UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

Learning finite state machine models of evolving systems: From evolution *over time* to variability *in space*

Carlos Diego Nascimento Damasceno

Tese de Doutorado do Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional (PPG-CCMC)

ICMC USP SÃO CARLOS

Carlos Diego Nascimento Damasceno

Learning finite state machine models of evolving systems: From evolution *over time* to variability *in space*

Thesis submitted to the Institute of Mathematics and Computer Sciences – ICMC-USP – in accordance with the requirements of the Computer and Mathematical Sciences Graduate Program, for the degree of Doctor in Science. *FINAL VERSION*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dr. Adenilso da Silva Simão
Co-advisor: Prof. Dr. Mohammad Reza Mousavi

USP – São Carlos
August 2020

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados inseridos pelo(a) autor(a)

D155l

Damasceno, Carlos Diego Nascimento
Learning finite state machine models of evolving systems: From evolution over time to variability in space / Carlos Diego Nascimento Damasceno; orientador Adenilso da Silva Simão; coorientador Mohammad Reza Mousavi. -- São Carlos, 2020.
128 p.

Tese (Doutorado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2020.

1. Aprendizado de autômatos. 2. Evolução de software. 3. Sistemas reativo. 4. Varibilidade de software. I. Simão, Adenilso da Silva, orient. II. Mousavi, Mohammad Reza, coorient. III. Título.

Bibliotecários responsáveis pela estrutura de catalogação da publicação de acordo com a AACR2:
Gláucia Maria Saia Cristianini - CRB - 8/4938
Juliana de Souza Moraes - CRB - 8/6176

Carlos Diego Nascimento Damasceno

Aprendizado de modelos de máquinas de estados finitos de sistemas em evolução: Da evolução ao longo do tempo para variabilidade em espaço

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Doutor em Ciências – Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. Adenilso da Silva Simão
Coorientador: Prof. Dr. Mohammad Reza Mousavi

USP – São Carlos
Agosto de 2020

Aos meus pais, Eloi e Lucia, que ensinaram-me a ter apreço pelas pequenas coisas que vida nos traz.

To my parents, Eloi and Lucia, who taught me to appreciate every tiny little thing that life brings us.

ACKNOWLEDGEMENTS

- Aos meus pais, Eloi e Lucia, que não mediram esforços para me fornecer uma base sólida de educação e princípios éticos que me permitiram concretizar o sonho de me tornar *cientista*. Obrigado pelas aulas de matemática (*cinco vezes cinco!?!?*) e pela paciência em lidar com minhas peripécias e brinquedos constantemente desmontados. Amo vocês!

- Aos meus familiares, que torceram por mim e me ajudaram direta ou indiretamente durante esta árdua caminhada.

- Ao meu orientador Adenilso Simão, por ter me "adotado" durante esses 6 anos de pós-graduação na USP, compartilhando suas experiência como docente e pesquisador, por sempre estar disponível para ouvir minhas ideias milaborantes e confiar no meu trabalho.

- To my co-supervisor Mohammad Reza Mousavi, for receiving me at the University of Leicester and for being such a great tutor and research collaborator.

- Aos funcionários da Universidade de São Paulo e de Leicester com quem tive contato e aos terceirizados que cuidaram da limpeza dos laboratórios onde atuei, também deixo meus agradecimentos pela sua dedicação.

- Aos professorxs e amigxs do CEFER USP São Carlos que fortaleceram o meu físico e psicológico ao longo desses anos com muitas práticas de corrida. Gizele, Cris, Evert, Deividi, Adilson Wanderley, Javier, Clodoaldo, Raul Cassaro, Raphael Montanari, Lais Americo e Laura Botero vocês são incríveis!

- Aos professores da Universidade Federal do Pará (UFPa), em especial Fábio Lobato, Àdamo Santana e Cleidson de Souza, pelo seu importante papel e influência durante o início da minha formação como cientista da computação e pesquisador.

- À todos os meus amigos e parceiros de laboratório da USP, em especial, Brauner Oliveira, Daniel Soares, Diogenes Souza, Jorge Cutigi, Laiza Silva, Leonildo Azevedo, Lina Garces, Rachel Reis, Stevão Andrade, Tiago Volpato, Valdemar Neto, que colaboraram para fazer esta árdua caminhada mais divertida e cheia de aprendizados.

- To all the new and old friends I met in England and made it an amazing experience, in particular, Cristina Ruiz, Darci Martins, Hélène Martins, Hugo Araujo, Jalal Miftah, Jan Ringert, José Rojas, Rayna Dimitrova, Nola O'Donell, Sally O'Donell, and Stefan O'Donell.

- Aos meus amigos de graduação, Lucas Melo, Felipe Leite, Teófilo Augusto e Eduardo Carvalho, pelas suas contribuições nas iterações iniciais da minha apresentação final.
- À Kamila Takayama Lyra, minha parceira de jornada, pelo carinho e por me ensinar a ser uma pessoa melhor a cada dia. Amo você!
- E ao CNPq, CAPES e FAPESP, pelo apoio financeiro.

“Nessa estrada não nos cabe
Conhecer ou ver o que virá
O fim dela ninguém sabe
Bem ao certo onde vai dar

Vamos todos
Numa linda passarela
De uma aquarela que um dia enfim
Descolorirá.”
Toquinho, Aquarela (1983)

RESUMO

DAMASCENO, C. D. N. **Aprendizado de modelos de máquinas de estados finitos de sistemas em evolução: Da evolução ao longo do tempo para variabilidade em espaço**. 2020. 128 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2020.

Manutenção e evolução são principios básicos do ciclo de vida de software. Apesar disso, artefatos de modelagem frequentemente tendem a ser negligenciados. Consequentemente, modelos podem ficar desatualizados e dificultar a adoção de algumas técnicas tais como verificação e teste baseado em modelos. Estudos recentes têm mostrado que técnicas de aprendizado de modelos de máquinas de estados finitos têm se tornado bastante populares no teste e verificação de software. Apesar disso, algoritmos para aprendizado de modelos ainda sofrem com problemas de escalabilidade assim como com a evolução *ao longo do tempo* que pode requerer o *re-aprendizado do zero*. Adicionalmente, há uma lacuna de pesquisas sobre estratégias de aprendizado de modelos para linhas de produto de software, i.e., sistemas onde variantes de software co-existem e, consequentemente, incorporam variabilidade *no espaço*. Esta Tese de Doutorado avança no estado da arte da engenharia de software baseada em modelos apresentando contribuições teóricas e práticas sobre aprendizado de modelos para sistemas que incorporam evolução ao longo do tempo e variabilidade no espaço. As três principais contribuições desta Tese de Doutorado são: (i) um algoritmo adaptativo de aprendizado de modelos que explora versões de software pré-existentes *on-the-fly* para descartar conhecimento redundante e descontinuado representados em termos de sequências de entradas que não levem à descoberta de estados. Usando máquinas de estados reais do projeto OpenSSL, mostra-se que o algoritmo proposto consegue ser mais eficiente que o estado da arte e menos sensível à evolução de software. (ii) Preenche-se a lacuna de pesquisas em algoritmos de aprendizado de modelos para linhas de produto com o algoritmo $FFSM_{Diff}$, uma técnica automatizada para identificar comportamentos similares e anotar estados e transições de máquinas de estados finitos com restrições de características (FFSM, sigla do inglês). Usando 105 modelos derivados de seis linhas de produto acadêmicas, mostra-se que o algoritmo proposto consegue combinar famílias de máquinas de estados em FFSMs significativamente sucintas, especialmente quando há um alto reúso entre os produtos analisados. (iii) Um conjunto de experiências que incorporam amostragem de produtos no $FFSM_{Diff}$. Os resultados indicam que modelos de FFSM construídos usando amostragem podem ser tão precisos quanto aqueles feitos usando aprendizado exaustivo e, consequentemente, cobrem o comportamento de uma linha de produto.

Palavras-chave: Aprendizado de autômatos, Evolução de software, Sistemas reativos, Varibilidade de software.

ABSTRACT

DAMASCENO, C. D. N. **Learning finite state machine models of evolving systems: From evolution *over time* to variability *in space***. 2020. 128 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2020.

Maintenance and evolution have been accepted as integral principles in the software development life-cycle. They are essential for any system that operates in or addresses problems or activities of the real world if it is to remain useful and profitable. Nevertheless, as time passes and modifications occur, modeling artifacts are often neglected due to the lack of proper maintenance. Hence, it may render outdated models and hinder the application of model-based reasoning techniques, such as model-based testing and model checking. To address these issues, recent academic and industrial studies have shown that finite state machine (FSM) model learning techniques are becoming increasingly popular in software verification and testing. Despite these advances, model learning algorithms are still hampered by scalability issues, as well as the constant changes *over time* that may require *learning from scratch*. Furthermore, there is a lack of investigations about learning strategies for software product lines (SPL), i.e., systems where variants shall co-exist to satisfying the needs of distinct market segments and, hence, incorporate variability *in space*. In this PhD Thesis, we improve upon the state-of-the-art of model-based software engineering by introducing theoretical and experimental contributions to address model learning in the setting of evolving systems that incorporate modifications *over time* and variability *in space*. Our main contributions are three-fold: (i) We have introduced the `partial-Dynamic` $\mathtt{L}^*_{\mathtt{M}}$, an adaptive algorithm that explores models from pre-existing versions *on-the-fly* to discard *redundant* and *deprecated* knowledge in terms of input sequences that may not lead to state discovery. Using realistic models of the OpenSSL toolkit, we have shown that our algorithm has been more efficient than state-of-the-art techniques and less sensitive to software evolution. (ii) We have filled the gap of model learning algorithms for variability-intensive systems by introducing the $FFSM_{Diff}$ algorithm. It is an automated technique to identify similar behavior shared among product-specific FSMs, annotate states, and transitions with feature constraints, and integrate them into succinct featured finite state machines (FFSM). Using 105 FSMs derived from six SPLs of academic benchmarks, we have shown that our algorithm can effectively merge families of state machines into succinct FFSMs, especially if there is high feature reuse among products. (iii) We have extended our expertise upon the $FFSM_{Diff}$ algorithm and reported our experiences on learning FFSMs through product sampling. Our results have indicated that FFSMs learned by sampling can be as precise as those learned from exhaustive analysis and hence, collectively cover the behavior of an SPL.

Keywords: Automata learning, Software evolution, Reactive systems, Software Variability.

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF TABLES

CONTENTS

CHAPTER

1

INTRODUCTION

In early 1968, the North Atlantic Treaty Organization (NATO) established a study group to shed further light on many problems concerning software manufacturing (NATO, 1968). At this meeting, the phrase "software engineering" was coined as a provocative term to the need for software manufacturing to be based on theoretical foundations and practical principles that were traditional in other established branches of engineering. Among these principles, software maintenance and evolution have been accepted as fundamental principles of software life-cycle.

According to the ISO/IEC 14764, software maintenance is defined as the modification of a software product *after delivery* to correct faults, to improve non-functional attributes, or to adapt the product to a modified environment (IEEE, 2006). Software evolution has been addressed as a complementary idea that programs must be modified because they operate in the real world or address problems or activities from it (LEHMAN, 1979). Hence, changes in the real world shall affect software that will also require adaptations (GODFREY; GERMAN, 2014).

As software evolution takes place, many circumstances to compromise software quality and reliability may emerge (DEUTSCH, 1981). To mitigate these risks, modeling notations (BOOCH; RUMBAUGH; JACOBSON, 2005) have been used to explicitly describe static and dynamic aspects of software systems and support software analysis (IEEE, 2010).

According to Binder (1999), software analysis is necessarily a *model-based* activity, whether implicit in engineers' minds, informally sketched on papers, or formally denoted as explicit models (MARINESCU *et al.*, 2015). In software analysis, explicit models are created to scrutinize parts of the software (e.g., functions, components, or objects) and determine how they behave and relate to each other (IEEE, 2010). The unified modeling language (UML) has been the *de facto* tool to visualize, specify, construct, document, and test systems as well as for modeling business and similar processes (BOOCH; RUMBAUGH; JACOBSON, 2005).

Software models are key assets for the development of safety-critical systems where malfunctioning can result in death, injury, or damage to the environment (GURBUZ; TEKINERDO-

GAN, 2018). They support the analysis of requirements consistency (UTTING; PRETSCHNER; LEGEARD, 2012), improve program comprehension (SAID; QUANTE; KOSCHKE, 2019), reduce the costs for testing (DIAS-NETO; TRAVASSOS, 2010), increase fault detection rate (HEMMATI; ARCURI; BRIAND, 2013) and pave the way for automated software testing techniques (UTTING; PRETSCHNER; LEGEARD, 2012).

Software testing is a dynamic analysis technique that aims at verifying the behavior of a system under test (SUT) against its expected behavior using a finite set of test cases composed of inputs and expected outputs (IEEE, 2012). Traditionally, software testing is ad hoc and relies heavily on the engineers' expertise (MARINESCU *et al.*, 2015). Test engineers often use incomplete and informal requirements to gain insights about the intended software behavior and manually derive and execute test cases (PRETSCHNER; PHILIPPS, 2005).

By contrast, model-based testing (MBT) is a model-centric variant that relies on explicit models encoding the intended behavior or environment of a SUT to automate test case generation, selection, and execution (UTTING; PRETSCHNER; LEGEARD, 2012). Finite state machine-based testing is a specialization of MBT that uses finite state machines (FSM), e.g., Mealy machines (GILL, 1962), to derive test cases (BROY *et al.*, 2005). These test cases are composed of input-output pairs fed into the SUT to prove its conformance against an FSM specification (BERG; RAFFELT, 2005).

While MBT strategies have been studied for several decades (VASILEVSKII, 1973; CHOW, 1978) and introduced advances to the state of the art and state of the practice (MLYNARSKI *et al.*, 2012; SHAFIQUE; LABICHE, 2015; MARINESCU *et al.*, 2015), building useful test models is still time-consuming, tedious, error-prone and dependent on engineers' expertise (PRETSCHNER; PHILIPPS, 2005). Additionally, in the lack of proper maintenance (WALKINSHAW, 2013), test models often become outdated. Hence, they may hinder the application of MBT (MARIANI; PEZZÈ; ZUDDAS, 2015). To mitigate these problems, recent academic researches and industrial case studies (IRFAN; ORIAT; GROZ, 2013; MARIANI; PEZZÈ; ZUDDAS, 2015; VAANDRAGER, 2017; AICHERNIG *et al.*, 2018) have shown that black-box approaches for learning FSMs (ANGLUIN, 1987; SHAHBAZ; GROZ, 2009; MEINKE; SINDHU, 2011a; CASSEL FALK HOWAR, 2015) are becoming increasingly popular in software analysis and testing.

