Adaptive Behavioral Model Learning for Software Product Lines

Shaghayegh Tavassoli
sh.tavassoli@ut.ac.ir
University of Tehran
Tehran, IR

Carlos Diego N. Damasceno
d.damasceno@cs.ru.nl
Radboud University Nijmegen
Nijmegen, NL

Ramtin Khosravi
r.khosravi@ut.ac.ir
University of Tehran
Tehran, IR

Mohammad Reza Mousavi
mohammad.mousavi@kcl.ac.uk
King's College London
London, UK

ABSTRACT

Behavioral models enable the analysis of the functionality of software product lines (SPL), e.g., model checking and model-based testing. Model learning aims to construct behavioral models. Due to the commonalities among the products of an SPL, it is possible to reuse the previously-learned models during the model learning process. In this paper, an adaptive approach, called PL*, for learning the product models of an SPL is presented based on the well-known L^* algorithm. In this method, after learning each product, the sequences in the final observation table are stored in a repository which is used to initialize the observation table of the remaining products. The proposed algorithm is evaluated on two open-source SPLs and the learning cost is measured in terms of the number of rounds, resets, and input symbols. The results show that for complex SPLs, the total learning cost of PL* is significantly lower than that of the non-adaptive method in terms of all three metrics. Furthermore, it is observed that the order of learning products affects the efficiency of PL*. We introduce a heuristic to determine an ordering which reduces the total cost of adaptive learning.

CCS CONCEPTS

• **Networks → Formal specifications**; • **Theory of computation → Query learning**; • **Hardware → Finite state machines**; • **Software and its engineering → Software product lines**.

KEYWORDS

Adaptive Model Learning, Software Product Lines, Automata Learning, Finite State Machines

ACM Reference Format:
Shaghayegh Tavassoli, Carlos Diego N. Damasceno, Ramtin Khosravi, and Mohammad Reza Mousavi. 2022. Adaptive Behavioral Model Learning for Software Product Lines. In *26th ACM International Systems and Software Product Line Conference - Volume A (SPLC '22), September 12–16, 2022, Graz, Austria.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3546932.3546991

1 INTRODUCTION

Models are the foundations of many rigorous analysis techniques in engineering in general and software engineering in particular. Behavioral models specify how a system behaves as a result of interacting with its user and environment. Examples of behavioral models include variants of state machines and sequence diagrams. Behavioral models are often non-existent or outdated and one needs to reconstruct them from implementations in order to enable further analysis [12]. Model learning is a mechanized approach that comes to rescue in such situations [41].

In software product lines, model learning is challenged by variability [15, 43]: one needs to learn behavioral models over the variability space and if performed crudely, this can be practically impossible. The key to overcome this challenge is to reuse the learned models and their underlying data structures while moving across the variability space [42]. Adaptive model learning [17] is fit for this purpose, because its algorithms are precisely designed to reuse the results of the past queries, as well as the structure of the behavioral models in the subsequent learning process [11].

In this paper, we design an adaptive learning algorithm for software product lines, called PL^*, and evaluate its efficiency against its non-adaptive counterparts. Two important components of a model learning algorithm are the membership queries (checking for the output to a given sequence of inputs) and the equivalence queries (verifying the model learned hitherto). The main factors in evaluating the efficiency of a model learning algorithm are the number of learning rounds (and equivalence queries), the number of resets, and the total number of input symbols used in learning [2, 11, 41]. We evaluate the efficiency of our algorithm on two subject systems. We statistically evaluate our results (with 3-wise sampling, based on earlier experiments [12], and with different product orderings) and observe, with high statistical confidence, that PL* is more efficient than the non-adaptive approach. Hence, we affirmatively answer the following research questions:

- RQ1 Does adaptive learning lead to fewer learning rounds and equivalence queries?
- RQ2 Does adaptive learning lead to fewer resets?
- RQ3 Does adaptive learning lead to fewer total number of input symbols?

In standard (non-adaptive) model learning, the order of learning the products is immaterial, since no information is brought forward to learning the next products. In our adaptive learning method, we observe a stark difference in terms of efficiency among different orders. We define and formalize a heuristic to provide an efficient product ordering in the learning process and statistically establish a correlation between our proposed ordering and the efficiency of the learning algorithm (in terms of the total number of resets and the total number of input symbols) as the answer to our last research question:

RQ4 How does the choice of product ordering influence the efficiency of the learning process?

To our knowledge, this is the first application of adaptive model learning to software product lines (we refer to Section 2 for an analysis of the related work). It is a first step in this direction, which will pave the way for a line of research extending various model learning techniques to a family-based approach. This natural extension would require parameterizing the data structure we use in our approach (called observation tables [2]) with feature expressions.

The rest of this paper is organized as follows. In Section 2, we review the related work and position our research within the broader fields of model learning and software product lines. In Section 3, we recall some basic concepts and definitions from the aforementioned fields. In Section 4, we present our adaptive model learning algorithm. In Section 5, we outline our empirical evaluation methodology and describe the design of our experiments. In Section 6, we discuss the results of our experiments and reflect on the threat to the validity of our results. In Section 7, we conclude the paper and present the directions of our future research. A package containing the source code, models, and test scripts is available at https://github.com/sh-t-20/artifacts

2 RELATED WORK

In this section, we review the related work in three broad areas: adaptive model learning [5, 11, 17, 21, 45, 46], machine learning in SPLs [10, 12, 28, 31], and feature model mining [1, 20, 34].

Adaptive Model Learning. Adaptive model learning [17] is an extension of traditional model learning by reusing pre-existing models. Groce, Peled & Yannakakis [17] are among the first to reuse inaccurate models for adaptive model learning and model checking. The authors' results suggest that adaptive learning is especially useful when model updates are led by small changes with limited impact [17]. Our results corroborate their observation in the setting of software product lines; namely, we show that adaptive learning is more efficient when the order of products comprises fewer new non-mandatory features added in each step. Windmüller et al. [45] show that adaptive learning can be used to periodically build models from evolving complex applications. Also, they show that reusing separating sequences derived from models of previous versions can steer the learning process to find maintained states [45]. Huistra, Meijer, & van de Pol [21] report that the performance of adaptive learning is influenced by the SUL's complexity, the size of its update, and the quality of suffixes. Additionally, the authors report evidence that, if a set of reused separating sequences has low state distinguishing capacity, then irrelevant queries should be expected [21]. Chaki, Clarke, Sharygina & Sinha [5] presented an approach for efficiently model checking software upgrades by revalidating sequences from reused observation tables [5]. More recently, Damasceno et al. [11] showed that existing adaptive learning techniques are prone to performance issues when there are large differences between the reused and updated model. To address this issue, they introduced a novel adaptive learning technique that gradually revalidates and reuses sequences and outperforms state-of-the-art adaptive learning techniques [11]. Our results crucially build upon these earlier results and bring them to a new domain: software product lines provide a specific paradigm for adaptive learning, where the choice of adaptations can be controlled by the product sampling order. Our work differs from these by focusing on the reuse of multiple observation tables in an observation table repository and is to our knowledge the first attempt to lift adaptive model learning to the scope of software families. Yang et al. [46] present a way to combine the results of passive- and active model learning [46]. Our work differs from this piece of work in that we consider active model learning; the combination of adaptive active and passive learning for product families is a promising area for future work.

Machine and Model Learning in SPLs. Several studies applied machine learning [31, 38] and model learning [10, 12] in software product lines. For an extensive literature review on machine learning applied to SPLs, we refer the interested reader to Pereira et al. [31]. Lesoil [28] have recently showed that *variability* can be present at multiple layers of a system (e.g., at the hardware, software, and input data levels) and raise concerns about challenges in the adoption of machine learning principles in variability analysis. Family-based modeling approaches have been developed to enable efficient model-based testing of SPLs without exhaustively going through each and every product. Nevertheless, the creation and maintenance of family models are still difficult and time-consuming [30]. To mitigate this, Damasceno et al. [10, 12] introduced family model learning as a means for building behavioural variability models for SPLs. Using a benchmark set of 105 product models, the authors showed that succinct family models can be learned by matching and merging state machine models [12], particularly when there is a high degree of reuse among the SULs [10]. Additionally, they show that feature coverage criteria (particular, up to 3-wise) can alleviate the costs of learning family model by sampling product sets that cover the behavior of product families. These results have sparked the interest of the SPL community in pursuing further investigations at the intersection of model learning and variability analysis [13]. Our work advances this research line by reusing observation tables across multiple products. Particularly, using adaptive learning techniques is a novelty of our approach.

