symbolic model checking of software product lines, in: Proceedings of the 33rd International Conference on Software Engineering (ICSE'11), ACM, 2011, pp. 321–330. doi:10.1145/1985793.1985838.

[37] N. Macedo, J. Brunel, D. Chemouil, A. Cunha, D. Kuperberg, Lightweight Specification and Analysis of Dynamic Systems with Rich Configurations, in: Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE'16), ACM, 2016, pp. 373–383. doi:10.1145/2950290.2950318.

[38] J. Meinicke, C. Wong, C. Kästner, T. Thüm, G. Saake, On essential configuration complexity: measuring interactions in highly-configurable systems, in: D. Lo, S. Apel, S. Khurshid (Eds.), Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE'16), ACM, 2016, pp. 483–494. doi:10.1145/2970276.2970322.

[39] H. Sabouri, M. M. Jaghoori, F. S. de Boer, R. Khosravi, Scheduling and Analysis of Real-Time Software Families, in: X. Bai, F. Belli, E. Bertino, C. K. Chang, A. Elçi, C. C. Seceleanu, H. Xie, M. Zulkernine (Eds.), Proceedings of the 36th Annual IEEE Computer Software and Applications Conference (COMPSAC'12), IEEE, 2012, pp. 680–689. doi:10.1109/COMPSAC.2012.95.

[40] Research and Technology Organisation of NATO, Improving Common Security Risk Analysis report, RTO Technical Report TR-IST-049 (2008).

[41] U.S. Department of Defense, Defense Acquisition Guidebook, Section 8.5.3.3 (2009).
URL https://bit.ly/3NJjs07

[42] J. Hu, H. Niu, J. Carrasco, B. Lennox, F. Arvin, Fault-tolerant cooperative navigation of networked uav swarms for forest fire monitoring, Aerospace Science and Technology 123 (2022) 107494.

[43] C. Dubslaff, C. Baier, S. Klüppelholz, Probabilistic model checking for feature-oriented systems, LNCS Trans. Aspect Oriented Softw. Dev. 12 (2015) 180–220. doi:10.1007/978-3-662-46734-3_5.
URL https://doi.org/10.1007/978-3-662-46734-3_5

[44] A. Classen, P. Heymans, P. Schobbens, A. Legay, Symbolic model checking of software product lines, in: R. N. Taylor, H. C. Gall, N. Medvidovic (Eds.), Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, Waikiki, Honolulu , HI, USA, May 21-28, 2011, ACM, 2011, pp. 321–330. doi:10.1145/1985793.1985838.
URL https://doi.org/10.1145/1985793.1985838

[45] M. Cordy, A. Classen, P. Heymans, A. Legay, P. Schobbens, Model checking adaptive software with featured transition systems, in: J. Cá-

mara, R. de Lemos, C. Ghezzi, A. Lopes (Eds.), Assurances for Self-Adaptive Systems - Principles, Models, and Techniques, Vol. 7740 of Lecture Notes in Computer Science, Springer, 2013, pp. 1–29. doi:10.1007/978-3-642-36249-1_1.
URL https://doi.org/10.1007/978-3-642-36249-1_1

[46] D. Fischbein, S. Uchitel, V. A. Braberman, A foundation for behavioural conformance in software product line architectures, in: R. M. Hierons, H. Muccini (Eds.), Proceedings of the 2006 Workshop on Role of Software Architecture for Testing and Analysis, held in conjunction with the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2006), ROSATEA 2006, Portland, Maine, USA, July 17-20, 2006, ACM, 2006, pp. 39–48. doi:10.1145/1147249.1147254.
URL https://doi.org/10.1145/1147249.1147254

[47] M. H. ter Beek, A. Fantechi, S. Gnesi, F. Mazzanti, Modelling and analysing variability in product families: Model checking of modal transition systems with variability constraints, J. Log. Algebraic Methods Program. 85 (2) (2016) 287–315. doi:10.1016/j.jlamp.2015.11.006.
URL https://doi.org/10.1016/j.jlamp.2015.11.006