Coined by Angluin (1987), model learning has been introduced as an *active* procedure to formulate a hypothesis $\mathscr{H}$ about the "language" (i.e., behavior) of a system under learning (SUL). It can be thought as the inverse process of MBT where test cases are pursued to derive a model that fits the behavior of the SUL (WEYUKER, 1983) rather than to describe its expected and actual behaviors (BROY *et al.*, 2005). Model learning is often described in terms of the Minimally Adequate Teacher (MAT) framework (ANGLUIN, 1987). In the MAT framework, we assume the input/output finite vocabulary of the SUL is known and that it produces outputs for any input within a known and finite amount of time. The MAT framework is composed by two

iterative phases: hypothesis construction and hypothesis validation.

In the (i) *hypothesis construction*, a learning algorithm poses Membership Queries (`MQ`) to gain knowledge about the SUL using `reset` operations and input sequences. The `MQ`s and query outputs are organized in a data structure known as the *observation table*; it is composed of sets of *transfer* and *separating* sequences for reaching and distinguishing states of the SUL, respectively.

In the (ii) *hypothesis validation*, a hypothesis $\mathscr{H}$ about the "language" of the SUL is formulated using the `MQ`s posed so far and tested using Equivalence Queries (`EQ`). The `EQ`s are often concretized in terms of MBT, where it returns `yes` if the hypothesis is correct (i.e., `all pass`); otherwise, it finds a `counterexample` exposing a non-conformance (i.e., `failed test`) to refine the hypothesis.

1.1 Motivation

Companies such as Siemens (HAGERER *et al.*, 2002), Springer (NEUBAUER *et al.*, 2012), Volvo (FENG *et al.*, 2013), Orange (SHAHBAZ; GROZ, 2014), Philips (SCHUTS; HOOMAN; VAANDRAGER, 2016) and Océ (SMEENK *et al.*, 2015) have been using model learning to address software analysis and testing problems. Model learning has been harnessed for black-box model checking (PELED; VARDI; YANNAKAKIS, 1999), detecting feature interactions (SHAHBAZ; PARREAUX; KLAY, 2007), analyzing network protocols (AARTS *et al.*, 2012; FITERĂU-BROŞTEAN; HOWAR, 2017), and characterizing software evolution (HUNGAR; NIESE; STEFFEN, 2003; DE RUITER; POLL, 2015). Furthermore, it has been used for testing web services (BAINCZYK *et al.*, 2016), learning models from non-resettable systems (GROZ *et al.*, 2015), supporting automated test generation (RAFFELT *et al.*, 2009) and deriving failure models (CHAPMAN *et al.*, 2015; KUNZE *et al.*, 2016).

Although model learning has been studied since 1987 and significant progress has been achieved on a wide range of problems (IRFAN; ORIAT; GROZ, 2013; MARIANI; PEZZÈ; ZUDDAS, 2015; VAANDRAGER, 2017; AICHERNIG *et al.*, 2018), the application of model learning to industrial systems is still hampered by scalability issues (DUHAIBY *et al.*, 2018). Software changes that are exacerbated by agile methodologies, where requirements and implementations are continuously evolving (MEINKE; WALKINSHAW, 2012), may often lead to the need for *learning from scratch*.

Adaptive model learning (GROCE; PELED; YANNAKAKIS, 2002) is a variant that attempts to speed up learning by reusing the knowledge from models of alternative or previous versions (HUISTRA; MEIJER; VAN DE POL, 2018). As a result, maintained states shall be reached and distinguished at the cost of fewer queries than in *learning from scratch*. A few studies (GROCE; PELED; YANNAKAKIS, 2002; CHAKI *et al.*, 2008; WINDMÜLLER *et al.*, 2013; HUISTRA; MEIJER; VAN DE POL, 2018) have shown that pre-existing models can steer

learning to maintained states by reusing sequences applied in the past queries and hence, reduce the cost for learning models of evolving systems.

While adaptive techniques can boost learning, after several releases *over time*, old separating sequences may lead to *deprecated* queries that no longer distinguish states. Similarly, former transfer sequences may become *redundant* and stop accessing different states. These are known to be significant threats for the efficient application of model learning (HUISTRA; MEIJER; VAN DE POL, 2018). Additionally, in this PhD Thesis, we also claim that these threats can affect projects where evolution occurs *in space* (CLEMENTS; NORTHROP, 2001).

Software product lines (SPL) offer effective means to support the mass production and customization of families of software products (CLEMENTS; NORTHROP, 2001). Unlike traditional systems that are independent and self-contained, SPLs are developed for reuse and with reuse. Therefore, products are not created anew but derived from reusable assets (VAN DER LINDEN; SCHMID; ROMMES, 2007). SPLs continuously deliver versions as requirements evolve and concurrently produce variants satisfying the needs of distinct market segments. These are respectively known as variability *over time* and *in space* and describe the existence of artifacts with different features at different times and the same instant (POHL; BÖCKLE; VAN DER LINDEN, 2005). Thus, the cost and time-to-market will decrease, while the quality of individual products increases (OSTER *et al.*, 2011).

Analyzing (e.g., validating, verifying, and testing) and maintaining SPLs on a product-based basis is demanding due to the number of valid configurations (THÜM *et al.*, 2014a). Hence, substantial effort has been spent on extending notations and associated reasoning techniques to SPLs (GRULER; LEUCKER; SCHEIDEMANN, 2008; CLASSEN *et al.*, 2013; BEOHAR; MOUSAVI, 2014; FRAGAL; SIMAO; MOUSAVI, 2017). These have led to family-based techniques relying on unified representations of all valid products known as *family model* (THÜM *et al.*, 2014a) or *150% model* (BEUCHE; SCHULZE; DUVIGNEAU, 2016).

Family models are FSMs annotated with propositional logic formulae to express *presence conditions* for states and transitions (FRAGAL; SIMAO; MOUSAVI, 2017), i.e., the combination of features involved in the concerned part of the model (THÜM *et al.*, 2014a). Thus, using SAT solvers (BERRE; PARRAIN, 2010) and feature modeling (KANG *et al.*, 1990), family models are amenable to model-based testing (UTTING; PRETSCHNER; LEGEARD, 2012) and model checking (BAIER; KATOEN, 2008) techniques where redundant analysis of shared assets are avoided or minimized. Therefore, the cost of family-based analysis becomes mainly determined by the number and size of features and the amount of feature sharing, rather than the number of valid products (THÜM *et al.*, 2014a).

Despite these possibilities, the creation and maintenance of family models are challenging tasks (OSTER, 2012) due to crosscutting features that may hurdle traceability (SCHAEFER *et al.*, 2012). Additionally, the lack of maintenance may render outdated family models as it happens in the life cycle of traditional systems (WALKINSHAW, 2013).

1.2 Problem Statement and Research Objectives

In this PhD Thesis, we address the problem of learning models from evolving systems in the following sense:

Research Problem

Given an *evolving system* that has changed *over time* (*in space*) where its versioning scheme (variability model) is known, but version-specific FSMs (family models) are *unavailable* or *outdated*, how can we *efficiently* and *effectively* learn (family-based) *finite state machines* specifying its behavior?

While there are several studies on model learning *from scratch* (IRFAN; ORIAT; GROZ, 2013; MARIANI; PEZZÈ; ZUDDAS, 2015; VAANDRAGER, 2017; AICHERNIG *et al.*, 2018), there are only a few works on *adaptive learning variants* (GROCE; PELED; YANNAKAKIS, 2002; CHAKI *et al.*, 2008; WINDMÜLLER *et al.*, 2013; HUISTRA; MEIJER; VAN DE POL, 2018). Additionally, there is a research gap intersecting model learning and software variability analysis *in space*.

Thus, in this PhD Thesis, we pursued three research objectives. These are referred to as `Learning to reuse`, `Learning from difference`, and `Learning by sampling`. The research objectives and their relationships are illustrated in Figure 1.

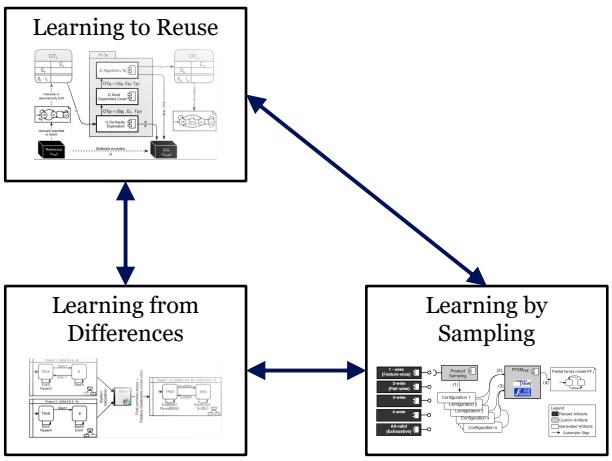


Figure 1 – Research Objectives of this PhD Thesis

Source: Elaborated by the author.

In the next sections, each of these research objectives and their relationships are discussed, how they have been addressed, our main findings, and references to manuscripts that have been published or submitted.

1.2.1 Learning to reuse

As the first research objective, it has been **investigated optimization strategies to mitigate threats to the performance of adaptive learning** led by *deprecated* and *redundant* sequences. To address these issues, we have improved upon the state of the art of model learning by proposing the `partial-Dynamic` L^*_M (∂L^*_M) algorithm, an adaptive technique that explores *on-the-fly* observation tables to efficiently and effectively build models from systems evolving *over time*.

To evaluate the ∂L^*_M technique, we have extended the LearnLib framework for automata learning (RAFFELT; STEFFEN, 2006) and relied on FSMs from a large-scale analysis of several versions of the OpenSSL toolkit (DE RUITER, 2016). We have empirically shown that the ∂L^*_M technique achieves the same effectiveness of the state of the art for adaptive learning but higher efficiency than the existing approaches in terms of `MQ`s. Additionally, our technique has been less sensitive to evolution *over time* than the other techniques. These findings indicate that ∂L^*_M is an efficient and effective adaptive technique.

For learning models from systems that evolve *over time*...

We have introduced the ∂L^*_M algorithm to mitigate the costs for learning Mealy machines from evolving systems. We have performed an experiment to compare our technique against three state-of-the-art adaptive algorithms (CHAKI *et al.*, 2008; HUISTRA; MEIJER; VAN DE POL, 2018). We have used as subjects a set of models of realistic size and structure from the OpenSSL project (DE RUITER, 2016). Our results have indicated that the ∂L^*_M algorithm is effective in learning models and more efficient in terms of `MQ`s.

The `partial-Dynamic` L^*_M algorithm has been published as a regular paper at the 15th International Conference on integrated Formal Methods held in Bergen, Norway (DAMASCENO; MOUSAVI; SIMAO, 2019b). For the sake of reproducibility and repeatability, I open-sourced our code artifacts, FSMs, and test scripts in a lab package available on GitHub at <https://github.com/damascenodiego/DynamicLstarM/releases/tag/iFM2019/>.

1.2.2 Learning from difference

As it has been previously discussed, software modifications may also emerge in terms of variability *in space*, i.e., if multiple versions shall co-exist to satisfy the needs of distinct market segments (CLEMENTS; NORTHROP, 2001; POHL; BÖCKLE; VAN DER LINDEN, 2005). The modeling and maintenance of variability-intensive systems have been reported to be challenging (CLASSEN *et al.*, 2013) as there is often an exponential number of valid products (PERROUIN *et al.*, 2010) and crosscutting features (SCHAEFER *et al.*, 2012; OSTER, 2012; BENDUHN, 2014).

Motivated by these issues, we have extended the investigations on model learning towards SPLs and pursued a second research objective to **formulate approaches for leveraging the concept of *model learning* to *behavioral variability analysis***. To date, this is the first work investigating the problem that we refer to as *family model learning*.

By incorporating feature model analysis (BENAVIDES; SEGURA; RUIZ-CORTÉS, 2010) into model learning algorithms (WALKINSHAW; BOGDANOV, 2013), we have introduced $FFSM_{Diff}$, a fully automated technique to learn featured finite state machines (FFSM), a family-based formalism that unifies Mealy Machines of SPLs into a single representation (FRAGAL; SIMAO; MOUSAVI, 2017). Our technique incorporates variability to identify similar behavior shared among products FSMs (WALKINSHAW; BOGDANOV, 2013), annotate states and transitions with feature constraints, and integrate them into succinct FFSMs.

To evaluate our technique, we have employed the FeatureIDE framework for feature-oriented development (THÜM *et al.*, 2014b) and the Apache Commons Mathematics Library (Apache, 2016) into the LearnLib framework for automata learning (LearnLib, 2017) to find similar behavior shared between product-specific FSMs. As subject systems, we have relied on abstract representations of SPLs from academic benchmarks (FRAGAL; SIMAO; MOUSAVI, 2017; CLASSEN, 2010) that included non-trivial aspects, such as the possibility of infinite behavior and the existence of states with similar or identical behavior in different products (WALKINSHAW; BOGDANOV, 2013). Our empirical results have indicated that the $FFSM_{Diff}$ algorithm can effectively merge families of product FSMs into succinct family models, especially if there is high feature reuse among products.

Our study paves the way to the adoption of family-based analysis techniques even when family models are unavailable or outdated. It can support domain engineering (CLEMENTS; NORTHROP, 2001; POHL; BÖCKLE; VAN DER LINDEN, 2005), SPL re-engineering (FENSKE; THüM; SAAKE, 2013), SPL evolution (MARQUES *et al.*, 2019), and traceability analysis (VALE *et al.*, 2017). The ideas surrounding our algorithm can be extended to other family-based notations (BENDUHN *et al.*, 2015; CLASSEN *et al.*, 2013; BEOHAR; MOUSAVI, 2014).

For learning family models from product-lines...

We have introduced the $FFSM_{Diff}$ algorithm, an automated technique for family model learning. Our technique can identify similar behavior shared among product FSMs, integrate them into a succinct family model, and annotate its states and transitions with feature constraints. We have presented an experiment evaluating our technique and showing its effectiveness for learning family models. Our results have indicated that our algorithm is effective and that the amount of feature reuse is a factor that affects the performance of family model learning.

A full paper about this contribution has been published at the 23rd International Systems and Software Product Line Conference held in Paris, France, in September 2019 (DAMASCENO; MOUSAVI; SIMAO, 2019a). For the sake of reproducibility, I have included a lab package with a variety of artifacts (e.g., source code, test scripts, FFSMs, FSMs, feature models) on GitHub at <https://github.com/damascenodiego/learningFFSM/releases/tag/splc19>.