Feature Model Mining. Feature models are a key asset in variability management and analysis [4]. Using SAT- [27] or SMT [37] solvers, feature models are amenable to automated reasoning. However, as an SPL may also be built using extractive and reactive approaches [3], SPL projects may initially lack feature models [20]. To address this issue, reverse engineering concepts have been used to (semi-)automate the construction of feature models from sets of product configurations [1, 20, 34]. Our work complements the role of such structural variability model learning techniques (aka feature model mining) by providing an efficient means to extract behavioral variability models. The integration of these two sets of techniques is a promising line of future research.

3 BACKGROUND

In this section, some of the terms used in this paper are described. The software product lines are briefly explained. Some notations for modeling an SPL and its products are defined. The non-adaptive model learning process is explained. Some of the metrics used for evaluating the efficiency of model learning are defined. Also in this section, the product sampling concept is briefly described.

3.1 Software Product Lines

A software product line (SPL) is a set of software products that have a common set of features and are designed for a specific requirement [9]. An SPL is defined by a set of features F and a feature model [10, 12]. A feature-model [26, 35] is a structural variability model representing the hierarchical structure of the SPL features. Each product p consists of a subset of the SPL features. The set of valid product configurations is specified by the feature model. In this representation, features are classified into mandatory and optional types. Mandatory features are present in all valid configurations by default. From a group of *alternative* features, only one of them can be present in each product. When a set of features are defined using *or*, each product may contain one or more of them [12, 26]. Figure 1 shows the feature model of a sample SPL. In this figure, A and C are mandatory features and B is an optional feature. The features D and E are alternatives. F, G and H form an 'or' group of features. This SPL contains 28 valid product configurations.

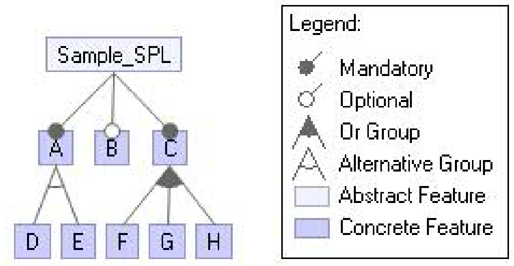

Figure 1: The feature model of a sample SPL

3.2 Finite State Machines

A finite state machine (FSM) [16] is a widely used behavioral model which is defined as the tuple $M = \langle S, s_0, I, O, \delta, \lambda\rangle$. In this definition, S is the set of states and s_0 is the initial state ($s_0 \in S$). The set of input alphabet is represented by I and the set of outputs is denoted by O. The transition function δ determines the next state, $s_2 \in S$, assuming that the FSM is in state $s_1 \in S$ and the input $a \in I$ is presented ($\delta(s_1, a) = s_2$). The output function is represented by λ which is a mapping from a pair of a state and an input to an output. The state machines learned by the existing model learning methods are deterministic FSMs. In a deterministic FSM, for each state and input alphabet, there is at most one transition and one output [11, 41].

3.3 Model Learning

Model learning [41] is a method used to construct the behavioral model of a software system in the form of a state machine. Model learning is classified into two types: passive and active. In passive learning, different runs of software (e.g., log files) are used for learning a behavioral model of the software. In active learning, however, a model is learned by interacting with the system under learning (SUL) through various types of queries and observing the resulting outputs. The L^* algorithm [2] proposed by Dana Angluin, is a seminal example of active model learning, where two types of queries are used: a membership query (MQ) is used to determine the output sequence for a given input sequence. An equivalence query (EQ) is used to ask if the constructed hypothesis H is language-equivalent to the SUL. The query results are stored in an observation table (described below), and the learning is performed in rounds, in each of which a hypothesis is constructed [2, 11, 41].

3.3.1 Observation Table. An observation table is defined as a triple $OT = (S, E, T)$, where $S \subseteq I^*$ is a finite prefix-closed set of prefixes (transfer sequences); $E \subseteq I^+$ is a finite set of suffixes (separating sequences) and $T : I^+ \times I^+ \longrightarrow I^+$ is a function such that for each $s \in S \cup S.I$ and $e \in E$, $T(s, e)$ is the SUL's output suffix of size $|e|$ for the $s.e$ input sequence. An observation table can be represented as a 2-dimensional array, where each row is a subset of $S.I$ with a representative $s \in S.I$, and each column is a sequence $e \in E$. An observation table is *closed* if for all $s_1 \in S.I$, there exists a prefix $s_2 \in S$ such that $row(s_1)$ equals $row(s_2)$. An observation table is *consistent* if for all $s_1, s_2 \in S$ such that $row(s_1) = row(s_2)$, $row(s_1.v)$ equals $row(s_2.v)$ for all $v \in I$ [2, 11, 41].

3.3.2 The L^* Algorithm. In this section, Angluin's L^* algorithm [2] is described. At the beginning of this algorithm, sets S and E are initialized to $\{\epsilon\}$ and the initial T values are obtained by posing MQs. The following steps are repeated until the observation table is consistent and closed:

- If the observation table is not consistent, the algorithm finds $s_1, s_2 \in S$, $v \in I$ and $e_1 \in E$ such that $row(s_1) = row(s_2)$ and $T(s_1.v, e_1) \neq T(s_2.v, e_1)$. Then, $v.e_1$ is added to E and the new values of T are calculated by posing MQs.
- If the observation table is not closed, the algorithm finds $s_1 \in S$ and $v \in I$ such that $row(s_1.v) \neq row(s)$ for all $s \in S$. Then, $s_1.v$ is added to S and the new values of T are obtained using MQs.

When the observation table is closed and consistent, the algorithm constructs a hypothesis H and poses an EQ to verify it. If H is correct, the learning algorithm terminates. If the hypothesis H is not correct, a counterexample is provided. A counterexample is an input sequence in which the result of H is different from the result of the SUL [2, 11, 41]. Then, the counterexample is used to update the observation table by adding prefixes or suffixes (for which several heuristics have been proposed) [23].

The L^*_M [36] is an active model learning algorithm for learning mealy machines using the settings of L^*. In L^*_M, the observation table is defined as $OT = \{S_M, E_M, T_M\}$ and is initialized using $S_M = \{\epsilon\}$ and $E_M = I$ [36]. In this paper, the L^*_M algorithm is used to perform the experiments.

3.4 Product Sampling

Product-based behavioral analysis of an SPL, can be costly due to the exponential number of valid configurations. Using sample-based approaches may result in increasing the efficiency of the SPL analysis. In these approaches, a subset of valid products is used to cover the behavior of an SPL. Products whose behavior has already been covered by other products are not included in the sample [39]. The T-wise [24] method is one of the sampling techniques applicable in the SPL context. In this method, valid combinations of T-features are used to cover the T-wise interactions of features in the SPL [12, 24, 32].

4 THE PL* ALGORITHM

In adaptive model learning, the transfer sequences and the separating sequences in the observation tables of the existing models are reused to initialize the observation table of the new model [11]. In our approach, we build upon an adaptive learning algorithm [11], and apply to learn the FSM models of a set of products sampled from an SPL. Assume *Sample* = $(p_1, p_2, \ldots, p_n)$ is a sequence of n products sampled from an SPL.

In this algorithm, *otRepository* is defined as a set of observation tables learned from earlier products. To refer to these observation tables, we use the notation $OT_i = (S_i, E_i, T_i)$ for the observation table of product i. At the beginning of the learning process, the *otRepository* is empty. The PL* algorithm consists of the following steps:

(1) First, the FSM of p_1 is learned using a non-adaptive learning method (e.g., using the L^*_M algorithm). We only deviate from L^* by initialising the initial set of suffixes (i.e., E_1) with the alphabet of the product p_1. (We do the same for all other subsequent products, as well, i.e., we add their alphabet to their initial set of suffixes.)