[48] M. Leucker, D. Thoma, A formal approach to software product families, in: T. Margaria, B. Steffen (Eds.), Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change - 5th International Symposium, ISoLA 2012, Heraklion, Crete, Greece, October 15-18, 2012, Proceedings, Part I, Vol. 7609 of Lecture Notes in Computer Science, Springer, 2012, pp. 131–145. doi:10.1007/978-3-642-34026-0_11.
URL https://doi.org/10.1007/978-3-642-34026-0_11

[49] M. Erwig, E. Walkingshaw, The choice calculus: A representation for software variation, ACM Trans. Softw. Eng. Methodol. 21 (1) (2011) 6:1–6:27. doi:10.1145/2063239.2063245.
URL https://doi.org/10.1145/2063239.2063245

[50] M. Lochau, S. Mennicke, H. Baller, L. Ribbeck, Incremental model checking of delta-oriented software product lines, J. Log. Algebraic Methods Program. 85 (1) (2016) 245–267. doi:10.1016/j.jlamp.2015.09.004.
URL https://doi.org/10.1016/j.jlamp.2015.09.004

[51] D. Clarke, M. Helvensteijn, I. Schaefer, Abstract delta modeling, in: E. Visser, J. Järvi (Eds.), Generative Programming And Component Engineering, Proceedings of the Ninth International Conference on Generative Programming and Component Engineering, GPCE 2010, Eindhoven, The Netherlands, October 10-13, 2010, ACM, 2010, pp. 13–22. doi:10.1145/1868294.1868298.
URL https://doi.org/10.1145/1868294.1868298

[52] M. Tribastone, Behavioral relations in a process algebra for variants, in: S. Gnesi, A. Fantechi, P. Heymans, J. Rubin, K. Czarnecki, D. Dhungana (Eds.), 18th International Software Product Line Conference, SPLC '14, Florence, Italy, September 15-19, 2014, ACM, 2014, pp. 82–91. doi:10.1145/2648511.2648520.
URL https://doi.org/10.1145/2648511.2648520

[53] A. Classen, M. Cordy, P. Heymans, A. Legay, P. Schobbens, Model checking software product lines with SNIP, Int. J. Softw. Tools Technol. Transf. 14 (5) (2012) 589–612. doi:10.1007/s10009-012-0234-1.
URL https://doi.org/10.1007/s10009-012-0234-1

[54] M. Cordy, A. Classen, P. Heymans, P. Schobbens, A. Legay, Provelines: a product line of verifiers for software product lines, in: 17th International Software Product Line Conference co-located workshops, SPLC 2013 workshops, Tokyo, Japan - August 26 - 30, 2013, ACM, 2013, pp. 141–146. doi:10.1145/2499777.2499781.
URL https://doi.org/10.1145/2499777.2499781

[55] M. H. ter Beek, F. Mazzanti, A. Sulova, VMC: A tool for product variability analysis, in: D. Giannakopoulou, D. Méry (Eds.), FM 2012: Formal Methods - 18th International Symposium, Paris, France, August 27-31, 2012. Proceedings, Vol. 7436 of Lecture Notes in Computer Science, Springer, 2012, pp. 450–454. doi:10.1007/978-3-642-32759-9_36.
URL https://doi.org/10.1007/978-3-642-32759-9_36

[56] M. H. ter Beek, F. Mazzanti, VMC: recent advances and challenges ahead, in: S. Gnesi, A. Fantechi, M. H. ter Beek, G. Botterweck, M. Becker (Eds.), 18th International Software Product Lines Conference - Companion Volume for Workshop, Tools and Demo papers, SPLC '14, Florence, Italy, September 15-19, 2014, ACM, 2014, pp. 70–77.

doi:10.1145/2647908.2655969.
URL https://doi.org/10.1145/2647908.2655969

[57] A. S. Dimovski, A. Legay, A. Wasowski, Variability abstraction and refinement for game-based lifted model checking of full CTL, in: R. Hähnle, W. M. P. van der Aalst (Eds.), Fundamental Approaches to Software Engineering - 22nd International Conference, FASE 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Vol. 11424 of Lecture Notes in Computer Science, Springer, 2019, pp. 192–209. doi:10.1007/978-3-030-16722-6_11.
URL https://doi.org/10.1007/978-3-030-16722-6_11

[58] SPIN 2015: Proceedings of the 22nd International Symposium on Model Checking Software - Volume 9232, Springer-Verlag, Berlin, Heidelberg, 2015.