An extended abstract presenting an overview and indicating future work for this PhD Thesis has been presented and published in the PhD-iFM'19, a PhD Symposium at International Conference on integrated Formal Methods 2019, held in Bergen, Norway, on December 3, 2019 (DAMASCENO, 2019).

1.2.3 Learning by sampling

To close this PhD Thesis, we have pursued the third and last research objective to **investigate optimization techniques towards efficient family model learning**. Thus, we have extended our expertise upon the $FFSM_{Diff}$ algorithm and reported experiences on incorporating product sampling (VARSHOSAZ *et al.*, 2018) into the process of family model learning (DAMASCENO; MOUSAVI; SIMAO, 2019a).

Using 105 product FSMs derived from six SPLs (SAMIH *et al.*, 2014; CLASSEN, 2010; FRAGAL; SIMAO; MOUSAVI, 2017), we have shown that family models learned from a subset of sampled product configurations (PERROUIN *et al.*, 2010) can be as precise as those learned using exhaustive analysis. These results pave the way towards optimized family model learning procedures for product-lines by means of product sampling.

For optimizing family model learning for product-lines...

We have extended our analysis upon the $FFSM_{Diff}$ algorithm by incorporating product sampling. We have designed an experiment using 105 products from six SPLs of academic benchmarks to analyze the effectiveness of our algorithm on learning succinct FFSMs. We have shown that the amount of feature reuse is a factor that affects the size of learned family models. Our results have shown that family models learned by sampling can be as precise as those learned by exhaustive analysis.

These results have been submitted as a journal paper for a Special Issue[1] on "Configurable Systems" in the Empirical Software Engineering Journal (DAMASCENO; MOUSAVI; SIMAO, 2020). The full set of plots, tabulated results, coding artifacts, and models are available on GitHub at the link <https://github.com/damascenodiego/learningFFSM/releases/tag/EMSE>.

[1] <https://www.springer.com/journal/10664/updates/17198898>

1.3 Structure of this PhD Thesis

The remainder of this PhD Thesis is organized as follows:

Chapter 2: First, it introduces the basic definitions of software testing and model-based testing. Second, it presents model learning, reviews the literature on its adaptive learning variants, and discusses techniques to quantify the precision of learned models using state-based model comparison. Third, it covers the concept of software product lines and approaches to support the analysis of SPLs, such as product sampling and family-based modeling using Feature Finite state Machine.

Chapter 3: First, it discusses the problem of learning models from evolving systems from the perspective of evolution *over time*. Second, it introduces the $\partial \mathtt{L}^*_{\mathtt{M}}$ algorithm, an approach to mitigate problems of reusing *redundant* and *deprecated* sequences that may undermine the performance of adaptive learning. Third, using realistic FSMs from the OpenSSL toolkit, it presents an empirical analysis upon our technique to show that, by exploring observation tables on-the-fly, it is less sensitive to software evolution than the state of the art for adaptive learning and achieves the same effectiveness but higher efficiency than the existing approaches for adaptive learning in terms of MQs. This chapter discusses the contribution towards the first research objective of this PhD Thesis.

Chapter 4: First, it proposes the $FFSM_{Diff}$ algorithm, a technique for learning family models from product-specific FSMs of product-lines. Second, using academic benchmarks of abstract representations of SPLs given in terms of FSMs, it shows that the $FFSM_{Diff}$ algorithm can automatically learn succinct family models out of product-specific FSMs, and include new product-specific behavior into an existing family model. Third, it extends our analysis upon the $FFSM_{Diff}$ algorithm for learning family models using product sampling. Finally, it also discusses the impact of incorporating different product sampling criteria into the process of family model learning. This chapter discusses the contribution towards the second and third research objective of this PhD Thesis.

Chapter 5: We close this PhD Thesis by reviewing its contributions in terms of publications that have been published or submitted. It enumerates a few related work, research assumptions and limitations to this study. Finally, we indicate a non-exhaustive list of possibilities of future work that this PhD Thesis shall open the opportunity to be conducted.

CHAPTER

2

BACKGROUND

Many circumstances exist for injecting human mistakes into activities involved in the software development life cycle (DEUTSCH, 1981). Mistakes can happen at the beginning of the requirements elicitation as well as later in the design and development stages, leading to defects and failures (IEEE, 2010). To mitigate these issues, there is a range of software engineering principles to build quality collectively named verification and validation (IEEE, 2012).

Verification and validation (V&V) concern whether products conform to their requirements and satisfy their intended use and user needs (IEEE, 2010). V&V activities are often classified as *static* or *dynamic* analysis. In static analysis, a system or component is evaluated without execution, (e.g., based on its form, structure, content, or documentation) to detect problems or violations of development standards (MYERS; SANDLER; BADGETT, 2012); and, in *dynamic analysis*, the evaluation is based on its behavior during program or model execution with a set of inputs and observing the resulting behavior (AMMANN; OFFUTT, 2008). Software testing is the primary technique for V&V used in industry (BROY *et al.*, 2005).

According to Binder (1999), the activity of software analysis and testing are supposed to be a *model-based* activity. In these activities, test models can be implicit in test engineers' minds, informally sketched on papers, and dependent on the engineers' expertise (MARINESCU *et al.*, 2015). Therefore, an efficient and effective testing often tends to be time-consuming (BERG; RAFFELT, 2005). Testing component-based and service-oriented systems can also include other challenges as they usually lack explicit models (ISBERNER; HOWAR; STEFFEN, 2014a). To tackle these issues, recent studies have been incorporating model learning into software analysis and testing (MARIANI; PEZZÈ; ZUDDAS, 2015).

Coined by Angluin (1987), model learning has been introduced as an *active* procedure to formulate a hypothesis $\mathscr{H}$ about the "language" (i.e., behavior) of a system under learning (SUL). Model learning can be thought of as the inverse process of testing where inputs are pursued to derive a test model that fits the behavior of the SUL (WEYUKER, 1983), rather than

to describe aspects of the expected and actual behaviors (BROY *et al.*, 2005).

Although model learning has been harnessed for a range of problems, its application in industry is still hampered by scalability issues (DUHAIBY *et al.*, 2018); agile methodologies (MEINKE; WALKINSHAW, 2012), where requirements and implementations are continuously evolving; and software diversity (SCHAEFER *et al.*, 2012), where commonalities and variability are managed to develop reusable assets and derive products (CLEMENTS; NORTHROP, 2001). Thus, these may need *learning from scratch* (HUISTRA; MEIJER; VAN DE POL, 2018).

in this chapter, we present the theoretical background supporting the research objectives of this PhD thesis as follows: in Section 2.1, we introduce the basic definitions of software testing and place a special emphasis on FSM-based testing; in Section 2.2, we introduce model learning, review the literature on its adaptive variants, and introduce an approach for quantifying the precision of learning algorithms by means of model comparison; in Section 2.3, we cover concepts of software product lines (SPL), behavioral modeling of SPLs using the Feature Finite-state Machine (FFSM) notation, and techniques for SPL testing, such as configuration similarity and product sampling.

2.1 Software testing

Software testing is a dynamic analysis technique in which a system under test (SUT) is executed under specified conditions, results are observed, and an evaluation of some aspects of the SUT is made (IEEE, 2010). A test case specifies the *test inputs*, *execution conditions*, *expected outputs*, and *execution order* (IEEE, 2010). A set of test cases is called *test suite*. The evaluation of obtained outputs is performed using a test oracle (JORGENSEN, 2013).

A *test oracle* is an instrument that determines whether a given obtained output is an acceptable behavior of the SUT or not (BARR *et al.*, 2015). Test oracles are classified into four categories: (i) specified oracles, (ii) derived oracles, (iii) implicit oracles, and (iv) human oracles. Table 1 describes each of these categories of test oracles with examples.

Exhaustive testing executes the SUT using all possible test inputs from its input domain. Since input domains are often infinite or significantly large, exhaustive testing is not feasible in practice. Moreover, determining whether a SUT is correct or not is, in general, impossible due to theoretical limits. Thus, testing criteria are used to systematize the task of software testing. Figure 2 depicts the relationship between the concepts of software testing.

Testing techniques and *testing criteria* define what specific elements of a *software artifact* that a test case has to exercise (AMMANN; OFFUTT, 2008). *Testing criteria* define what specific elements of a *test artifact* (i.e., *test requirements*) a test case has to exercise (AMMANN; OFFUTT, 2008) to constitute a "*thorough*" test suite (GOODENOUGH; GERHART, 1975). Testing criteria can support test generation from different kinds of artifacts (e.g., source code,

Table 1 – Categories of test oracles

Oracle	Relies on	Examples
Specified	Formal specifications to judge whether the SUT has an acceptable behavior	Mealy machines (BROY *et al.*, 2005), contracts (BURDY *et al.*, 2005), assertions (MASSOL; HUSTED, 2003)
Derived	Information derived from documentations, system executions, properties, alternative versions of SUTs	Pseudo-oracles (WEYUKER, 1982), regression testing (ELBAUM *et al.*, 2004), model learning (ISBERNER; HOWAR; STEFFEN, 2015), and invariant detection (ERNST *et al.*, 2007)
Implicit	General knowledge (e.g., buffer overflows, segfaults, exceptions)	Fuzzing (BEKRAR *et al.*, 2011), model learning (DE RUITER; POLL, 2015)
Human	Reducing human involvement in evaluating test outputs	Test suite minimization (YOO; HARMAN, 2012), and meta-heuristics (AFSHAN; MCMINN; STEVENSON, 2013)

Source: Adapted from Barr *et al.* (2015).

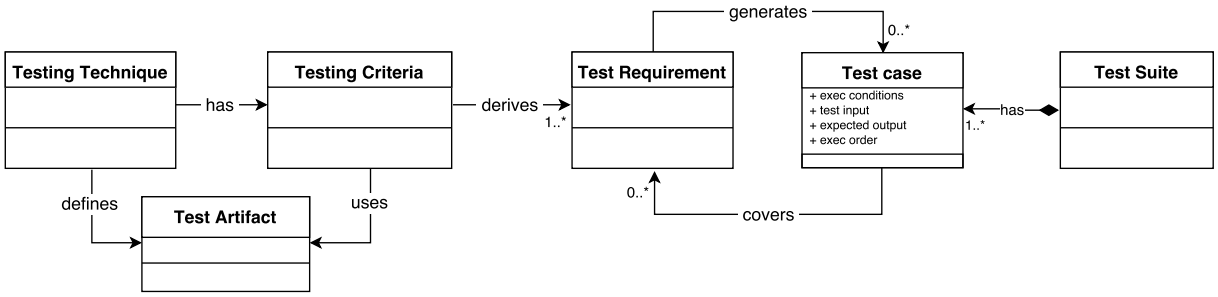


Figure 2 – Relationships between software testing concepts

Source: Adapted from Machado, Vincenzi and Maldonado (2010).

models) (MACHADO; VINCENZI; MALDONADO, 2010).

For the variant of software testing that relies on explicit models as test artifact, we give the name of model-based testing (MBT) (UTTING; PRETSCHNER; LEGEARD, 2012). In this PhD Thesis, we focus on the variant of MBT that relies on complete deterministic Mealy machines (GILL, 1962), hereafter called finite-state machines (FSM).

2.1.1 Finite-state machines-based testing

Finite-state machine-based testing is a specialization of MBT that relies on FSMs to specify test models (BROY *et al.*, 2005). FSMs have been successfully used to denote the behavior of hardware (DUHAIBY *et al.*, 2018), software (HUISTRA; MEIJER; VAN DE POL, 2018), and communication protocols at abstract levels (DE RUITER, 2016).

Definition 2.1.1. (Complete Deterministic FSM) An FSM $\mathscr{M} = \langle S, s_0, I, O, \delta, \lambda \rangle$ is a 6-tuple where S is the finite set of states, $s_0 \in S$ is the initial state, I is the set of inputs, O is the set of outputs, $\delta : S \times I \to S$ is the transition function, and $\lambda : S \times I \to O$ is the output function.

Initially, an FSM is in the initial state s_0. Given a current state $s_i \in S$, when a defined

input $x \in I$, such that $(s_i, x) \in S \times I$, is applied, the FSM responds by moving to state $s_j = \delta(s_i, x)$ and producing output $y = \lambda(s_i, x)$. The concatenation of two inputs α and ω is denoted by $\alpha \cdot \omega$. An input sequence $\alpha = x_1 \cdot x_2 \cdot \ldots \cdot x_n \in I^*$ is defined in state $s \in S$ if there are states $s_1, s_2, \ldots, s_{n+1}$ such that $s = s_1$ and $\delta(s_i, x_i) = s_{i+1}$, for all $1 \leq i \leq n$. Transition and output functions are lifted to input sequences, as usual. For the empty input sequence ε, $\delta(s, \varepsilon) = s$ and $\lambda(s, \varepsilon) = \varepsilon$. For a non-empty input sequence $\alpha \cdot x$ defined in state s, we have $\delta(s, \alpha \cdot x) = \delta(\delta(s, \alpha), x)$ and $\lambda(s, \alpha \cdot x) = \lambda(s, \alpha)\lambda(\delta(s, \alpha), x)$.

An input sequence α is a prefix of β, denoted by $\alpha \leqslant \beta$, when $\beta = \alpha \cdot \omega$, for some sequence ω. An input sequence α is a proper prefix of β, denoted by $\alpha < \beta$, when $\beta = \alpha \cdot \omega$, for $\omega \neq \varepsilon$. The prefixes of a set T are denoted by $pref(T) = \{\alpha | \exists \beta \in T, \alpha \leq \beta\}$. If $T = pref(T)$, it is *prefix-closed*.

An input sequence $\alpha \in I^*$ is a transfer sequence from s to s', if $\delta(s, \alpha) = s'$. An input sequence γ is a separating sequence for $s_i, s_j \in S$ if $\lambda(s_i, \gamma) \neq \lambda(s_j, \gamma)$. Two states $s_i, s_j \in S$ are equivalent if, for all $\alpha \in I^*$, $\lambda(s_i, \alpha) = \lambda(s_j, \alpha)$, otherwise they are distinguishable. An FSM is *deterministic* if, for each state s_i and input x, there is at most one possible state $s_j = \delta(s_i, x)$ and output $y = \lambda(s_i, x)$. Notice that our definition only allows for complete deterministic FSMs, which are the focus of this PhD Thesis. If all states of an FSM are pairwise distinguishable, then the FSM is *minimal*.

Example 2.1.1. (The windscreen wiper FSM) Figure 3 depicts a windscreen wiper system supporting intervaled and fast wiping, if any raindrop is sensed, such that $S = \{off, itv, rain\}$, $I = \{rain, swItv\}$ and $O = \{0, 1\}$. Transition and output functions are represented by directed edges labeled with input/output symbols.