(2) Once the L^*_M algorithm successfully terminates, the resulting observation table OT_1 is added to the *otRepository*. Therefore, *otRepository* equals $\{OT_1\}$.

For each $i \in \{2, \ldots, n\}$, the following steps are iteratively repeated:

(3) The model of product p_i is learned using adaptive L^*_M. In this step, the observation table of p_i is initialized using *otRepository* = $(OT_1, \ldots, OT_{i-1})$. A sequence is "defined in the alphabet of p_i" if it solely comprises input symbols in the alphabet of p_i. To initialize the OT_i, the set of sequences in $\bigcup_{j \in \{1,\ldots,i-1\}} S_j$ which are defined in the alphabet of p_i, is considered as the initial value of S_i. The set of sequences in $\bigcup_{j \in \{1,\ldots,i-1\}} E_j$ which are defined in the alphabet of p_i, are added to E_i (note that the alphabet of p_i is initially added to E_i by default).

(4) Once adaptive L^*_M terminates, OT_i is added to the *otRepository* and at the end of this step, *otRepository* equals $(OT_1, \ldots, OT_i)$.

A schematic representation of the proposed adaptive learning method is shown in Figure 2. In this figure, M_i is the model learned for the i-th product (p_i). In this figure, arrows from PL* processes to the observation table repository show that the observation table of the recently learned product is stored in the repository. The arrows starting from the repository show the sets of sequences in the repository which are used to initialize the observation table of the new products. After learning the model of each product, the learned model is incorporated into a family model (a feature-annotated behavioral model of the software product line [14]). The two processes of learning the product models and updating the family model can be performed concurrently.

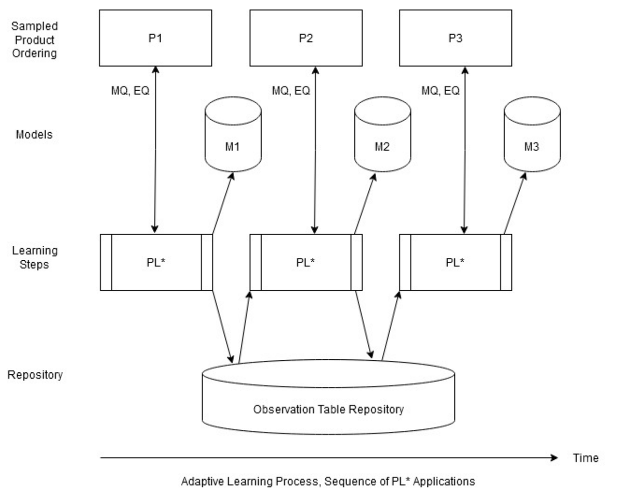

Figure 2: A schematic representation of the proposed adaptive learning method

Based on the results of the experiment described in Section 6.1, we observed that the product learning order can affect the efficiency of the PL* method. The product learning order can be random or it can be determined using heuristic methods. To find a good learning order, it is necessary to determine which characteristics of a learning order result in increasing the efficiency. Based on the results of the experiments (Section 6.1) and observing learning orders with high, medium, and low efficiency, a heuristic is presented to determine the desired learning orders. In the experiments, we observed that when the number of new non-mandatory features that are added by each product is small, the efficiency of the PL* method increases. Using this observation, we present a heuristic to provide an ordering which decreases the total cost of learning.

Suppose the number of non-mandatory features of an SPL is F and a sample of size n from this SPL is available for learning. The product learning order $O = \langle p_1, p_2, ..., p_n \rangle$ is a sequence of products in this sample. If i is smaller than j, the FSM of p_i must be learned earlier than the FSM of p_j. The parameter D is defined as follows.

$$D = \begin{cases} 0, & \text{if } F_i = 0 \\ \sum_{i=1}^{n} \frac{1}{F_i}, & \text{if } F_i \neq 0 \end{cases} \tag{1}$$

In Equation 1, F_i is the number of new non-mandatory features added by p_i, i.e., the number of non-mandatory features in p_i not present in any product p_j, where $1 \leq j < i$. The reason for using $\frac{1}{F_i}$ in this formula is that the added cost of learning decreases as the number of new non-mandatory features increases (i.e., the difference between 1 and $\frac{1}{2}$ is larger than the difference between $\frac{1}{4}$ and $\frac{1}{5}$).

5 EMPIRICAL EVALUATION METHODOLOGY

To evaluate the efficiency of the proposed adaptive learning method, a set of experiments is performed. These experiments are designed to answer the following questions by comparing the quantitative metrics between the PL* method and the non-adaptive learning method:

RQ1 Does adaptive learning lead to fewer total number of rounds and equivalence queries?
RQ2 Does adaptive learning lead to fewer resets?
RQ3 Does adaptive learning lead to fewer total number of input symbols?

These quantitative metrics arise from the way model learning algorithms operate: MQs are the simple and basic building blocks to build a hypothesis about the system under learning and hence, their total number is indicative of how long it takes before the hypotheses are constructed. EQs are much heavier than MQs and their total number heavily influences the performance. In order to put these two types of queries together, one needs to factor in the substantial difference in the size of these two types of queries; this is best achieved by counting the total number of input symbols, which gives us a very natural indicator of the overall performance of the algorithm [41] (RQ3).

While performing these queries, sometimes a reset operation is needed to bring the FSM to a known state and pose further queries. This operation is known to be very costly and is often avoided as much as possible in learning algorithms. To reset a SUL, it may be necessary to completely restart the system and re-initialize many of its software components. Therefore, performing a reset may take a long time [18, 19]. Hence, we use the total number of resets as another efficiency metric for our comparison (RQ2).

5.1 Subject Systems

To evaluate the proposed adaptive learning method, we need access to SPLs with well-defined behavioral (e.g., FSM or labelled transition system) and structural models (e.g., Feature Models and the alphabet of each feature). It must also be possible to obtain the FSM of any valid configuration from these SPLs. The tested SPLs must be complex enough to involve a number of rounds and have a reasonably large number of queries in order to allow for a meaningful comparison. According to the mentioned characteristics, two open-source SPLs are used in the experiments:

5.1.1 The Minepump SPL. The Minepump SPL is presented in [7, 8] and is a simple mine-pump controller that includes 9 features (6 non-mandatory features). The feature model of this SPL is shown in Figure 3 [7]. The featured transition system of this SPL is provided in [12]; Using this featured behavioral model, it is possible to obtain the FSM of any valid configuration from this SPL. Sampling is performed using the 3-wise method. The sample created from this SPL contains 15 products. The FSMs of the products in this sample have a minimum of 9 states and a maximum of 21 states, and their average number of states is 13.86.

5.1.2 The BCS SPL. The Body Comfort System (BCS) SPL [29] is an automotive software system of a Volkswagen Golf model, whose original feature model has 27 features. Each component in

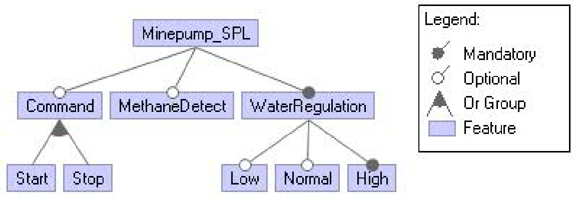

Figure 3: The feature model of the Minepump SPL [7]

this SPL represents a feature and provides a specific functionality. The I/O transition system of each component is provided at [29].

In this paper, a simplified version of the BCS SPL is used. The feature model of the simplified version of the BCS SPL is shown in Figure 4 (taken from [29] with minor modifications). The simplified version contains 12 features (6 non-mandatory features). The product FSMs are constructed using the following steps:

(1) The I/O transition system of each component is converted to a finite state machine (FSM).
(2) The FSMs of the components corresponding to the features of each product are merged to construct the FSM of that product.