[59] A. S. Dimovski, A. S. Al-Sibahi, C. Brabrand, A. Wasowski, Efficient family-based model checking via variability abstractions, Int. J. Softw. Tools Technol. Transf. 19 (5) (2017) 585–603. doi:10.1007/s10009-016-0425-2.
URL https://doi.org/10.1007/s10009-016-0425-2

[60] A. S. Dimovski, A. Wasowski, Variability-specific abstraction refinement for family-based model checking, in: M. Huisman, J. Rubin (Eds.), Fundamental Approaches to Software Engineering - 20th International Conference, FASE 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Vol. 10202 of Lecture Notes in Computer Science, Springer, 2017, pp. 406–423. doi:10.1007/978-3-662-54494-5_24.
URL https://doi.org/10.1007/978-3-662-54494-5_24

[61] M. H. ter Beek, E. P. de Vink, T. A. C. Willemse, Family-based model checking with mcrl2, in: M. Huisman, J. Rubin (Eds.), Fundamental Approaches to Software Engineering - 20th International Conference, FASE 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Vol. 10202 of Lecture Notes in Computer Science,

Springer, 2017, pp. 387–405. doi:10.1007/978-3-662-54494-5_23.
URL https://doi.org/10.1007/978-3-662-54494-5_23

[62] A. S. Dimovski, $\hbox {CTL}^{\star }$ family-based model checking using variability abstractions and modal transition systems, Int. J. Softw. Tools Technol. Transf. 22 (1) (2020) 35–55. doi:10.1007/s10009-019-00528-0.
URL https://doi.org/10.1007/s10009-019-00528-0

[63] A. Legay, S. Sedwards, L. Traonouez, Rare events for statistical model checking an overview, in: K. G. Larsen, I. Potapov, J. Srba (Eds.), Reachability Problems - 10th International Workshop, RP 2016, Aalborg, Denmark, September 19-21, 2016, Proceedings, Vol. 9899 of Lecture Notes in Computer Science, Springer, 2016, pp. 23–35. doi:10.1007/978-3-319-45994-3_2.
URL https://doi.org/10.1007/978-3-319-45994-3_2

[64] M. Z. Kwiatkowska, G. Norman, D. Parker, PRISM 4.0: Verification of probabilistic real-time systems, in: G. Gopalakrishnan, S. Qadeer (Eds.), Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings, Vol. 6806 of Lecture Notes in Computer Science, Springer, 2011, pp. 585–591. doi:10.1007/978-3-642-22110-1_47.
URL https://doi.org/10.1007/978-3-642-22110-1_47

9. Vitae

Roberto Casaluce is a PhD student in Artificial Intelligence for Society at the University of Pisa and the Sant'Anna School of Advanced Studies, Pisa. He completed an MSc in Politics and an MSc in Data Science at Birkbeck, University of London. For his latter master's thesis he worked on a project within the ETP Group of ETH Zurich using NLP methods to classify Clean Energy Storage Technologies patents. After submitting his master's thesis, he worked as a research assistant on the same project for six months. His research interests include statistical model checking techniques, process mining, and NLP.

Andrea Burattin is Associate Professor at the Technical University of Denmark since April 2019. Previously, he worked as Assistant Professor at

the same university, and as postdoctoral researcher at the University Innsbruck (Austria) and at the University of Padua (Italy). In 2013 he obtained his PhD degree from a joint PhD School between the University of Bologna and Padua (Italy). His PhD thesis received the Best Process Mining Dissertation Award from the IEEE Task Force on Process Mining. He is member of the steering committee of the IEEE Task Force on Process Mining.

Francesca Chiaromonte is a Professor of Statistics at the Sant'Anna School of Advanced Studies, where she is the Scientific Coordinator of EMbeDS – a department of excellence fostering data- and computation-intensive research in the social sciences. She is a member of the board of the National Italian PhD in AI and Society, and she holds the Dorothy Foehr Huck and J. Lloyd Huck Chair in Statistics for the Life Sciences at the Pe nnsylvania State University. Francesca is a fellow of the American Statistical Association since 2016, and a fellow of the Institute for Mathematical Statistics since 2022.