Figure 3 – The windscreen wiper FSM

An input sequence $\alpha \in \Omega_M$ starting with a reset r is a *test case* of M. Given two test cases $\alpha, \beta \in T$, if α is a proper prefix of β, the execution of β implies the execution of α; thus, α can be discarded from T without affecting test results. A test suite of M consists of a finite set T of test cases, such that there are no two sequences $\alpha, \beta \in T$ where $\alpha < \beta$. The number of symbols of a sequence α is represented by $|\alpha|$ and describes the length of the test sequence α. Given a test case α, the execution cost is calculated as $|\alpha| + 1$ (i.e., the length of α plus one reset operation). The number of resets of T (i.e., number of test cases) is represented by $|T|$.

In FSM-based testing, we aim at evaluating whether the transitions between the states of a SUT are correct and hence prove the equivalence between two FSMs (BROY *et al.*, 2005). Two FSMs $M_S = \langle S, s_0, I, O, \delta, \lambda \rangle$ and $M_I = \langle S', s'_0, I, O', \delta', \lambda' \rangle$ are equivalent ($M_S \equiv M_I$) and vice-versa if for each states of M_S there is an equivalent state in M_I. To achieve this, there are special sequences that underpin most of the techniques in the literature and provide partial information about the SUT.

Definition 2.1.2. (State cover) A set Q of input sequences is *state cover* for $\mathscr{M}$ if $\varepsilon \in Q$ and, for all $s_i \in S$, there is a sequence $\alpha \in Q$ to reach state s_i, i.e., $\delta(s_0, \alpha) = s_i$.

Definition 2.1.3. (Transition cover) A set P of input sequences is *transition cover* for $\mathscr{M}$ if $\varepsilon \in P$ and, for all $(s,x) \in S \times I$, there are sequences $\alpha, \alpha x \in P$ to reach state s and cover the transition labeled with the input x that departs from s, i.e., $\delta(s_0, \alpha) = s$.

Definition 2.1.4. (Characterization set) A set W of input sequences is *characterization set* for $\mathscr{M}$ if for all $s_i, s_j \in S, i \neq j$, there is an $\alpha \in W$ such that $\lambda(s_i, \alpha) \neq \lambda(s_j, \alpha)$.

2.1.1.1 W method

The W method (CHOW, 1978; VASILEVSKII, 1973) is one of the most classic test generation methods for FSM. In the W method, the test case generation proceeds by concatenating the characterization set W to the leaves of the tree representation of the transition cover set P. Thus, the characterization set W identifies all states and transitions in the FSM covered by the transition cover set P, i.e., $r \cdot P \cdot W$.

2.1.1.2 Wp method

The W-partial (Wp) method (FUJIWARA *et al.*, 1991) is a test case generation method, where the term *partial* applies as it uses a subset of the W set. The Wp method is a well-known improvement of the traditional W method (CHOW, 1978; VASILEVSKII, 1973) that attempts to reduce the number of test cases by following a two stages process:

1. **States verification:** to verify the set of states Q in the SUT using a set of test cases $C_1 = r \cdot Q \cdot W$.
2. **Uncovered transitions verification:** to assess the subset of remaining transitions $R = P \backslash Q$ using a set of test cases $C_2 = \bigcup_{\alpha \in R} r \cdot \alpha \cdot W_i$, where $W_i \subset W$ distinguishes $s_i = \delta(s_0, \alpha)$ from all the other states of S.

As many FSM-based test case generation methods, the W (CHOW, 1978; VASILEVSKII, 1973) and the Wp (FUJIWARA *et al.*, 1991) methods have full fault detection capability. Additionally, they can detect an estimated number of extra states m using a traversal set $\bigcup_{i=0}^{m}(I^i)$, where I^i contains all sequences from I^* with length i. However, finding an upper bound m is

often non-trivial as it requires knowledge about the SUT. Hence, underestimation can lead to incorrect models or overestimation can lead to scalability issues.

MBT has been studied for several decades (VASILEVSKII, 1973; CHOW, 1978) and introduced advances in the state-of-the-art and state-of-the-practice of software testing and analysis (MLYNARSKI *et al.*, 2012; SHAFIQUE; LABICHE, 2015; MARINESCU *et al.*, 2015). However, building useful test models is still time-consuming, tedious, error-prone, and dependent on engineers' expertise (BERG; RAFFELT, 2005; PRETSCHNER; PHILIPPS, 2005). Additionally, component-based and service-oriented systems often lack complete specification models (ISBERNER; HOWAR; STEFFEN, 2014a), and in the lack of proper maintenance (WALKINSHAW, 2013), models may become outdated. To mitigate these problems, recent studies (IRFAN; ORIAT; GROZ, 2013; MARIANI; PEZZÈ; ZUDDAS, 2015; VAANDRAGER, 2017; AICHERNIG *et al.*, 2018) have shown that black-box approaches for learning FSMs (ANGLUIN, 1987; SHAHBAZ; GROZ, 2009; MEINKE; SINDHU, 2011a; CASSEL FALK HOWAR, 2015) are becoming increasingly popular in software analysis and testing.

2.2 Model learning

Coined by Angluin (1987), active model learning is an *active* procedure that aims at formulating a hypothesis about the behavior of an SUL by pursuing inputs and observing outputs (IRFAN; ORIAT; GROZ, 2013). It differs from *passive* learning, where models are synthesized from a finite set of traces without interacting with the SUL (TRETMANS, 2011).

According to Weyuker (1983), MBT and active model learning can be thought of as being inverse processes. The MBT process begins with a SUT and specification (i.e., the test model) and looks for test cases to characterize conformance relations (BROY *et al.*, 2005). Model learning starts with a set of queries (i.e., test cases), and derives a behavioral specification (i.e., test model) that fits the behavior of a given SUL (IRFAN; ORIAT; GROZ, 2013). The model learning procedure is often described in terms of the Minimally Adequate Teacher (MAT) framework (ANGLUIN, 1987). In Figure 4, we illustrate the MAT framework.

In the MAT framework, a `learning algorithm` iteratively poses test inputs as `queries` to observe `outputs` and formulate a `hypothesis` (i.e., test model) about the behavior of the SUL. This procedure is supported by a `teacher` (i.e., oracle) able to answer if such a hypothesis is correct or not, and return `counterexamples` that expose differences. Therefore, the MAT framework is organized as an iterative process of two phases: *hypothesis construction*, where a model is built by posing `Membership Queries (MQ)`; and *hypothesis validation*, where the hypothesis is subjected to `Equivalence Queries (EQ)`.

An `Equivalence Query (EQ)` checks whether a `hypothesized model` denotes the real behavior of an SUL. The result of an `EQ` is `yes`, if the hypothesis is correct (i.e., `all pass`), otherwise a `counterexample` points non-conformances (i.e., `failed test`). Often, teachers

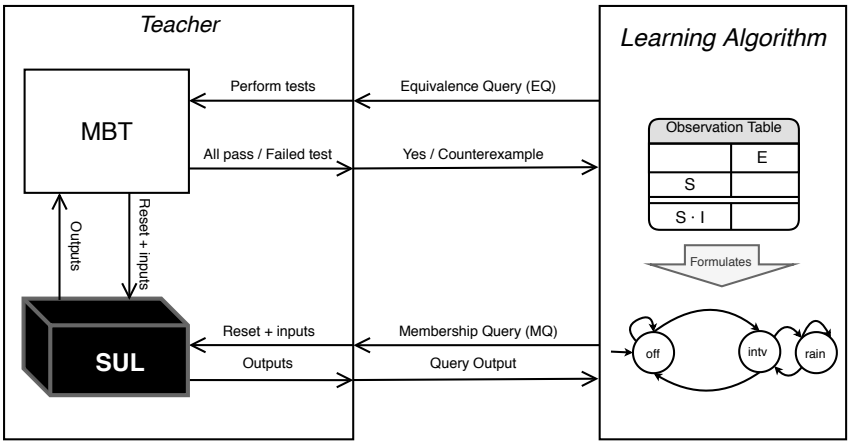


Figure 4 – The Minimally Adequate Teacher (MAT) framework

Source: Adapted from Vaandrager (2017).

use MBT to compute inputs and outputs and reset the SUL to its initial state.

A Membership Query (MQ) consists of inputs and reset performed to gain knowledge (i.e., query output). If a teacher replies with a counterexample, other MQs are asked to improve the hypothesized model until the hypothesis becomes correct. The outputs are organized in an *observation table* that is incrementally refined to build a hypothesis.

Since Angluin (1987) seminal paper, the MAT framework has been extended to other formalisms, such as nondeterministic finite automata (BOLLIG *et al.*, 2009), mealy machines (NIESE, 2003; LI; GROZ; SHAHBAZ, 2006; SHAHBAZ; GROZ, 2009), register automata (AARTS *et al.*, 2012; CASSEL FALK HOWAR, 2015; AARTS *et al.*, 2015; CASSEL *et al.*, 2016), and input-output transition systems (VOLPATO; TRETMANS, 2015). In this PhD Thesis, we discuss model learning in terms of the L_M^* algorithm (SHAHBAZ; GROZ, 2009).

2.2.1 The L_M^* algorithm

Groce, Peled and Yannakakis (2002) have proposed to model the input and output alphabet of SULs as a collection of concatenated symbols (i.e., $\Sigma = I \cup O$). Thus, model learning algorithms designed for deterministic finite automata (ANGLUIN, 1987) could be applied to SULs representable as Mealy machines. However, this increment at the size of the input domain would directly affect the time complexity for learning (GROCE; PELED; YANNAKAKIS, 2002). Shahbaz and Groz (2009) have extended the work by Angluin (1987) to optimize the process of learning to Mealy machines by leveraging the concepts of observation table and membership queries to this formalism and, hence, introducing the L_M^* algorithm.

The L_M^* algorithm (SHAHBAZ; GROZ, 2009) builds hypotheses about an SUL in terms of FSMs by iteratively posing MQ until a correct model is formulated (SHAHBAZ; GROZ, 2009). In the L_M^* algorithm, MQs are also referred to as observation queries (OQ) as mealy machines

express SULs in terms of input/output pairs, rather than (un)accepted inputs. An observation table (OT) is a triple $OT = (S_M, E_M, T_M)$, where $S_M \subseteq I^*$ is a prefix-closed set of inputs labeling the rows of OT, $E_M \subseteq I^+$ is a suffix-closed set of inputs labeling the columns of OT, and $T_M : (S_M \cup S_M \cdot I) \times E_M \to O^+$ maps inputs to outputs.

Definition 2.2.1. (Observation Table) An observation table $\texttt{OT} = (S_M, E_M, T_M)$ is a triple, where $S_M \subseteq I^*$ is a prefix-closed set of transfer sequences (i.e., prefixes); $E_M \subseteq I^+$ is a set of separating sequences (i.e., suffixes); and T_M is a table where rows are labeled by elements from $S_M \cup (S_M \cdot I)$, columns are labeled by elements from E_M, such that for all $pre \in S_M \cup (S_M \cdot I)$ and $suf \in E_M$, $T_M(pre, suf) = \lambda(\delta(q_0, pre), suf)$ where q_0 is the initial state.

Traditionally, the $\texttt{L}^*_\texttt{M}$ algorithm (SHAHBAZ; GROZ, 2009) starts with the sets of prefixes $S_M = \{\varepsilon\}$ and suffixes $E_M = I$ so that it can reach the initial state and observe the outputs of the outgoing transitions, respectively. Two rows $pre_1, pre_2 \in S_M \cup (S_M \cdot I)$ are equivalent, denoted by $pre_1 \cong pre_2$, when for all $suf \in E_M$ it holds that $T_M(pre_1, suf) = T_M(pre_2, suf)$. The equivalence class of a row r is denoted by $[r]$. Thus, the algorithm poses MQs until the properties of closedness and consistency hold:

Definition 2.2.2. (Closedness property) An observation table OT is closed if, for all $pre_1 \in (S_M \cdot I)$, there is a $pre_2 \in S_M$ where $pre_1 \cong pre_2$.

Definition 2.2.3. (Consistency property) An observation table OT is consistent if for all $pre_1, pre_2 \in S_M$, such that $pre_1 \cong pre_2$, it holds that $pre_1 \cdot \alpha \cong pre_2 \cdot \alpha$, for all $\alpha \in I$.

If an observation table is not closed, it searches for a row $s_1 \in S_M \cdot I$, such that $s_1 \ncong s_2$ for all $s_2 \in S_M$, moves it to S_M, and completes the observation table by asking MQs for the new rows. If the observation table is not consistent, it searches for $s_1, s_2 \in S_M, e \in E_M, i \in I$, such that $s_1 \cong s_2$ but $T_M(s_1 \cdot i, e) \neq T_M(s_2 \cdot i, e)$, adds $i \cdot e$ to E_M, and completes the observation table by asking MQs for the new column. Given a *closed* and *consistent* observation table, the $\texttt{L}^*_\texttt{M}$ formulates a hypothesis $\mathscr{H} = (Q_M, q_{0_M}, I, O, \delta_M, \lambda_M)$ where $Q_M = \{[pre] | pre \in S_M\}$, $q_{0_M} = [\varepsilon]$ and, for all $pre \in S_M$, $i \in I$, $\delta_M([pre], i) = [pre \cdot i]$ and $\lambda_M([pre], i) = T_M(pre, i)$.

After formulating $\mathscr{H}$, the $\texttt{L}^*_\texttt{M}$ algorithm works under the assumption that an EQ returns either a counterexample (CE) exposing the non-conformance, or yes, if $\mathscr{H}$ is equivalent to the SUL. When a CE is found, a CE processing method adds prefixes and/or suffixes to the OT and refines $\mathscr{H}$. These steps are repeated until $\texttt{EQ} = \texttt{yes}$. For black-box systems, EQs are often approximated using random walks (ANGLUIN, 1987; HOWAR; STEFFEN; MERTEN, 2010), conformance testing (CHOW, 1978; VASILEVSKII, 1973; FUJIWARA *et al.*, 1991), or both (ISBERNER; HOWAR; STEFFEN, 2015; MEINKE; SINDHU, 2011b).

Example 2.2.1. (OT from the windscreen wiper FSM) In Table 2, we show an observation table built using $\texttt{L}^*_\texttt{M}$, a $\texttt{CE} = \texttt{swItv} \cdot \texttt{rain} \cdot \texttt{rain} \cdot \texttt{rain}$ and the processing method by Rivest

and Schapire (1993) that uses binary search to find the shortest suffix from CE that refines a hypothesis. The cost to build this OT is 24 MQs and 1 EQ.