The sample created from this SPL using the 3-wise method contains 16 products. The FSMs of the products in this sample have a minimum of 14 states and a maximum of 864 states, and their average number of states is 117.25.

5.2 Experiment Design

To perform the experiments, a subset of the valid configurations of each subject SPL is used. The samples are produced using the T-wise product sampling method [24] and the Chvatal algorithm [6]. For sampling using T-wise method, the value of T is set to 3. In [12], it is shown that in T-wise sampling method, the use of T = 3 results in a more precise family model than in cases where T = 1 or T = 2. In this sampling method, using T greater then 3 is not cost-effective [12]. Sampling is performed using the FeatureIDE [40] library. The FSMs of all products in each sample are learned using the PL* method and the non-adaptive learning method. The total learning cost is calculated for each learning method.

In these experiments, model learning is performed using the ExtensibleLStarMealyBuilder class of the LearnLib [33] library version 0.16.0. In the non-adaptive learning method, before applying model learning to each product, the observation table is initialized using $S_M = \epsilon$ and $E_M = I$, where I is the input alphabet of the product [36]. In the adaptive learning method, the observation tables are initialized using the method explained in Section 3 (the PL* algorithm). Model learning is performed using the following parameters:

- The equivalence oracle type is WP, which is an established and structured method for detecting faults (i.e., incorrectly learned states and transitions).
- The observation table closing strategy is CloseFirst.
- Caching is not used.

To evaluate the experiment results, statistical tests are performed using the SciPy [25] library of Python. The Matplotlib [22] library is used to visualize the results. The experiments show that in the

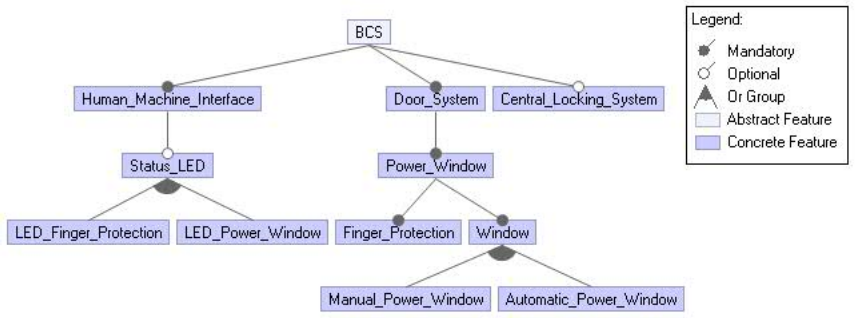

Figure 4: The feature model of a simplified version of the BCS SPL (inspired by [29])

PL* method, the order of learning the products affects the total cost of learning. To more accurately evaluate the results, the following experiments are performed:

5.2.1 Comparing the Learning Methods. To compare the efficiency of the PL* method with the non-adaptive learning method, a sample of 200 different random learning orders is produced for each subject SPL. Each learning order is considered as a permutation of the products in the sample of products. Considering each of the sampled learning orders, model learning is performed using the PL* method and the non-adaptive learning method. The following methods are used to generalize the results by catering for the following random exogenous variables in the learning process:

(1) Using random orders for the input alphabet
(2) Using random orders for the initial prefixes
(3) Using random orders for the initial suffixes

The total amount of the learning cost metrics for each learning method is calculated for each combination of random values. The total value of each metric for each learning order is calculated using the sum of the values of that metric for learning the model of all products in that order. The amount of metrics in the PL* method and the non-adaptive learning method are compared using the one-sided paired sample T-test.

5.2.2 The Effect of Learning Order. To evaluate and quantify the effect of learning order on the efficiency of the PL* method, for each subject SPL two learning orders are considered: one learning order with a high learning efficiency and one with a low learning efficiency. To obtain these learning orders, the results of the previous experiments are sorted in ascending order based on the total number of resets, the total number of input symbols and the total number of rounds, respectively. In the resulting table, the first row corresponds to the order with the highest learning efficiency and the last row corresponds to the order with the lowest learning efficiency among the orders tested. Considering these learning orders, the product models are learned using the PL* method and the total amount of metrics are calculated. This experiment is repeated 50 times for each order. The amount of metrics for these learning orders are compared using the non-paired T-test.

6 RESULTS

In this section, we first present the results of the experiments performed to evaluate the efficiency of the proposed adaptive learning method in comparison to the non-adaptive algorithm. Then, we show how different orderings of the products affect the total cost of learning in the adaptive algorithm. Finally, we discuss the obtained results and the threats to their validity. To make the diagrams clearer, the scale of each diagram is adjusted according to the values in that diagram. In this section, the average and standard deviation values of the number of resets and the number of input symbols are rounded to the nearest whole number. For the metrics shown in Tables 1 to 7, the standard deviation in the non-adaptive learning method is zero and hence, is not shown in the table. To highlight the amount of improvement made by the PL* method, "improvement percentage" is defined. For each learning cost metric, if m is its value in the PL* method and m' is its value in the non-adaptive learning method, the improvement percentage is calculated as $(1 - \frac{m}{m'}) * 100\%$. For each specific learning cost metric, if the improvement percentage is positive, it means that using the PL* method improves learning efficiency in terms of that metric. On the other hand, if the improvement percentage of some metric is negative, it shows that using the PL* method reduces learning efficiency in terms of that metric.

6.1 Comparing the learning methods (RQ1-RQ3)

In this experiment, the total amount of the learning cost metrics is calculated for 200 random orders using the PL* method and the non-adaptive learning method. In the non-adaptive learning method, the values of the learning cost metrics are exactly the same for all orders tested, obviously. Figure 5 shows the distribution of the total number of resets for the subject SPLs. The distribution of the total number of input symbols is shown in Figure 6. In these figures, the blue diagrams show the box-plots of the metrics for the PL* method. The values of metrics for the non-adaptive learning method are represented by the horizontal line on top of each box plot.

The values of the efficiency metrics in the PL* method and the non-adaptive learning method are compared using the one-sided paired sample T-test. Tables 1, 2, and 3 summarize the results of these tests for the number of rounds, the total number of resets, and the total number of input symbols, respectively. In the table

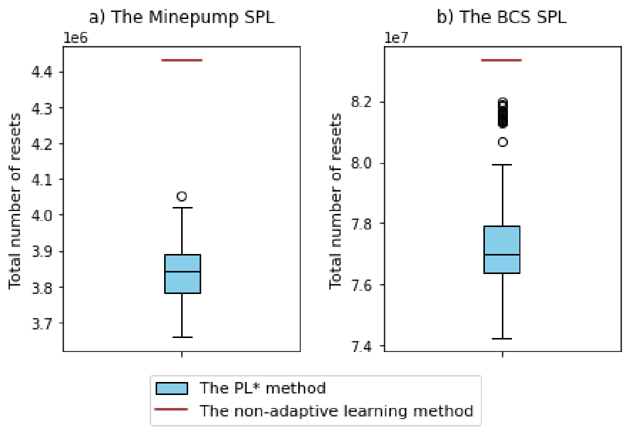

Figure 5: Distribution of the total number of resets

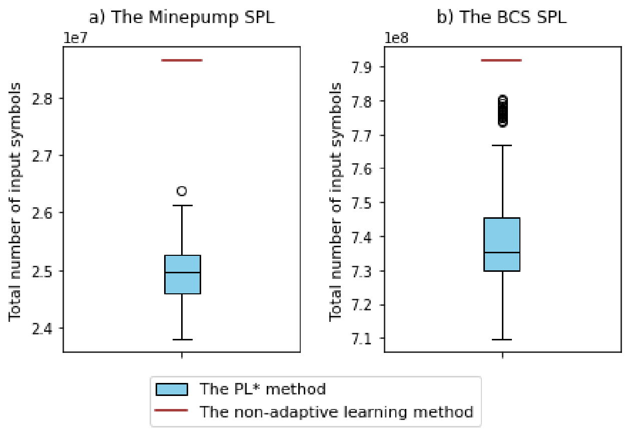

Figure 6: Distribution of the total number of input symbols

for each metric, the “Ratio” column shows the ratio of the value of that metric in the PL* method to the value of the same metric in the non-adaptive learning method; Ratio values are rounded to three decimal places.