Alberto Lluch Lafuente is Full Professor at the Technical University of Denmark. He is currently head of the section for Software Systems Engineering. Previously, he was Associate Professor at the Technical University of Denmark, Assistant Professor at IMT School for Advanced Studies Lucca and Postdoctoral Researcher at the University of Pisa. He obtained his PhD degree from the Albert-Ludwigs Universität in Freiburg in 2003.

Andrea Vandin is Associate Professor in Computer Science at Sant'Anna School of Advanced Studies, Pisa, since December 2022, as well as Adjunct Associate Professor at DTU Technical University of Denmark. Previously, he worked as Associate Professor at DTU, and Assistant Professor at IMT School for Advanced Studies Lucca, and the University of Southampton UK. In 2013, he obtained his PhD degree in Computer Science and Engineering at IMT. He is member of the board of the National Italian PhD in AI and Society.

Appendix A. Addressing typos in models

Figure A.23 lists a partial probabilistic process of the elevator model with 5 floors and involving four concurrent processes. This example highlights how a minor typo that could easily be overlooked when specifying the model, especially in complex models like the elevator, might be spotted by our methdology. The typo is in line 8, where, instead of adding a weight greater than zero,

```
begin processes diagram
  begin process LiftProc //... Lift Process
  states = Lift,LiftTurnButtonDown
  transitions=
      Lift -(open,1 ,{door=1})-> LiftTurnButtonDown,
      Lift -(close,1 ,{door=0})-> Lift,
      Lift -(up,1 ,{floor=floor + 1})-> Lift,
      Lift -(down,0 ,{floor=floor - 1})-> Lift,
      Lift -(clean,100 ,{buttonL0=0,
                buttonL1=0,buttonL2=0,
                buttonL3=0, buttonL4=0})-> Lift,

  LiftTurnButtonDown -(ask({floor==0}),100,{buttonL0=0,buttonF0= 0})-> Lift,
  LiftTurnButtonDown -(ask({floor==1}),100,{buttonL1=0,buttonF1= 0})-> Lift,
  LiftTurnButtonDown -(ask({floor==2}),100,{buttonL2=0,buttonF2= 0})-> Lift,
  LiftTurnButtonDown -(ask({floor==3}),100,{buttonL3=0,buttonF3= 0})-> Lift,
  LiftTurnButtonDown -(ask({floor==4}),100,{buttonL4=0,buttonF4= 0})-> Lift
  end process

 begin process ControllerProc
    //... Controller Process
    end process

 begin process ButtonsProc
    //... Button Process
     end process

 begin process PeopleProc
    //... People Process
    end process
```

Figure A.23: Probabilistic processes of the Elavator model in QFLan

it is erroneously set to zero. As expected, the natural consequence of this typo is that the simulator will not traverse that transition, resulting in unexpected behavior and therefore biased SMC analysis. Figure A.24 illustrates the diff model, displaying only the relevant sections of the four concurrent processes. Among these sections, there exists at least one transition that the simulator did not traverse due to the typo in line 8, identifiable by the dashed red edge. The original model and the full diff model generated by our methodology can be accessed in `https://doi.org/10.5281/zenodo.8362717`.

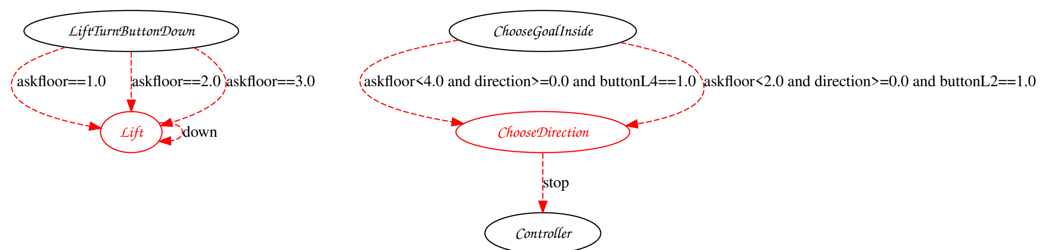

Figure A.24: Excerpt of the diff model for the elevator model with 5 floors with the typo.