Table 2 – OT extracted from the windscreen wiper FSM

		rain	*swItv*	*rain* · *rain*
S_r	ε	0	1	0 · 0
	swItv	1	0	1 · 0
	swItv · *rain*	0	1	0 · 1
$S_r \cdot I_r$	*rain*	0	1	0 · 0
	swItv · *swItv*	0	1	0 · 0
	swItv · *rain* · *rain*	1	0	1 · 0
	swItv · *rain* · *swItv*	0	1	0 · 1

The worst-case complexity of the algorithm for the number of MQs is $\mathcal{O}(|I|^2mn + |I|mn^2)$ parameterized on the size of the input domain I, the length m of the longest CE and the number of states n of the minimal FSM describing the SUL.

In Algorithm 1, we present the L^*_M algorithm (SHAHBAZ; GROZ, 2009). As input, the algorithm takes the SUL and a fixed input alphabet I. As output, the algorithm returns a hypothesis $\mathscr{H}$ modeling the reactive behavior of the SUL.

Source: Adapted from Shahbaz and Groz (2009).

Motivated by the impact of CEs on the complexity of the $\mathtt{L}^*_\mathtt{M}$ algorithm, there is a range of processing methods available in the literature (IRFAN; ORIAT; GROZ, 2013). Additionally, caching filters have been incorporated as a component for pre-processing and avoiding redundant queries (MARGARIA; RAFFELT; STEFFEN, 2005). Finally, there is also a branch of research that investigates variants for learning that rely on previously inferred models to reduce the costs of model learning for evolving systems, namely adaptive model learning (GROCE; PELED; YANNAKAKIS, 2002; CHAKI *et al.*, 2008; WINDMÜLLER *et al.*, 2013; HUISTRA; MEIJER; VAN DE POL, 2018).

2.2.2 Adaptive model learning

Adaptive model learning is a variant of learning which attempts to speed up learning by reusing pre-existing models, e.g., from previous/alternative versions (GROCE; PELED; YANNAKAKIS, 2002). In adaptive learning, transfer and/or separating sequences built from pre-existing models are used to initialize learning algorithms with sets of prefixes and suffixes and attempt to perform better than starting from the traditional sets of sequences for reaching the initial state (i.e., $S_M = \{\varepsilon\}$) and collecting outputs from outgoing transitions (i.e., $E_M = I$). Therefore, a reduction in the number of MQs and EQs may be obtained.

Algorithm 1 – The L^*_M Algorithm

Input: The SUL and the input alphabet I
Output: Hypothesized mealy machine $\mathscr{H}$

procedure MEALYLEARNING(SUL, I): Hypothesized model $\mathscr{H}$
  Initialize the rows $S_M = \{\varepsilon\}$, columns $E_M = I$ and $S_M \cdot I = \{\varepsilon \cdot i\}$, for all $i \in I$
  Complete $OT = (S_M, E_M, T_M)$ by asking MQs, for all $s \in (S_M \cup S_M \cdot I)$ and $e \in E_M$
  repeat
    while (S_M, E_M, T_M) is *not closed* or *not consistent* **do**
      if (S_M, E_M, T_M) is *not consistent* **then**
        Find $s_1, s_2 \in S_M$, $e \in E_M$, $i \in I$ where $s_1 \cong s_2$, but $T_M(s_1 \cdot i, e) \neq T_M(s_2 \cdot i, e)$
        $E_M = E_M \cup \{i \cdot e\}$ ▷ add $i \cdot e$ to E_M
        Complete (S_M, E_M, T_M) by asking MQ for the new column $i \cdot e$
      end if
      if (S_M, E_M, T_M) is *not closed* **then**
        Find $s_1 \in S_M \cdot I$, such that $s_1 \ncong s_2$, for all $s_2 \in S_M$
        $S_M \cdot I = S_M \cdot I \setminus s_1$; $S_M = S_M \cup \{s_1\}$ ▷ move s_1 to S_M
        $S_M \cdot I = S_M \cdot I \cup \{s_1 \cdot i\}$, for all $i \in I$ ▷ add $s_1 \cdot i$ to $S_M \cdot I$, for all $i \in I$
        Complete (S_M, E_M, T_M) by asking MQ for the newly added rows
      end if
    end while
    Build a hypothesized model $\mathscr{H}$ from $OT = (S_M, E_M, T_M)$
    Ask an EQ to the MAT
    if MAT replies `counterexample` to EQ **then**
      Process `counterexample` ▷ e.g., add all the prefixes to S_M
      Complete (S_M, E_M, T_M) by asking MQ for the missing elements
    end if
  until MAT replies `Yes` to an EQ
  return Hypothesis $\mathscr{H}$ built from $OT = (S_M, E_M, T_M)$
end procedure

To the best of our knowledge, there are only four studies that address problems related to adaptive model learning (GROCE; PELED; YANNAKAKIS, 2002; WINDMÜLLER *et al.*, 2013; HUISTRA; MEIJER; VAN DE POL, 2018; CHAKI *et al.*, 2008). Each of these studies is discussed in the next sections.

Adaptive model checking

Groce, Peled and Yannakakis (2002) have been one of the earliest to address the task of reusing sequences from inaccurate (but not completely irrelevant) models to reduce the time spent on model learning and model checking. To investigate this issue, the authors have performed an experimental study to evaluate the benefits of reusing either transfer sequences or separating sequences, or both sequences, against the costs for standard *learning from scratch*. Their results have indicated that adaptive learning can be efficient, especially when there are minor modifications, or when these changes may have a very limited effect on the correctness of properties checked.

Active continuous quality control

Windmüller *et al.* (2013) have presented an adaptive learning technique for periodically inferring automata models from complex applications that evolve over time. Their findings have shown that the reuse of separating sequences taken from models of previous versions shall enable model learning algorithms to find states maintained in newer versions.

Adaptive learning for regression testing

Huistra, Meijer and van de Pol (2018) have shown that the performance of adaptive model learning algorithms is influenced by three factors: (i) the complexity of the SUL, (ii) the amount of difference between the model of the reused version and the SUL, and (iii) the quality of the suffixes. Thus, if a set of reused separating sequences has low quality (i.e., lacks the capacity of distinguishing states in a further release), the authors indicate that adaptive learning may tend to pose irrelevant queries, and hence perform an extra effort to reach novel states. To mitigate these issues, the authors have suggested that calculating subsets of good separating sequences should reduce the number of deprecated sequences.

Verification of evolving software via component substitutability analysis

When software evolves, specification models may become outdated and hamper the effective application of model checking. (MARIANI; PEZZÈ; ZUDDAS, 2015). To tackle these issues, Chaki *et al.* (2008) have introduced *DynamicCheck*, an approach to reduce the cost for model checking software upgrades (SERY; FEDYUKOVICH; SHARYGINA, 2015). Central to their approach is `Dynamic` $\mathtt{L}^*$, an adaptive learning algorithm that reuses observation tables from previous versions for inferring DFAs (ANGLUIN, 1987). As a result, upgrade checking can succeed in a small fraction of the time to verify its reference version (CHAKI *et al.*, 2008).

Originally, the $\mathtt{L}^*_{\mathtt{M}}$ starts from $S = \{\varepsilon\}$, $E = I$ and hence, if there is any previously learned model from some reference version v_{ref}, it misses opportunities for optimizing the learning process. To this end, `Dynamic` $\mathtt{L}^*$ restores the *agreement* of an outdated table $\mathtt{OT}_o = (S_r, E_r, T_o)$ built from v_{ref} by re-posing `MQs` to the updated release v_{updt} to build an updated observation table $\mathtt{OT}_r = (S_r, E_r, T_r)$.

Definition 2.2.4. (Agreement) An $\mathtt{OT}_r = (S_r, E_r, T_r)$ agrees with v_{updt} if and only if, for all $s \in (S_r \cup S_r \cdot I_u)$ and $e \in E_r$, it holds $T_r(s,e) = \lambda_u(s,e)$, where λ_u is an output function of v_{updt}.

After restoring the agreement, the observation table $\mathtt{OT}_r$ may have redundant prefixes and deprecated suffixes. To discard them, the `Dynamic` $\mathtt{L}^*$ searches for a smaller $S_R \subseteq S_r$ with the same state coverage capability but fewer prefixes, referred to as *well-formed cover* (CHAKI *et al.*, 2008).

Definition 2.2.5. (Well-Formed cover subset) Let S_r be the set of prefixes from an observation table $\mathtt{OT}_r$ in agreement with v_{updt}; a subset $S_R \subseteq S_r$ is a *well-formed cover*, if and only if (i) S_R is prefix-closed, (ii) for all $s1, s2 \in S_R$, it holds that $s1 \ncong s2$, and (iii) S_R is a maximal subset of S_r.

After finding $S_R \subseteq S_r$, it uses a *Column function* to group prefixes into equivalence classes given a subset of suffixes. Thus, it searches for an optimal subset of suffixes $E_R \subseteq E_r$, referred to as the *experiment cover* (CHAKI *et al.*, 2008).

Definition 2.2.6. (Column Function) Let S_R be well-formed cover, an observation table $\mathtt{OT}_{R'} = (S_R, E_r, T_{R'})$ derived from $\mathtt{OT}_r$, the input set I_u of v_{updt}, and an $e \in E_r$; the column function is $Col(S_R, e) : S_R \times E_r \to \{B_1, B_2, ..., B_n\}$ where B_i are non-empty partitions of S_R (i.e., $B_i \subseteq S_R$), $\cap_{i=1}^{n} B_i = \emptyset$, $\cup_{i=1}^{n} B_i = S_R$, $Col(S_R, \varepsilon) = \{S_R\}$ and $x, y \in B_i$ if and only if $T(x, e) = T(y, e)$.

An $E_R \subseteq E_r$ is an *experiment cover subset* iff for all distinct $e_1, e_2 \in E_R$, it holds that $Col(S_R, e_1) \neq Col(S_R, e_2)$ and for all $e' \in E_R$ there is an $e \in E_r$ where $Col(S_R, e) = Col(S_R, e')$. Finally, the $\mathtt{L}^*_{\mathtt{M}}$ algorithm is initialized with the subsets $S = S_R$ and $E = E_R$, rather than the $S = \{\varepsilon\}$ and $E = I$ indicated in the Line 2 of Algorithm 1. Therefore, the cost for upgrade model checking is reduced to a small fraction of the cost needed to verify its reference version from scratch (CHAKI *et al.*, 2008).

2.2.3 Structural comparison of state-based models

According to Walkinshaw and Bogdanov (2013), structurally comparing two state machines is a difficult task that involves establishing similarity relationships between states and transitions. To achieve this goal, the authors have introduced LTS_{Diff}, an algorithm to compute the precise difference between two labeled transition systems, a well-known variant of FSM.

In the LTS_{Diff} algorithm, the similarities between two FSMs are described in terms of states and their surrounding transitions matching input and output symbols. To indicate the surrounding transitions, we incorporate a 7th element D in our definition of FSMs to represent the input domain of each state. Thus, we depict our two FSMs as $M_r = \langle S_r, s_{0_r}, I_r, O_r, D_r, \delta_r, \lambda_r \rangle$ and $M_u = \langle S_u, s_{0_u}, I_u, O_u, D_u, \delta_u, \lambda_u \rangle$. The algorithm first calculates the set of matching transitions for all pairs of states $a \in S_r$ and $b \in S_u$ using the individual number of pairs of states that can be reached via matching transitions, as follows:

$$
\begin{aligned}
Succ_{a,b} = \{(c, d, i, o) \in S_r \times S_u \times (I_r \cup I_u) \times (O_r \cup O_u),\ &\text{such that} \\
\delta_r(a, i) = c,\ \delta_u(b, i) = d,\ &\textit{and} \\
\lambda_r(a, i) = \lambda_u(b, i) = o\}&
\end{aligned}
$$

Second, a global similarity score is calculated by aggregating the scores of all states connected to the original pair as follows:

$$S^{G}_{Succ}(a,b) = \frac{1}{2} \frac{\sum_{(c,d,i,o) \in Succ_{a,b}} (1 + k \times S^{G}_{Succ}(c,d))}{|\sum^{out}_{r}(a) - \sum^{out}_{u}(b)| + |\sum^{out}_{r}(b) - \sum^{out}_{u}(a)| + |Succ_{a,b}|}$$

An attenuation ratio k is used to give precedence to state pairs that are closer to the original pair of states, and the notation $\sum^{out}_{r}(a)$ refers to the set of labels of outgoing transitions for state a of M_r. Therefore, the expression $|\sum^{out}_{r}(a) - \sum^{out}_{u}(b)| + |\sum^{out}_{r}(b) - \sum^{out}_{u}(a)|$ denotes the number of outgoing transitions from both states a and b that do not match each other.

Given two FSMs M_r and M_u, the global similarity score $S^{G}_{Succ}(a,b)$ is used to build a system of linear equations, such that each equation corresponds to the $S^{G}_{Succ}(a,b)$ for one specific pair of states $(a,b) \in S_r \times S_u$.

The global similarity is calculated both in terms of future behavior (i.e., outgoing transitions) and past behaviors (i.e., incoming transitions). The global similarity score for incoming transitions $S^{G}_{Prev}(a,b)$ is calculated in a similar manner.

Let the equations for the outgoing and incoming transitions be $S^{G}_{Succ}(a,b)$ and $S^{G}_{Prev}(a,b)$, the similarity scores for each pair of states (a,b) are averaged as follows:

$$S(a,b) = \frac{S^{G}_{Succ}(a,b) + S^{G}_{Prev}(a,b)}{2}$$

Example 2.2.2. (Illustration of a system of linear equations) In Table 3, we depict the system of equations resulting from the comparison of the FSMs in Figures 5 and 6. State pairs are represented by the first two letters of their respective names.

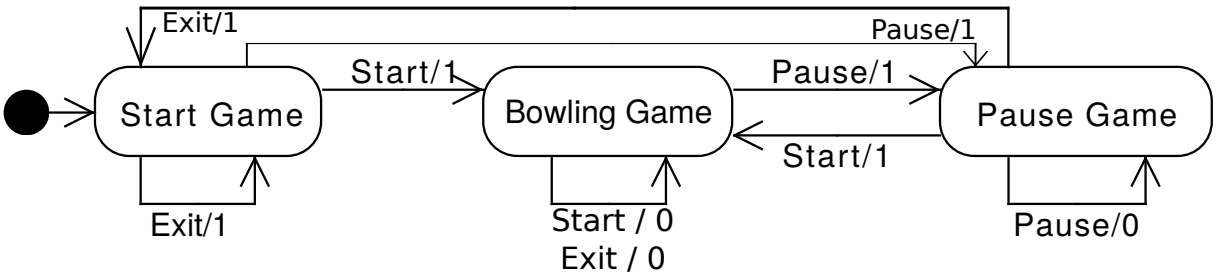


Figure 5 – Example of reference version for an FSM

Source: Elaborated by the author.