Table 1 shows that in the Minepump SPL, the use of the PL* method reduces the number of learning rounds by about 39% compared to the non-adaptive learning method. In the BCS SPL, the number of rounds decreases by about 23%. The results of one-sided paired sample T-tests show that in the tested SPLs, the number of learning rounds in the PL* method is significantly less than that of the non-adaptive learning method (p-value < 0.01).

Another metric which is evaluated in these experiments, is the total number of resets, which is the sum of the number of reset operations of the MQs and the EQs. Table 2 shows the results of these experiments for the total number of resets. In the Minepump SPL, the total number of resets in the PL* method is approximately 13% lower than that of the non-adaptive learning method. In the BCS SPL, the amount of the reduction in the number of resets is approximately 7%. The calculated p-values show that the total number of resets in the PL* method is significantly less than the amount of this metric in the non-adaptive learning method. Therefore, in the tested SPLs, using the PL* method reduces the total number of queries in the learning process.

Another effective factor in the efficiency of the model learning algorithms is the length of queries. To estimate this parameter, the total number of input symbols can be used, which is the sum of the input symbols used in MQs and in the implementation of EQs [41]. Table 3 shows that using the PL* method reduces the total number of input symbols by about 12% in the Minepump SPL and by about 6% in the BCS SPL compared to the non-adaptive learning method. The above results show that the total number of input symbols in the PL* method is significantly less than that of the non-adaptive learning method.

Therefore, in these experiments, the use of PL* method increases the learning efficiency in terms of the number of learning rounds, the total number of resets and the total number of input symbols. The number of resets and input symbols are evaluated for MQs and EQs separately.

Tables 4 and 5 summarize the results of the experiments for the MQ resets and the MQ input symbols, respectively. In the Minepump SPL, using the PL* method increases the number of MQ resets by about 7%. In the BCS SPL, the increase in the number of MQ resets is approximately 18%. The PL* method increases the number of MQ input symbols by about 9% in the Minepump SPL and by approximately 22% in the BCS SPL.

Tables 6 and 7 show the experiment results for the EQ resets and the EQ input symbols, respectively. In the Minepump SPL, using the PL* method reduces the number of EQ resets by about 13%. In the BCS SPL, the amount of reduction in the number of EQ resets is approximately 7%. The PL* method decreases the number of EQ input symbols by about 13% in the Minepump SPL and by about 6% in the BCS SPL.

The results of the experiments show that the PL* method can improve the learning efficiency in terms of the total number of rounds, resets and input symbols. The PL* method increases the number of MQs. This adaptive learning method reduces the total cost of learning by reducing the number of EQs. Tables 4 and 6 show that in the subject SPLs, the number of EQ resets is at least one order of magnitude higher than the number of MQ resets. Similarly, Tables 5 and 7 show that the number of EQ input symbols is at least one order of magnitude higher than the number of MQ input symbols. These results indicate that the impact of EQs on the total cost of learning is much greater than the effect of MQs. Therefore, in both subject SPLs, the PL* method increases the total learning efficiency.

In the PL* method, the observation table of the product under learning is initialized using sequences from the previously learned models which are defined in its alphabet. Therefore, it makes the initial observation table more similar to the final observation table (the observation table after learning). As a result, the PL* method can decrease the number of rounds. This learning method is suitable for SPLs which are complex enough that the model learning of some of their products requires more than one round. We have not yet evaluated the effect of caching on the learning methods.

Table 1: The total number of rounds in the PL* method and the non-adaptive learning method

SUL	PL* method		Non-adaptive learning method	Improvement percentage	p-value (one-sided paired T-test)
	Average	Standard deviation	Average		
The Minepump SPL	18.005	1.167	30.000	+39.9%	2.845e-204
The BCS SPL	16.910	0.998	22.000	+23.1%	7.034e-145

Table 2: The total number of resets in the PL* method and the non-adaptive learning method

SUL	PL* method		Non-adaptive learning method	Improvement percentage	p-value (one-sided paired T-test)
	Average	Standard deviation	Average		
The Minepump SPL	3,838,078	74,075	4,429,400	+13.3%	1.095e-182
The BCS SPL	77,339,830	1,594,173	83,332,932	+7.1%	7.259e-120

Table 3: The total number of input symbols in the PL* method and the non-adaptive learning method

SUL	PL* method		Non-adaptive learning method	Improvement percentage	p-value (one-sided paired T-test)
	Average	Standard deviation	Average		
The Minepump SPL	24,950,092	465,514	28,637,112	+12.8%	5.103e-182
The BCS SPL	739,258,253	14,835,751	791,674,093	+6.6%	7.150e-115

Table 4: The number of MQ resets in the PL* method and the non-adaptive learning method

SUL	PL* method		Non-adaptive learning method	Improvement percentage
	Average	Standard deviation	Average	
The Minepump SPL	78,846	1,193	73,937	-6.7%
The BCS SPL	757,186	43,247	642,412	-17.9%

6.2 The Effect of Learning Order (RQ4)

To evaluate the effect of learning order on the PL* algorithm, from each subject SPL, two learning orders are selected: one order with a high learning efficiency (order 1) and one order with a relatively low learning efficiency (order 2), as explained in 5.2.2. Considering these orders, the model learning is performed using the PL* method. The experiment is repeated 50 times for each of these learning orders. The results of order 1 and order 2 are compared using the two-sided unpaired T-test.

In the Minepump SPL, the total number of learning rounds in all repetitions of this experiment is 15 for order 1 and 22 for order 2. However, to learn these models using the non-adaptive learning method, 30 rounds are required. As mentioned earlier, the efficiency of the non-adaptive learning method does not depend on the learning order of products. In the BCS SPL, the total number of rounds in the PL* method is 16 for order 1 and 18 for order 2, while the number of rounds in the non-adaptive learning method is 22.

Table 8 shows the results of these experiments on the total number of resets. In the Minepump SPL, the average of the total number of resets is 3714556.240 for order 1, while the value of this metric is 3898983.160 for order 2. The p-value of the two-sided unpaired T-test is 2.875e-14. In the BCS SPL, the average of the total number of resets is 76182731.480 for order 1 and 81718889.180 for order 2. In this experiment, the calculated p-value is 3.281e-42. Therefore, in these experiments, the total number of resets in order 1 is significantly different from that of order 2. Figure 7 shows the distribution of the total number of resets in the experimented learning orders.

Table 9 summarizes the results of the experiments on the total number of input symbols. In the Minepump SPL, the average of the total number of input symbols is 24166589.700 for order 1 and 25374188.260 for order 2. The p-value of the two-sided unpaired T-test is 1.421e-14. In the BCS SPL, the average of the total number of input symbols is 728779454.060 for order 1 and 778246372.540 for order 2. The calculated p-value in this experiment is 2.071e-40. Therefore, the total number of input symbols in order 1 is significantly different from the same metric in order 2. Figure 8 shows the distribution of the total number of input symbols in the experimented orders.

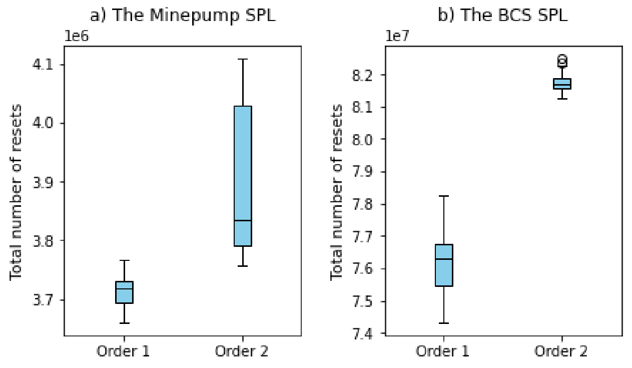

Figure 7: Distribution of the total number of resets in the experimented learning orders

The results show that in the PL* method, the order of learning the products can affect the efficiency of model learning in the SPL context.