2.2.3.1 The LTS_{Diff} algorithm

Given the averaged scores, the comparison of two models is performed in a similar fashion to how we manually navigate using maps in unfamiliar landscapes (WALKINSHAW; BOGDANOV, 2013). This process is described in Algorithm 2.

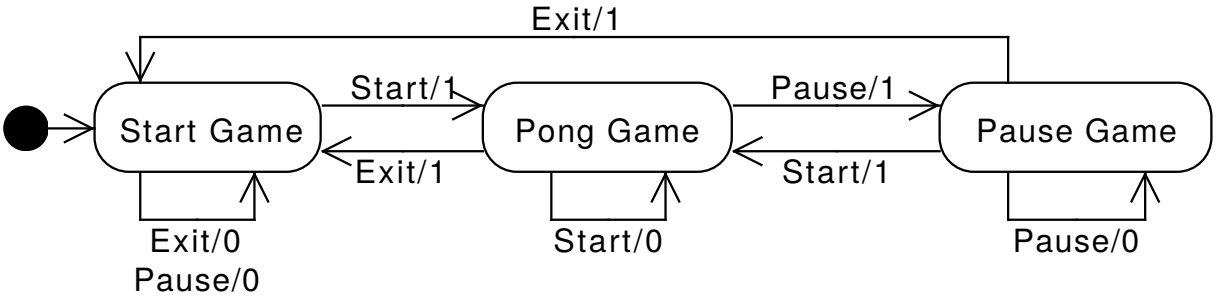


Figure 6 – Example of alternative version for an FSM

Source: Elaborated by the author.

Pair	(St,St)	(St,Po)	(St,Pa)	(Bo,St)	(Bo,Po)	(Bo,Pa)	(Pa,St)	(Pa,Po)	(Pa,Pa)	
(St,St)	10.0	0.0	0.0	0.0	-0.5	0.0	0.0	0.0	0.0	1
(St,Po)	-0.5	8.0	0.0	0.0	0.0	0.0	0.0	0.0	-0.5	2
(St,Pa)	-0.5	0.0	8.0	0.0	-0.5	0.0	0.0	0.0	0.0	2
(Bo,St)	0.0	0.0	0.0	9.5	0.0	0.0	0.0	0.0	0.0	1
(Bo,Po)	0.0	0.0	0.0	0.0	7.5	0.0	0.0	0.0	-0.5	2
(Bo,Pa)	0.0	0.0	0.0	0.0	0.0	12.0	0.0	0.0	0.0	0
(Pa,St)	0.0	0.0	0.0	0.0	-0.5	0.0	7.5	0.0	0.0	2
(Pa,Po)	-0.5	0.0	0.0	0.0	0.0	0.0	0.0	10.0	0.0	1
(Pa,Pa)	-0.5	0.0	0.0	0.0	-0.5	0.0	0.0	0.0	5.5	3

Table 3 – Ilustration of a system of linear equations

Source: Elaborated by the author.

Source: Adapted from Walkinshaw and Bogdanov (2013).

First, in Line 3, a filtering method denoted by *identifyLandmarks* selects the top $t\%$ most equivalent pairs, and, if one state is matched to several others, a ratio r includes only those pairs that are at least r times as good as any other match. If no state is selected, then, in Line 4-6, the initial states are selected as initial landmarks.

Second, in Line 7, the algorithm proceeds from the initial landmarks using $Surr(a,b)$ to reach the surrounding states through the incoming and outgoing transitions matching input/output labels. Thus, it builds a set of candidates matched state pairs *NPairs*.

Third, the set *NPairs* is iterated in the order of their similarity scores. In Line 10, once a pair (a,b) is selected, it is included in the set of confirmed matches *KPairs*, in Line 11, and, in Line 12, all elements that include either a or b are discarded from *NPairs*. This process iterates until no further pairs can be added to *KPairs* and *NPairs* becomes empty. Finally, in Lines 16-18, the *KPairs* set is used to calculate the sets of transitions added, removed, and maintained.

Algorithm 2 – The LTS_{Diff} algorithm

Input: FSM M_r, FSM M_u, k, t, r
Output: Sets of states added removed and maintained (Add, Rem, Kpt)

procedure $LTS_{Diff}(M_r, M_u, k, t, r)$: Sets of states (Add, Rem, Kpt)
  $PairsToScore = computeScores(M_r, M_u, k)$;
  $KPairs = identifyLandmarks(PairsToScore, t, r)$;
  if $KPairs == \emptyset$ and $S(s_{0_r}, s_{0_u}) > 0$ **then**
    $KPairs = (s_{0_r}, s_{0_u})$;
  end if
  $NPairs = \cup_{(a,b)\in KPairs} Surr(a,b) - KPairs$;
  while $NPairs \neq \emptyset$ **do**
    while $NPairs \neq \emptyset$ **do**
      $(a,b) = pickHighest(NPairs, PairsToScore)$;
      $KPairs = KPairs \cup (a,b)$;
      $NPairs = removeConflicts(NPairs, (a,b))$;
    end while
    $NPairs = \cup_{(a,b)\in KPairs} Surr(a,b) - KPairs$;
  end while
  $Add = \{b_1 \xrightarrow{a/b} b_2 \in D_u | \nexists (a_1 \xrightarrow{a/b} a_2 \in D_r \wedge (a_1,b_1) \in KPairs \wedge (a_2,b_2) \in KPairs)\}$;
  $Rem = \{a_1 \xrightarrow{a/b} a_2 \in D_r | \nexists (b_1 \xrightarrow{a/b} b_2 \in D_u \wedge (a_1,b_1) \in KPairs \wedge (a_2,b_2) \in KPairs)\}$;
  $Kpt = KPairs$;
  return *return* (Add, Rem, Kpt);
end procedure

2.2.3.1.1 Model precision

Originally, the LTS_{Diff} algorithm (WALKINSHAW; BOGDANOV, 2013) has been proposed to identify structural differences in models reverse-engineered by learning algorithms, such as the L^* (ANGLUIN, 1987), L^*_M (SHAHBAZ; GROZ, 2009), or RPNI (HIGUERA, 2010; WIECZOREK, 2017). This structural difference is categorized in terms of a confusion matrix (SOKOLOVA; LAPALME, 2009). In this sense, the confusion matrix for calculating model precision is composed of the following sets: the set of true-positives TP refers to the common transitions, the set of false-positives FP refers to the added transitions, and the set of false-negative FN refers to the removed transitions.

In Table 4, we show the confusion matrix used to compute the structural difference between two FSMs M_r and M_u, i.e., the target and learned models, respectively.

Based on these four sets, performance metrics, such as Precision, Recall, and F-measure, can be computed for any model learning algorithm (WALKINSHAW; BOGDANOV, 2013). Thus, the *Precision* metric quantifies the proportion of transitions in D_u that are also in D_r, and the *Recall* metric tells the proportion of transitions in D_r that are also in D_u. In Table 5, we show the formula used to calculate the aforementioned performance metrics.

		Target M_u	
		in D_u	not in D_u
Reference M_r	in D_r	$TP = D_r \setminus Rem$	$FN = Rem$
	not in D_r	$FP = Add$	$TN = \emptyset$

Table 4 – Confusion matrix to compute model learning performance metrics

Source: Adapted from Walkinshaw and Bogdanov (2013).

Measure	Formula	Description
Precision	$\frac{\lvert TP \rvert}{\lvert TP \cup FP \rvert}$	Proportion of transitions from M_u that are in M_r
Recall	$\frac{\lvert TP \rvert}{\lvert TP \cup FN \rvert}$	Proportion of transitions from M_r that are in M_u
F-Measure	$\frac{2 \times Precision \times Recall}{Precision + Recall}$	Harmonic mean between Precision and Recall

Table 5 – Performance metrics for comparing FSMs

Source: Adapted from Walkinshaw and Bogdanov (2013).

2.3 Software product lines

Technology companies, such as ABB, Boeing, Philips, and Siemens, have been facing an increasing demand for mass production and customization of hardware and software products (POHL; BÖCKLE; VAN DER LINDEN, 2005). To cope with this, they have been investing in establishing common platforms to build families of products. These platforms are often produced using software development approaches, such as software product line engineering (CLEMENTS; NORTHROP, 2001).

According to Pohl, Böckle and van der Linden (2005), the software product line engineering (SPLE) framework aims at supporting the development of multiple software applications (called *products*) from a common and managed set of *assets* in a systematic way. Unlike traditional software systems, which are tailored for a specific use, SPLs are developed for reuse and with reuse of *features* (CLEMENTS; NORTHROP, 2001). Thus, products are not created anew but derived from reusable assets (VAN DER LINDEN; SCHMID; ROMMES, 2007).

Let F be the set of features of an SPL, where a feature consists of increments to product functionalities (BENAVIDES; SEGURA; RUIZ-CORTÉS, 2010). A product p is defined by a subset of features $p \subseteq F$ from a feature model FM (KANG *et al.*, 1990). A feature model FM captures structural information and dependencies about common and variant features of an SPL as a hierarchically arranged set of features.

Features are interconnected by four kinds of relationships: *Mandatory*, if a child feature is included in all products in which its parent appears; *Optional*, if a child is optionally included;

Alternative, when only one child feature can be selected; and *Or*, when one or more of features can be included. Let the set of features in a feature model be F, the powerset $\mathscr{P}(F)$ of all feature combinations is constrained to a subset of valid products $P \subseteq \mathscr{P}(F)$ that satisfy its relationships. To illustrate these concepts, let the Arcade Game Maker SPL be our running example.

Example 2.3.1. (The Arcade Game Maker SPL) The Arcade Game Maker (AGM) SPL has three alternative features (i.e., `Brickle`, `Pong`, and `Bowling`) and one optional feature (i.e., `Save`). Figure 7 shows the AGM feature model. The feature model in Figure 7 has six valid product configurations, among which three satisfy the feature constraint ¬S.

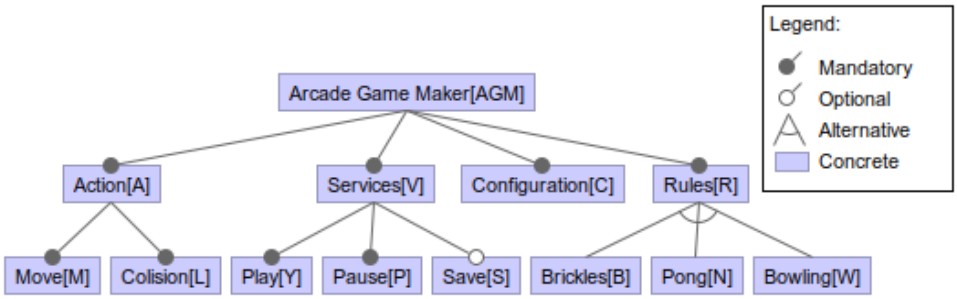


Figure 7 – The AGM feature model

Feature constraints are propositional logic formulae that interpret the elements from F in terms of propositional variables. SAT solvers (BERRE; PARRAIN, 2010) are often used to detect valid feature models, feature combinations, core features (i.e., features that are part of all products) and redundancies (BENAVIDES; SEGURA; RUIZ-CORTÉS, 2010). We denote by $B(F)$ the set of all feature constraints. The subset $\Lambda \subseteq B(F)$ defines all valid product configurations of an SPL. We interchangeably refer to products as sets of features and propositions.

The configuration $\rho \in B(F)$ of a product $p \in P$ is a feature constraint that expresses the conjunction of all features included in p and the conjunction of negated features absent from it, i.e., $\rho = (\bigwedge_{f \in p} f) \wedge (\bigwedge_{f \notin p} \neg f)$. Given a feature constraint $\chi \in B(F)$, a configuration $\rho \in \Lambda$ satisfies χ, denoted by $\rho \vDash \chi$, iff the feature constraint $\rho \wedge \chi$ is *true*. Given two feature constraints ω_a and ω_b from a feature model FM, and $\Lambda_a, \Lambda_b \subseteq \Lambda$ satisfying ω_a and ω_b, respectively, we say that ω_a and ω_b are equivalent under FM if $\Lambda_a = \Lambda_b$.

2.3.1 Analysis strategies for SPLs

The modeling and analysis (e.g., validation and verification) of SPLs are known to be demanding and cumbersome as one has to guarantee that features work as intended regardless of particular combinations. According to Thüm *et al.* (2014a), there are five categories of specification and analysis strategies for SPLs: product-based, domain-independent, family-wide, feature-based, and family-based. In Table 6, we summarize the main characteristics, disadvantages, and challenges of each analysis strategy.

Table 6 – Description of analysis strategies for SPLs

Strategy	Specification characteristics	Disadvantages and Challenges
Product-based	One specification model for every valid product of an SPL	Scales only for small SPLs, and involves redundant effort
Domain-independent	Specification independent but valid across SPLs	Only describes properties common across SPLs
Family-wide	One specification model that holds for all products of an SPL	Cannot express particular behavior inherent to some specific products
Feature-based	Specify the behavior of isolated features instead of products	There is no explicit reference to other features
Family-based	Specify properties of individual features and feature combinations	Traditional methods cannot be used as they are, maintenance of artifacts

Source: Adapted from Thüm *et al.* (2014a).

The two extremes of these specification strategies, i.e., product-based and family-based analysis, are discussed in the following sections.

2.3.1.1 *Product-based analysis*

In product-based strategies, each valid product of an SPL is specified and analyzed individually using traditional analysis techniques, such as standard MBT (UTTING; PRETSCHNER; LEGEARD, 2012) or model checking (BAIER; KATOEN, 2008). Product-based analysis strategies are often classified as *optimized* (or *sample-based*) if it evaluates a subset of valid products; or *unoptimized* if it analyzes all valid products in an exhaustive, i.e., brute-force, fashion.

Although theoretically possible, unoptimized strategies are often impractical due to the exponential number of valid products within an SPL, and inefficient due to redundant computations performed over shared assets (THÜM *et al.*, 2014a). Despite these issues, exhaustive analysis is often used as a *baseline* for other strategies (VARSHOSAZ *et al.*, 2018). In the next sections, we present two testing criteria that can be used as product-based strategies for efficiently and effectively analyzing product-lines.

Combinatorial Interaction Testing

As we have previously discussed, the number of possible product configurations usually grows exponentially with the number of features. Hence, the exhaustive analysis of SPLs is often impractical, especially for large product-lines (THÜM *et al.*, 2014a). To address these problems, product sampling techniques that provide subsets of valid configurations have been used to cover the behavior of SPLs and cater for possible feature interactions (APEL *et al.*, 2013). Therefore, they should reveal most of the faults in all other products (PERROUIN *et al.*, 2010).

According to Varshosaz *et al.* (2018), product sampling often relies on feature models (KANG *et al.*, 1990) and SAT solvers (BERRE; PARRAIN, 2010) to distinguish valid from invalid configurations (BENAVIDES; SEGURA; RUIZ-CORTÉS, 2010). To support product sampling, there are techniques using genetic algorithms (ENSAN; BAGHERI; GAŠEVIĆ, 2012; LOPEZ-HERREJON *et al.*, 2014) and T-wise coverage (PERROUIN *et al.*, 2010).

Combinatorial interaction testing (CIT) aims at using interaction coverage as testing criteria to sample product configurations for testing (KUHN; KACKER; LEI, 2013). It is based on the observation that most faults emerge by means of interactions between a small number of features (KUHN; WALLACE; GALLO, 2004). For interactions between any t features of SPLs, CIT is also referred to as T-wise testing (PERROUIN *et al.*, 2010).

The T-wise coverage criterion, defined below, aims at supporting the sampling process from the set of all possible combinations of selected and unselected features. These combinations (or interactions) are called a t-set.

Definition 2.3.1. (Valid t-set) A valid t-set is a set of features $\{\pm f_1, \pm f_2, ..., \pm f_t\}$ satisfying the constraints defined by the feature model FM over the set of features F, where $t < |F|$, $+f_i$ indicates a selected feature i and $-f_i$ an unselected one. A T-set is *invalid* if it does not satisfy the constraints of FM.

Definition 2.3.2. (T-wise Coverage) The t-wise coverage of a set of test configurations $TCS = \{PC_1, PC_2, \ldots, PC_m\}$ is the ratio $T_t = \frac{|\cup_{i=1}^{m} T_{t,PC_i}|}{|T_{t,FM}|}$, where T_{t,PC_i} is the set of t-sets included within the configuration PC_i, $T_{t,FM}$ is the set of all the possible t-sets of the FM, and $|A|$ denotes the cardinality of the set A.

Configuration similarity

Studies in software testing have shown that similar test cases tend to have equivalent fault detection capabilities. Therefore, no additional gain should be expected when these are simultaneously executed (CARTAXO; MACHADO; NETO, 2011). To mitigate these issues, similarity has been used as test selection, minimization and prioritization criterion (YOO; HARMAN, 2012) for access control testing (BERTOLINO *et al.*, 2015; DAMASCENO; MASIERO; SIMAO, 2018) and SPL testing (HENARD *et al.*, 2014; AL-HAJJAJI *et al.*, 2017).

In configuration similarity, a similarity metric describes a similarity relation between two configurations as a numeric value. Similarity metrics often range from zero, if product configurations are totally distinct, to one, if they implement the same set of features.

The Hamming distance is a well-known measure (DEZA; DEZA, 2013) that has been used to calculate the similarity between product configurations (AL-HAJJAJI *et al.*, 2017). It is represented as the normalized number of common selected and unselected features for two configurations:

Definition 2.3.3. (Configuration similarity) The configuration similarity between two product configurations p_i, p_j from a feature model FM with the set of features F is defined as follows:

$$confSim(p_i, p_j, F) = \frac{|p_i \cap p_j| + |(F \setminus p_i) \cap (F \setminus p_j)|}{|F|} \tag{2.1}$$

In the $confSim()$ metric, $|p_i \cap p_j|$ denotes the number of common features selected between p_i and p_j and $|(F \setminus p_i) \cap (F \setminus p_j)|$ represents the number of common unselected features between them. These two values are normalized by the total number of features $|F|$.

2.3.1.2 Family-based analysis

Family-based analysis operates on domain-level artifacts that incorporate knowledge about valid feature combinations. Thus, not every individual product is analyzed, and redundant computations are minimized or avoided (THÜM *et al.*, 2014a). The performance of family-based strategies is mainly influenced by the number and size of features and the amount of sharing during combinations, while product-based strategies often face costs proportional to the number of valid configurations (BRABRAND *et al.*, 2012). In the next section, we introduce the Featured Finite-state Machine notation (FRAGAL; SIMAO; MOUSAVI, 2017; FRAGAL, 2017) for expressing the behavior of individual features and feature combinations of product-lines.

2.3.1.2.1 Featured Finite-state Machines

The Featured Finite-state Machine is a model that extends FSMs (GILL, 1962) with feature constraints to express the behavior of SPLs.

Definition 2.3.4 (Featured Finite-state Machine)**.** An FFSM is a septuple $\langle F, \Lambda, C, c_0, Y, O, \Gamma \rangle$, where: F is a finite set of features, Λ is the set of product configurations, $C \subseteq S \times B(F)$ is a finite set of conditional states, where S is a finite set of state labels, $B(F)$ is the set of all feature constraints, and C satisfies the condition:

$$\forall (s, \phi) \in C,\ \exists \rho \in \Lambda | \rho \vDash \phi \tag{2.2}$$

$c_0 = (s_0, true) \in C$ is the initial conditional state of the FFSM, $Y \subseteq I \times B(F)$ is a finite set of conditional inputs, where I is the finite set of input symbols, O is the finite set of output symbols, and $\Gamma \subseteq C \times Y \times O \times C$ is the set of conditional transitions satisfying the condition:

$$\forall ((s, \phi), (x, \phi''), o, (s', \phi')) \in \Gamma, \exists \rho \in \Lambda | \rho \vDash (\phi \wedge \phi' \wedge \phi'') \tag{2.3}$$

Example 2.3.2. (The Arcade Game Maker FFSM) Figure 8 depicts an FFSM for the AGM SPL. The conditional state `Save Game[S]` and all conditional transitions reaching or leaving it are implemented by all products implementing feature S. The FSM in Figure 5 is an example of FSM derived using the configuration $\rho = (AGM \wedge A \wedge M \wedge L \wedge V \wedge Y \wedge P \wedge W \wedge \neg S \wedge \neg B \wedge \neg N)$.

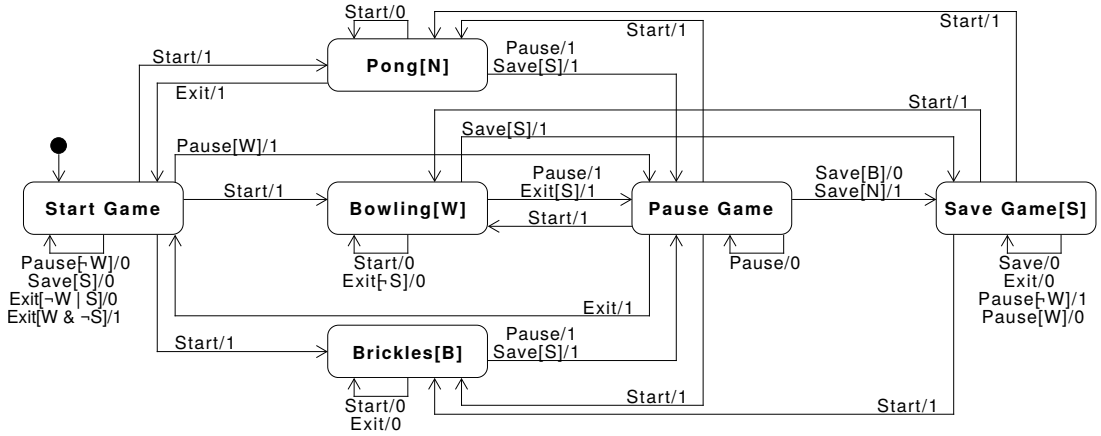


Figure 8 – FFSM of the AGM

Source: Adapted from Fragal (2017).

Conditions (2.2) and (2.3) ensure that all conditional states and transitions are present in at least one valid product of the SPL. A conditional state $c = (s, \phi) \in C$ is alternatively denoted by $s[\phi]$. A conditional transition $(c, (x, \phi), o, c')$ from conditional state c to c' with conditional input x and output o is alternatively denoted $c \xrightarrow{x[\phi]/o} c'$. The logical operators `and`, `or` and `not` are denoted by the symbols &, |, and ¬, respectively. An omitted condition means that the condition is *true*.

Given an FFSM $FF = \langle F, \Lambda, C, c_0, Y, O, \Gamma \rangle$ and a configuration $\rho \in \Lambda$, the model derivation operator Δ_ρ (FRAGAL; SIMAO; MOUSAVI, 2017) derives a product FSM $\Delta_\rho = (S, s_0, I, O, D)$, where: $S = \{s | (s, \phi) \in C \wedge (\phi \vDash \rho)\}$ is the set of states; $s_0 = s, c_0 = (s, \phi) \in C$ is the initial state; and $D = \{(s, x, o, s') | ((s, \phi), (x, \phi'), o, (s', \phi'')) \in \Gamma \wedge \rho \vDash (\phi \wedge \phi' \wedge \phi'')\}$ is the set of transitions.

To make FFSMs suitable for MBT, Fragal, Simao and Mousavi (2017) have proposed a validation technique to check whether an FFSM satisfies the basic properties of FSMs, i.e., determinism, completeness, initially connectedness, and minimality, at the product-line level. Added to this, they have also shown that their proposed SPL-level validation is sound, i.e., if an FFSM satisfies these properties, so do all the FSM products that can be derived from it.

Additionally, the FFSM notation has been extended to generate configurable test suites that can be pruned using feature constraints for groups of product configuration (FRAGAL *et al.*, 2018). The readability of FFSMs also has been improved by grouping up conditional states and transitions into hierarchical entities (FRAGAL; SIMAO; MOUSAVI, 2019). Therefore, the FFSM notation has the prospect of serving as a suitable basis for SPL modeling and testing.

2.4 Final remarks

In this section, we have presented the theoretical background and literature that supports this PhD thesis. We have discussed the concept of software testing, with an emphasis on FSM-

based testing, and how the lack of model maintenance may affect this activity. We have introduced model learning, how it can support software analysis and testing, and what improvements can be made to speed up the learning process by means of adaptive learning. We have discussed how techniques for model comparison be used to quantify the performance of automata learning algorithms. Finally, we have introduced software product lines and the FFSM notation for modeling behavioral variability in product-lines.

Along with this chapter, we have highlighted a few research gaps in the software engineering literature that we have explored as motivation to the development of this PhD thesis. In the next chapters, we go deeply into each of these gaps and address them by improving the state-of-the-art of model learning algorithms.

CHAPTER

3

ADAPTIVE MODEL LEARNING FOR EVOLVING SYSTEMS

According to Binder (1999), software analysis is necessarily a *model-based* activity, whether models are in engineers' minds, informally sketched on papers, or formally denoted as explicit models (BROY *et al.*, 2005). Explicit models are useful assets in the development of software-intensive systems as they pave the way to automated analysis (GURBUZ; TEKINERDOGAN, 2018). Nevertheless, as requirements change *over time*, model maintenance is often neglected due to its cost, and models are rendered outdated (WALKINSHAW, 2013).

To tackle this issue, active model learning (ANGLUIN, 1987) has been increasingly popular to derive behavioral models automatically (IRFAN; ORIAT; GROZ, 2013; MARIANI; PEZZÈ; ZUDDAS, 2015; VAANDRAGER, 2017). Model learning aims at building a hypothesis $\mathscr{H}$ about the "language" (i.e., reactive behavior) of a system under learning (SUL) by iteratively providing input sequences and observing outputs (VAANDRAGER, 2017). To formulate a hypothesis $\mathscr{H}$ about the SUL's behavior, a learning algorithm searches for pairs of *transfer* and *separating* sequences to reach and distinguish states, respectively. Nevertheless, traditional model learning algorithms often do not scale to real systems (DUHAIBY *et al.*, 2018). Additionally, the constant changes along the life-cycle (CHAKI *et al.*, 2008) may require frequent re-work, i.e., *learning from scratch*.

Adaptive learning (GROCE; PELED; YANNAKAKIS, 2002) attempts to speed up model learning by reusing the knowledge from existing models of evolving system. Studies (CHAKI *et al.*, 2008; GROCE; PELED; YANNAKAKIS, 2002; WINDMÜLLER *et al.*, 2013; HUISTRA; MEIJER; VAN DE POL, 2018) have shown that pre-existing models can steer learning by reusing the sequences applied in the past queries and hence, reduce the cost for model re-inference. However, after several changes, old separating sequences may lead to *deprecated* queries; and former transfer sequences may become *redundant*. These are known to be major threats to efficient adaptive model learning (HUISTRA; MEIJER; VAN DE POL, 2018).

In this chapter, we report our findings for our first research objective to investigate optimization strategies to re-learn models of fixed quality using adaptive learning and mitigate threats to its performance led by *deprecated* and *redundant* sequences. To address these issues, we introduce the `partial-Dynamic` $\mathtt{L}^*_\mathtt{M}$ ($\partial\mathtt{L}^*_\mathtt{M}$) algorithm, an adaptive technique where the cost for learning models from systems evolving *over time* is reduced employing an *on-the-fly* search for *redundant* and *deprecated* sequences in reused observation tables.

We present an empirical study comparing our technique against three state-of-the-art adaptive algorithms and evaluating how these sequences can hamper adaptive model learning. Through this chapter, we address the following research questions:

(RQ1.1) Is the `partial-Dynamic` $\mathtt{L}^*_\mathtt{M}$ algorithm more efficient than the state-of-the-art adaptive learning techniques?

(RQ1.2) Is the effectiveness of adaptive learning strongly affected by the temporal distance between versions?

In this study, we relied on the LearnLib framework (ISBERNER; HOWAR; STEFFEN, 2015) to evaluate adaptive model learning algorithms. We have used as subject systems 18 Mealy machines from a large-scale analysis of the OpenSSL project (DE RUITER, 2016), an open-source and commercial-grade cryptographic toolkit (OpenSSL Foundation, Inc., 2018a).

To answer RQ1.1, we have used the number of membership queries (`MQ`) to learn models with a fixed level of accuracy as a measure of effectiveness. To answer RQ1.2, we have used the temporal distance between versions, denoted by the difference between their release dates as measure of software evolution. We have opted for this metric because structural changes (e.g., changed transitions) in a black-box setting are unknown and behavioral metrics (e.g., percentage of failed tests) may mislead minor modifications closer to initial states compared to significant changes.

To date, there is a lack of studies on the pros and cons of adaptive learning and how much extra (and irrelevant) queries it may pose if low-quality models are reused. This chapter provides essential insights to fill in this gap and improve upon the state-of-the-art adaptive model learning algorithms. The `partial-Dynamic` $\mathtt{L}^*_\mathtt{M}$ algorithm has been published as a regular paper at the 15th International Conference on integrated Formal Methods (iFM) held in Bergen, Norway (DAMASCENO; MOUSAVI; SIMAO, 2019b).