6.2.1 How to Determine the Order of Learning. As mentioned earlier, the order of learning products can affect the efficiency of the PL* method. In this experiment, the use of parameter D to determine the product learning order in the PL* method is evaluated.

Table 5: The number of MQ input symbols in the PL* method and the non-adaptive learning method

SUL	PL* method		Non-adaptive learning method	Improvement percentage
	Average	Standard deviation	Average	
The Minepump SPL	433,967	7,513	401,613	-8.1%
The BCS SPL	5,826,720	351,470	4,804,082	-21.3%

Table 6: The number of EQ resets in the PL* method and the non-adaptive learning method

SUL	PL* method		Non-adaptive learning method	Improvement percentage
	Average	Standard deviation	Average	
The Minepump SPL	3,759,232	73,731	4,355,463	+13.6%
The BCS SPL	76,582,644	1,575,454	82,690,520	+7.3%

Table 7: The number of EQ input symbols in the PL* method and the non-adaptive learning method

SUL	PL* method		Non-adaptive learning method	Improvement percentage
	Average	Standard deviation	Average	
The Minepump SPL	24,516,124	463,540	28,235,499	+13.1%
The BCS SPL	733,431,533	14,740,136	786,870,011	+6.7%

Table 8: The effect of product learning order on the total number of resets in the PL* method

SUL	Learning order 1		Learning order 2		p-value (two-sided unpaired T-test)
	Average	Standard deviation	Average	Standard deviation	
The Minepump SPL	3,714,556	25,498	3,898,983	124,018	2.875e-14
The BCS SPL	76,182,731	978,964	81,718,889	269,506	3.281e-42

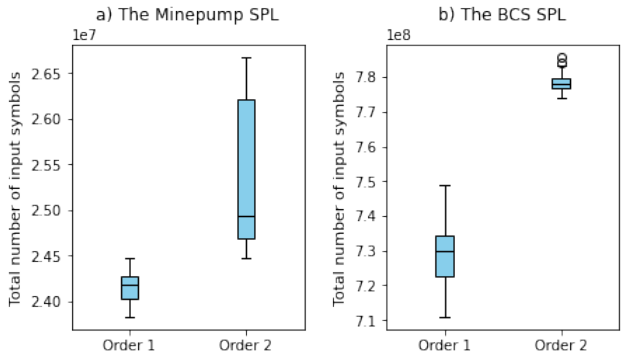

Figure 8: Distribution of the total number of input symbols in the experimented learning orders

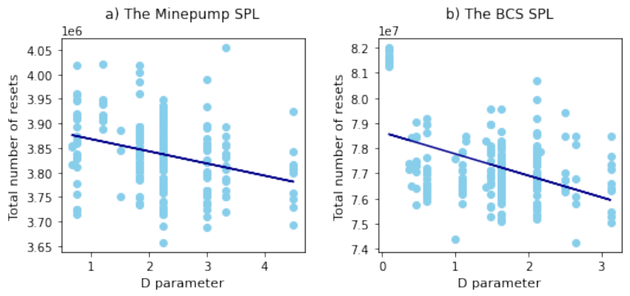

Figure 9: Diagram of the total number of resets vs. the parameter D

Using the equation 1, the parameter D is calculated for all 200 learning orders in Experiment 6.1. The Pearson correlation coefficient r between the parameter D and the learning cost metrics and its p-value is calculated.

Table 10 summarizes the results of these experiments for the total number of resets. The Pearson correlation coefficient between the parameter D and the total number of resets is -0.305 for the Minepump SPL (p-value = 1.127e-05) and -0.430 for the BCS SPL (p-value = 2.183e-10). Figure 9 shows the diagrams of the total number of resets vs. parameter D and its regression line.

The correlation coefficient between the parameter D and the total number of input symbols and its p-value is summarized in Table 11. The Pearson correlation coefficient between the parameter D and the total number of input symbols is -0.301 for the Minepump SPL (p-value = 1.484e-05) and -0.404 for the BCS SPL (p-value = 2.988e-09). The diagrams of the total number of input symbols against the parameter D and its regression line are shown in Figure 10.

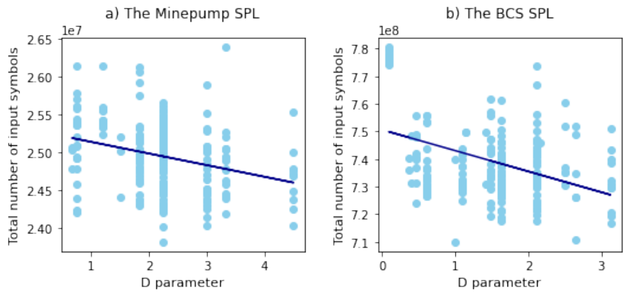

Figure 10: Diagram of the total number of input symbols vs. the parameter D

Table 9: The effect of product learning order on the total number of input symbols in the PL* method

SUL	Learning order 1		Learning order 2		p-value
	Average	Standard deviation	Average	Standard deviation	(two-sided unpaired T-test)
The Minepump SPL	24,166,590	160,544	25,374,188	796,102	1.421e-14
The BCS SPL	728,779,454	9,355,222	778,246,373	2,470,848	2.071e-40

Table 10: The Pearson correlation coefficient between the parameter D and the total number of resets

SUL	r	p-value
The Minepump SPL	-0.305	1.127e-05
The BCS SPL	-0.430	2.183e-10

Table 11: The Pearson correlation coefficient between the parameter D and the total number of input symbols

SUL	r	p-value
The Minepump SPL	-0.301	1.484e-05
The BCS SPL	-0.404	2.988e-09

The above experiments show that the order of learning the products can affect the efficiency of the PL* method. In these experiments, it is observed that the total learning efficiency usually increases if the number of new non-mandatory features that are added simultaneously is small. Using this observation, the parameter D is defined as a heuristic to find an order which increases the efficiency of learning. Experimental results show that there is a mild negative correlation between the value of D and the total number of resets. There is also a mild negative correlation between the value of D and the total number of input symbols. Therefore, it is possible to determine the proper order for learning a subset of products using the PL* method. However, for all learning orders tested, the PL* method is more efficient than the non-adaptive learning method.

6.3 Threats to Validity

Because the PL* method has been tested on a small number of case studies, the results may be biased according to the characteristics of the subject systems. This is a threat to generalization of our result. To mitigate this threat, we plan to test the PL* method on more subject systems. Evaluating the PL* method in case studies featuring the evolution of behavior both in space and time is a way to test more and larger subject systems (considering evolution in time will enrich our set of case studies and make our method applicable to a larger problem space). We also see a threat to the generalization of the product ordering heuristic due to our limited set of subject systems; with a large set of case studies, we see a possibility of (statistically) learning the optimal order as well.

The use of a particular T-wise sampling algorithm (with T = 3, based on [12]) may pose another threat for the generalization of our results. We plan to extend our results by considering other values for T and more advanced sampling algorithms [44].

We captured the random exogenous variables involved in our experiments, such as the order of alphabet symbols and prefixes in learning, and minimized their threats to the validity of the results by taking a large sample. This threat is sufficiently mitigated for the current experiment and we did not observe any significant influence of these random variables in our results to plan a further mitigation.

To minimize the threats to conclusion validity, we opted for the most general statistical tests in our experiment design: for comparing the learning methods, we used one-sided paired sample T-tests. For comparing and evaluating the learning orders, we started off with an unpaired sample T-test but strengthened the results by using the Pearson correlation coefficient for measuring the correlation of learning efficiency with the value of the parameter D.

7 CONCLUSION

In this paper, we presented an adaptive model-learning approach that reuses the learned information about the behavior of products while covering the variability space. It has been shown through an empirical evaluation on two subject systems that our proposed adaptive approach significantly outperforms the standard model learning approach based on Angluin's L* algorithm. For our comparison, we have used the number of resets and the total number of input symbols, as well as the number of equivalence and membership queries. Additionally, we studied the role of product ordering in learning efficiency and provide a heuristic through defining a parameter that was shown to correlate with learning efficiency for our subject systems.

Performing more experimental evaluation with other subject systems is among our priorities for future work. We plan to extend our technique to a family-based learning process by extending the learning data structures to ones annotated with feature expressions. Other model learning techniques have been proposed recently, which can be extended to the adaptive and family-based setting following the same recipe. It was observed that the randomness in the order of prefixes and suffixes affects the efficiency of the PL* method, while it does not affect the efficiency of the non-adaptive learning method. Evaluating the effect of randomness of the order of prefixes and suffixes on the efficiency of the PL* method is another line of our future research works.

Acknowledgements. The work of Mohammad Reza Mousavi was partially supported by the UKRI Trustworthy Autonomous Systems Node in Verifiability, Grant Award Reference EP/V026801/2.

REFERENCES

[1] Ra'Fat Al-Msie'deen, Marianne Huchard, Abdelhak Seriai, Christelle Urtado, and Sylvain Vauttier. 2014. Reverse Engineering Feature Models from Software Configurations using Formal Concept Analysis. In *Proceedings of the Eleventh International Conference on Concept Lattices and Their Applications, Košice, Slovakia, October 7-10, 2014 (CEUR Workshop Proceedings, Vol. 1252)*, Karell Bertet and Sebastian Rudolph (Eds.). CEUR-WS.org, 95–106. http://ceur-ws.org/Vol-1252/cla2014_submission_13.pdf

[2] Dana Angluin. 1987. Learning Regular Sets from Queries and Counterexamples. *Inf. Comput.* 75, 2 (1987), 87–106. https://doi.org/10.1016/0890-5401(87)90052-6

[3] Sven Apel, Don Batory, Christian Kästner, and Gunter Saake. 2013. A Development Process for Feature-Oriented Product Lines. In *Feature-Oriented Software*

Product Lines: Concepts and Implementation. Springer, Berlin, Heidelberg, 17–44. https://doi.org/10.1007/978-3-642-37521-7_2

[4] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. 2010. Automated analysis of feature models 20 years later: A literature review. *Information Systems* 35, 6 (2010), 615–636. https://doi.org/10.1016/j.is.2010.01.001

[5] Sagar Chaki, Edmund Clarke, Natasha Sharygina, and Nishant Sinha. 2008. Verification of evolving software via component substitutability analysis. *Formal Methods in System Design* 32, 3 (01 Jun 2008), 235–266.

[6] Vasek Chvátal. 1979. A Greedy Heuristic for the Set-Covering Problem. *Math. Oper. Res.* 4, 3 (1979), 233–235. https://doi.org/10.1287/moor.4.3.233

[7] Andreas Classen. 2010. Modelling with FTS: a Collection of Illustrative Examples, Technical Report, University of Namur. (2010).

[8] Andreas Classen et al. 2011. Modelling and model checking variability-intensive systems. *Ph. D. dissertation* (2011).

[9] Paul Clements and Linda M. Northrop. 2002. *Software product lines - practices and patterns.* Addison-Wesley.

[10] Carlos Diego Nascimento Damasceno, Mohammad Reza Mousavi, and Adenilso da Silva Simão. 2019. Learning from difference: an automated approach for learning family models from software product lines. In *Proceedings of the 23rd International Systems and Software Product Line Conference, SPLC 2019, Volume A, Paris, France, September 9-13, 2019,* Thorsten Berger, Philippe Collet, Laurence Duchien, Thomas Fogdal, Patrick Heymans, Timo Kehrer, Jabier Martinez, Raúl Mazo, Leticia Montalvillo, Camille Salinesi, Xhevahire Tërnava, Thomas Thüm, and Tewfik Ziadi (Eds.). ACM, 10:1–10:12. https://doi.org/10.1145/3336294.3336307

[11] Carlos Diego Nascimento Damasceno, Mohammad Reza Mousavi, and Adenilso da Silva Simão. 2019. Learning to Reuse: Adaptive Model Learning for Evolving Systems. In *Integrated Formal Methods - 15th International Conference, IFM 2019, Bergen, Norway, December 2-6, 2019, Proceedings (Lecture Notes in Computer Science, Vol. 11918)*, Wolfgang Ahrendt and Silvia Lizeth Tapia Tarifa (Eds.). Springer, 138–156. https://doi.org/10.1007/978-3-030-34968-4_8

[12] Carlos Diego Nascimento Damasceno, Mohammad Reza Mousavi, and Adenilso da Silva Simão. 2021. Learning by sampling: learning behavioral family models from software product lines. *Empir. Softw. Eng.* 26, 1 (2021), 4. https://doi.org/10.1007/s10664-020-09912-w

[13] Sophie Fortz. 2021. *LIFTS: Learning Featured Transition Systems.* Association for Computing Machinery, New York, NY, USA, 1–6. https://doi.org/10.1145/3461002.3473066

[14] Vanderson H. Fragal, Adenilso Simão, and Mohammad Reza Mousavi. 2016. Validated Test Models for Software Product Lines: Featured Finite State Machines. In *Formal Aspects of Component Software - 13th International Conference, FACS 2016, Besançon, France, October 19-21, 2016, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 10231)*, Olga Kouchnarenko and Ramtin Khosravi (Eds.). 210–227. https://doi.org/10.1007/978-3-319-57666-4_13

[15] Matthias Galster, Danny Weyns, Dan Tofan, Bartosz Michalik, and Paris Avgeriou. 2014. Variability in Software Systems - A Systematic Literature Review. *IEEE Trans. Software Eng.* 40, 3 (2014), 282–306. https://doi.org/10.1109/TSE.2013.56

[16] Arthur Gill et al. 1962. Introduction to the theory of finite-state machines. (1962).

[17] Alex Groce, Doron Peled, and Mihalis Yannakakis. 2002. Adaptive Model Checking. In *Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '02)*. Springer-Verlag, Berlin, Heidelberg, 357–370.

[18] Roland Groz, Adenilso da Silva Simão, Alexandre Petrenko, and Catherine Oriat. 2015. Inferring Finite State Machines Without Reset Using State Identification Sequences. In *Testing Software and Systems - 27th IFIP WG 6.1 International Conference, ICTSS 2015, Sharjah and Dubai, United Arab Emirates, November 23-25, 2015, Proceedings (Lecture Notes in Computer Science, Vol. 9447)*, Khaled El-Fakih, Gerassimos D. Barlas, and Nina Yevtushenko (Eds.). Springer, 161–177. https://doi.org/10.1007/978-3-319-25945-1_10

[19] Roland Groz, Adenilso da Silva Simão, Alexandre Petrenko, and Catherine Oriat. 2018. Inferring FSM Models of Systems Without Reset. In *Machine Learning for Dynamic Software Analysis: Potentials and Limits - International Dagstuhl Seminar 16172, Dagstuhl Castle, Germany, April 24-27, 2016, Revised Papers (Lecture Notes in Computer Science, Vol. 11026)*, Amel Bennaceur, Reiner Hähnle, and Karl Meinke (Eds.). Springer, 178–201. https://doi.org/10.1007/978-3-319-96562-8_7

[20] Evelyn Nicole Haslinger, Roberto E. Lopez-Herrejon, and Alexander Egyed. 2011. Reverse Engineering Feature Models from Programs' Feature Sets. In *2011 18th Working Conference on Reverse Engineering.* IEEE, 308–312. https://doi.org/10.1109/WCRE.2011.45

[21] David Huistra, Jeroen Meijer, and Jaco van de Pol. 2018. Adaptive Learning for Learn-Based Regression Testing. In *Formal Methods for Industrial Critical Systems (Lecture Notes in Computer Science)*, Falk Howar and Jiri Barnat (Eds.). Springer, Switzerland, 162–177.

[22] J. D. Hunter. 2007. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering* 9, 3 (2007), 90–95. https://doi.org/10.1109/MCSE.2007.55

[23] Muhammad-Naeem Irfan, Catherine Oriat, and Roland Groz. 2013. Model Inference and Testing. *Adv. Comput.* 89 (2013), 89–139. https://doi.org/10.1016/B978-0-12-408094-2.00003-5

[24] Martin Fagereng Johansen, Øystein Haugen, and Franck Fleurey. 2011. Properties of Realistic Feature Models Make Combinatorial Testing of Product Lines Feasible. In *Model Driven Engineering Languages and Systems, 14th International Conference, MODELS 2011, Wellington, New Zealand, October 16-21, 2011. Proceedings (Lecture Notes in Computer Science, Vol. 6981)*, Jon Whittle, Tony Clark, and Thomas Kühne (Eds.). Springer, 638–652. https://doi.org/10.1007/978-3-642-24485-8_47

[25] Eric Jones, Travis Oliphant, Pearu Peterson, et al. 2001–. SciPy: Open source scientific tools for Python. http://www.scipy.org/

[26] Kyo C Kang, Sholom G Cohen, James A Hess, William E Novak, and A Spencer Peterson. 1990. *Feature-oriented domain analysis (FODA) feasibility study.* Technical Report. Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst.

[27] Daniel Le Berre and Anne Parrain. 2010. The SAT4J library, Release 2.2, System Description. *Journal on Satisfiability, Boolean Modeling and Computation* 7 (2010), 59–64. http://satassociation.org/jsat/index.php/jsat/article/view/82

[28] Luc Lesoil, Mathieu Acher, Arnaud Blouin, and Jean-Marc Jézéquel. 2021. Deep Software Variability: Towards Handling Cross-Layer Configuration. In *15th International Working Conference on Variability Modelling of Software-Intensive Systems* (Krems, Austria) *(VaMoS'21)*. Association for Computing Machinery, New York, NY, USA, Article 10, 8 pages. https://doi.org/10.1145/3442391.3442402

[29] Sascha Lity, Remo Lachmann, Malte Lochau, and Ina Schaefer. 2012. *Delta-oriented software product line test models-the body comfort system case study.* Technical Report 2012-07. TU Braunschweig.

[30] Sebastian Oster. 2012. *Feature Model-based Software Product Line Testing.* Ph.D. Dissertation. Technische Universität. [Online] http://tuprints.ulb.tu-darmstadt.de/2881/.

[31] Juliana Alves Pereira, Mathieu Acher, Hugo Martin, Jean-Marc Jézéquel, Goetz Botterweck, and Anthony Ventresque. 2021. Learning software configuration spaces: A systematic literature review. *Journal of Systems and Software* 182 (2021), 111044. https://doi.org/10.1016/j.jss.2021.111044

[32] Gilles Perrouin, Sagar Sen, Jacques Klein, Benoit Baudry, and Yves Le Traon. 2010. Automated and Scalable T-wise Test Case Generation Strategies for Software Product Lines. In *Third International Conference on Software Testing, Verification and Validation, ICST 2010, Paris, France, April 7-9, 2010.* IEEE Computer Society, 459–468. https://doi.org/10.1109/ICST.2010.43

[33] Harald Raffelt and Bernhard Steffen. 2006. LearnLib: A Library for Automata Learning and Experimentation. In *Fundamental Approaches to Software Engineering, 9th International Conference, FASE 2006, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2006, Vienna, Austria, March 27-28, 2006, Proceedings (Lecture Notes in Computer Science, Vol. 3922)*, Luciano Baresi and Reiko Heckel (Eds.). Springer, 377–380. https://doi.org/10.1007/11693017_28

[34] Uwe Ryssel, Joern Ploennigs, and Klaus Kabitzsch. 2011. Extraction of Feature Models from Formal Contexts. In *Proceedings of the 15th International Software Product Line Conference, Volume 2* (Munich, Germany) *(SPLC '11)*. ACM, New York, NY, USA, 1–8. https://doi.org/10.1145/2019136.2019141

[35] Pierre-Yves Schobbens, Patrick Heymans, and Jean-Christophe Trigaux. 2006. Feature Diagrams: A Survey and a Formal Semantics. In *14th IEEE International Conference on Requirements Engineering (RE 2006), 11-15 September 2006, Minneapolis/St.Paul, Minnesota, USA.* IEEE Computer Society, 136–145. https://doi.org/10.1109/RE.2006.23

[36] Muzammil Shahbaz and Roland Groz. 2009. Inferring Mealy Machines. In *FM 2009: Formal Methods, Second World Congress, Eindhoven, The Netherlands, November 2-6, 2009. Proceedings (Lecture Notes in Computer Science, Vol. 5850)*, Ana Cavalcanti and Dennis Dams (Eds.). Springer, 207–222. https://doi.org/10.1007/978-3-642-05089-3_14

[37] Joshua Sprey, Chico Sundermann, Sebastian Krieter, Michael Nieke, Jacopo Mauro, Thomas Thüm, and Ina Schaefer. 2020. SMT-based variability analyses in FeatureIDE. In *Proceedings of the 14th International Working Conference on Variability Modelling of Software-Intensive Systems (VAMOS '20)*. Association for Computing Machinery, New York, NY, USA, 1–9. https://doi.org/10.1145/3377024.3377036

[38] Paul Temple, Gilles Perrouin, Mathieu Acher, Battista Biggio, Jean-Marc Jézéquel, and Fabio Roli. 2021. Empirical assessment of generating adversarial configurations for software product lines. *Empir. Softw. Eng.* 26, 1 (2021), 6. https://doi.org/10.1007/s10664-020-09915-7

[39] Thomas Thüm, Sven Apel, Christian Kästner, Ina Schaefer, and Gunter Saake. 2014. A Classification and Survey of Analysis Strategies for Software Product Lines. *ACM Comput. Surv.* 47, 1 (2014), 6:1–6:45. https://doi.org/10.1145/2580950

[40] Thomas Thüm, Christian Kästner, Fabian Benduhn, Jens Meinicke, Gunter Saake, and Thomas Leich. 2014. FeatureIDE: An extensible framework for feature-oriented software development. *Sci. Comput. Program.* 79 (2014), 70–85. https://doi.org/10.1016/j.scico.2012.06.002

[41] Frits W. Vaandrager. 2017. Model learning. *Commun. ACM* 60, 2 (2017), 86–95. https://doi.org/10.1145/2967606

[42] Frank van der Linden, Klaus Schmid, and Eelco Rommes. 2007. *Software product lines in action - the best industrial practice in product line engineering.* Springer.

https://doi.org/10.1007/978-3-540-71437-8

[43] Jilles van Gurp, Jan Bosch, and Mikael Svahnberg. 2001. On the Notion of Variability in Software Product Lines. In *2001 Working IEEE / IFIP Conference on Software Architecture (WICSA 2001), 28-31 August 2001, Amsterdam, The Netherlands*. IEEE Computer Society, 45–54. https://doi.org/10.1109/WICSA.2001.948406

[44] Mahsa Varshosaz, Mustafa Al-Hajjaji, Thomas Thüm, Tobias Runge, Mohammad Reza Mousavi, and Ina Schaefer. 2018. A classification of product sampling for software product lines. In *Proceeedings of the 22nd International Systems and Software Product Line Conference - Volume 1, SPLC 2018, Gothenburg, Sweden, September 10-14, 2018*, Thorsten Berger, Paulo Borba, Goetz Botterweck, Tomi Männistö, David Benavides, Sarah Nadi, Timo Kehrer, Rick Rabiser, Christoph Elsner, and Mukelabai Mukelabai (Eds.). ACM, 1–13. https://doi.org/10.1145/3233027.3233035

[45] Stephan Windmüller, Johannes Neubauer, Bernhard Steffen, Falk Howar, and Oliver Bauer. 2013. Active Continuous Quality Control. In *Proceedings of the 16th International ACM Sigsoft Symposium on Component-based Software Engineering* (Vancouver, British Columbia, Canada) *(CBSE '13)*. ACM, New York, NY, USA, 111–120.

[46] Nan Yang, K. Aslam, R.R.H. Schiffelers, Leonard Lensink, D. Hendriks, L.G.W.A. Cleophas, and A. Serebrenik. 2019. Improving model inference in industry by combining active and passive learning. In *26th IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER 2019)*. Institute of Electrical and Electronics Engineers (IEEE), United States, 253–263.