The remaining of this chapter is organized as follows: in section 3.1, we introduce the `partial-Dynamic` $\mathtt{L}^*_\mathtt{M}$ algorithm for adaptive learning. in section 3.2, we present our experimental design to evaluate the effects of reusing sequences and its correlation with model quality. in sections 3.3, we analyze the results obtained from our experiments. in section 3.4, we enumerate the threats to validity of this investigation. in Section 3.5, we discuss our findings. in section 3.6, we draw our final remarks about our findings.

3.1 The partial-Dynamic L_M^* algorithm

In a software development life cycle, significant changes can lead to adaptive model learning algorithms posing irrelevant queries composed by *redundant* and *deprecated* sequences. As *redundant* queries, we mean those queries applied in the past to previous versions and composed by *transfer sequences* formerly able to access different states in the SUL. As *deprecated* queries, we mean those queries posed to previous versions and composed by *separating sequences* that no longer distinguish states in the SUL. Thus, the calculation of "useful" subsets of sequences should help to mitigate risks for adaptive learning algorithms (HUISTRA; MEIJER; VAN DE POL, 2018).

To find these "useful" subsets of sequences, we designed the partial-Dynamic L_M^* (∂L_M^*) algorithm. The ∂L_M^* algorithm is a novel adaptive learning technique that improves upon the state-of-the-art by exploring observation tables on-the-fly to avoid irrelevant queries rather than indiscriminately re-asking MQs.

The term *partial* applies as the ∂L_M^* algorithm explores reused observation tables *on-the-fly* using depth-first search (DFS) to find redundant prefixes, and breadth-first search (BFS) to find deprecated suffixes. Hence, it attempts to find "useful" parts of the observation table leading to an equivalent state coverage and separating capability by reusing fewer queries. A schematic overview of the ∂L_M^* algorithm is depicted in Figure 9.

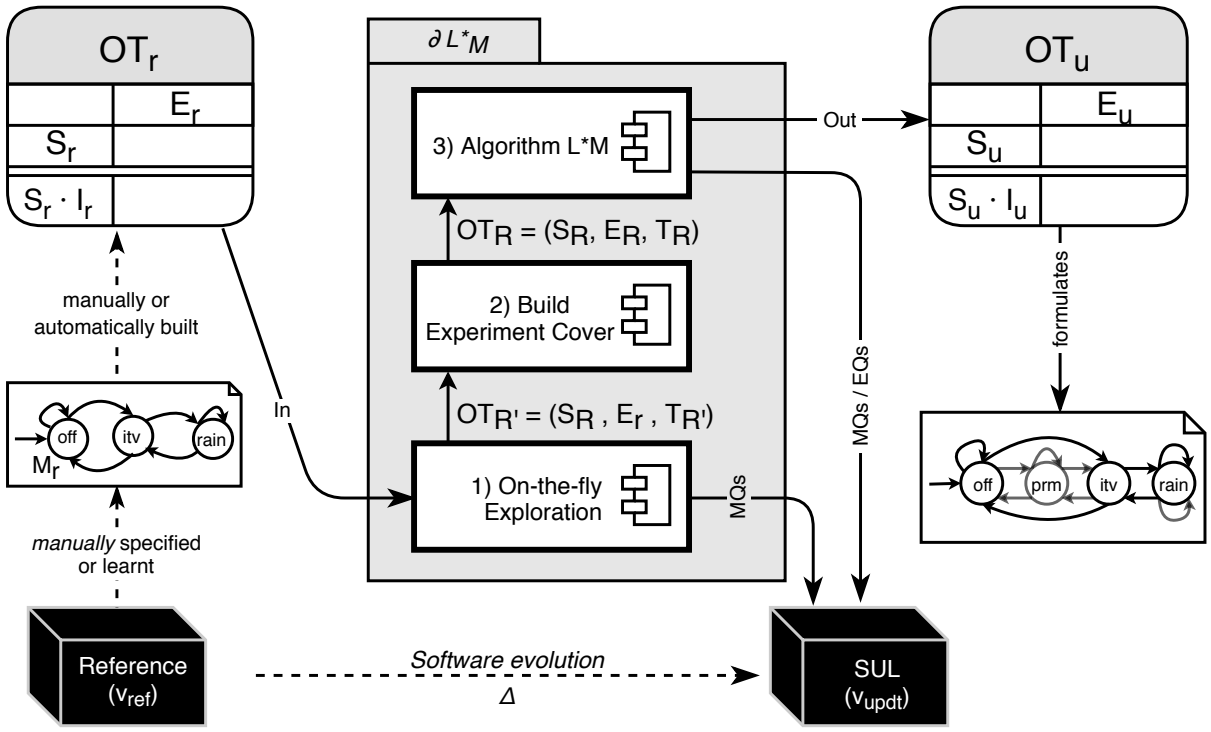


Figure 9 – partial-Dynamic L_M^* - Schematic overview

Source: Elaborated by the author.

As indicated in Figure 9, the $\partial \mathrm{L}^*_{\mathtt{M}}$ algorithm comprises three sequential steps that we discuss in the next sections. To illustrate the steps comprising our algorithm, we use an updated version of the windscreen wiper system as a running example of an evolving system.

Example 3.1.1. (Updated windscreen wiper) Let the FSMs in Figure 3 be the reference version v_{ref}, and the one in Figure 10 be the updated version v_{updt} of a hypothetical evolving system. Added states and transitions are depicted in dotted lines.

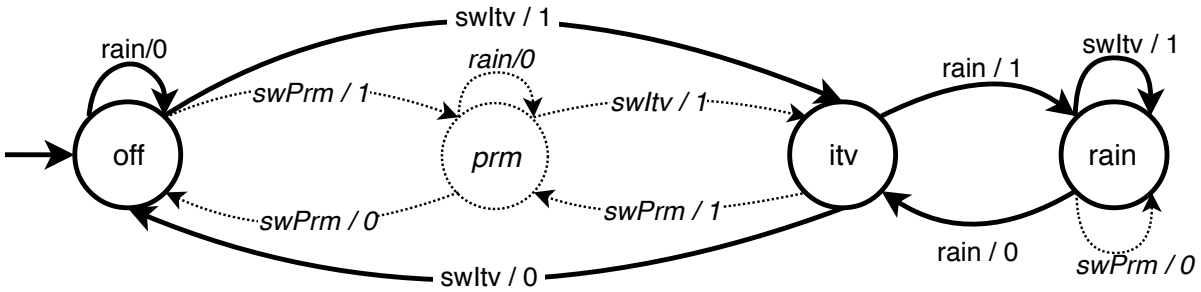


Figure 10 – Windscreen wiper with permanent movement

Source: Elaborated by the author.

We refer to their representation as minimal-complete FSMs and counterpart elements as $M_r = \langle Q_r, q_{0_r}, I_r, O_r, \delta_r, \lambda_r \rangle$ and $M_u = \langle Q_u, q_{0_u}, I_u, O_u, \delta_u, \lambda_u \rangle$.

3.1.1 On-the-fly exploration of the reused table

Let S_r and E_r be the sets of prefixes and suffixes respectively from an observation table $\mathtt{OT}_r = (S_r, E_r, T_r)$. Since we do not know the internal implementation of v_{updt}, $\mathtt{OT}_r$ may be outdated, redundant prefixes may emerge from S_r and hence, they may no longer reach the same distinct states (HUISTRA; MEIJER; VAN DE POL, 2018). Thus, an updated observation table $\mathtt{OT}_{R'} = (S_R, E_r, T_{R'})$ has to be created by restoring the agreement of $\mathtt{OT}_r$ to v_{updt} employing $\mathtt{MQ}$s.

To achieve this goal, we explore the tree representation of $S_r \cdot I_u$ using depth-first search (DFS) to pose $\mathtt{MQ}$s on-the-fly and build an updated table $\mathtt{OT}_{R'}$. Thus, we mitigate the risks for indiscriminately re-asking queries that may not lead to useful observations.

During the *on-the-fly* construction of this tree representation, we identify prefixes leading to states already discovered by E_r and discard their extensions to find a well-formed cover subset $S_R \subseteq S_r$. In this tree representation, paths leading from the root to nodes represent elements from the subset $S_r \cdot I_u$. Nodes are annotated using rows of the updated observation table $\mathtt{OT}_{R'}$.

Example 3.1.2. (Well-formed cover subset) In Figure 11, we depict a fragment of the tree representation for an $\mathtt{OT}_{R'}$ built from an outdated observation table where the set of prefixes is $S_r = \{\varepsilon, swItv, swItv \cdot rain, swItv \cdot rain \cdot rain, wItv \cdot rain \cdot rain \cdot swItv, rain\}$ and the set of suffixes is $E_r = \{rain, swItv, swPrm, rain \cdot rain\}$.

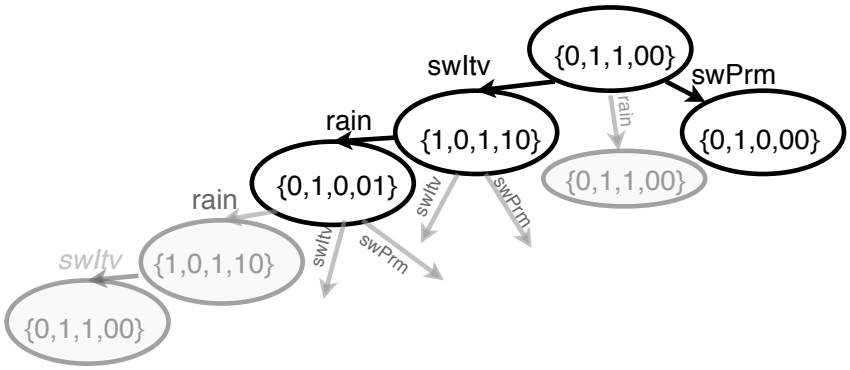


Figure 11 – Well-formed cover subset S_R generated from S_r

Source: Elaborated by the author.

Black arrows denote the well-formed cover subset. Discarded prefixes are depicted in gray. The cost to find this well-formed cover subset is 40 MQs, in contrast to 76 MQs, to completely restore the agreement of $\mathtt{OT}_r$ to v_{updt}.

3.1.2 Building an experiment cover tree

Once we find a subset $S_R \subseteq S_r$, we use the upper part from $\mathtt{OT}_{R'} = (S_R, E_r, T_{R'})$ to search for an experiment cover subset. An experiment cover subset $E_R \subseteq E_r$ is obtained by picking one representative element from each subset of equivalent suffixes (CHAKI *et al.*, 2008).

To address this task, we propose an optimization technique that runs a breadth-first search (BFS) on a tree representation of E_r, referred to as an *experiment cover tree*. The construction of this tree is performed similarly to homing trees (BROY *et al.*, 2005).

Definition 3.1.1. (Experiment cover tree) Consider an updated observation table $\mathtt{OT}_{R'} = (S_R, E_r, T_{R'})$ and an input domain I_u; an experiment cover tree is a rooted tree that satisfies the following constraints:

1. The *root* node is labeled as $lbl(root) = Col(S_R, \varepsilon)$;
2. Each edge e is labeled with one suffix $e \in E_r$;
3. Each node n linked to parent n_p by edge e is labeled as $lbl(n) = Col(lbl(n_p), e)$;
4. Non-leaf nodes n have outgoing edges for all suffixes $E_r \backslash E_{R(n)}$, where $E_{R(n)}$ is the set of suffixes labeling the edges in the path from *root* to n;
5. A node n is leaf iff
   a) for all $B_i \in lbl(n)$, $|B_i| = 1$; otherwise
   b) there is a lower node n_l where $lbl(n) = lbl(n_l)$.

The *experiment cover tree* is built using BFS and, if a node d satisfying *5a* is found, the suffixes labeling the path from *root* to d is returned as the experiment cover subset E_R. Otherwise, we traverse the experiment cover tree and the first node d found with maximum separating capability is selected as the E_R, i.e., $max(|Col(S_R, E_{R(d)})|)$. Neither MQs nor EQs are posed.

Example 3.1.3. (Experiment cover) Figure 12 shows a fragment of the experiment cover tree generated from the subset S_R in Figure 11 and the set of suffixes $E_r = \{rain, swItv, swPrm, rain \cdot rain\}$. The subset E_R is highlighted in black.

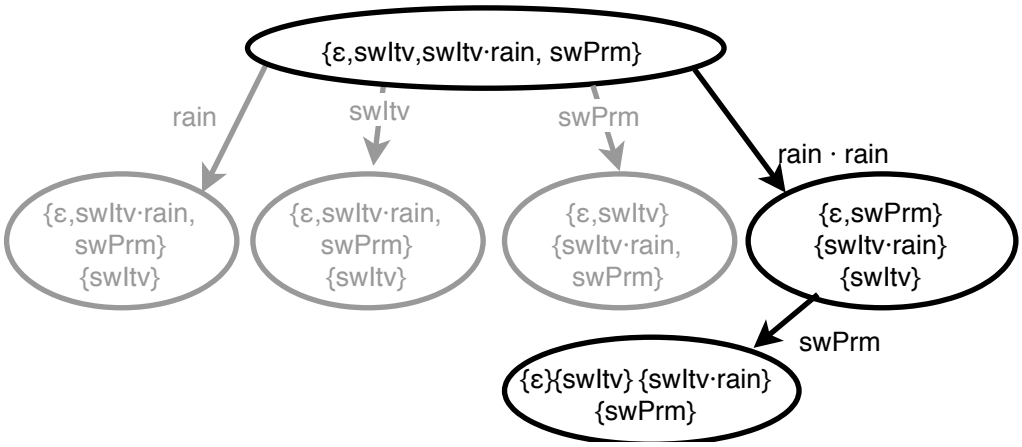


Figure 12 – Experiment cover tree

Source: Elaborated by the author.

3.1.3 Running $\mathtt{L}^*_{\mathtt{M}}$ using the outcomes of $\partial\mathtt{L}^*_{\mathtt{M}}$

At this stage, our approach has discarded redundant prefixes and deprecated suffixes. Hence, we initialize the $\mathtt{L}^*_{\mathtt{M}}$ algorithm using the well-formed and experiment cover subsets, rather than $S_u = \{\varepsilon\}$ and $E_u = I$. As results, we expect to build an observation table with higher state discovery capability in the first iteration utilizing fewer queries, especially if both versions v_{ref} and v_{updt} are not drastically different.

Example 3.1.4. In Table 7, we summarize the number of MQs and EQs posed to the SULs depicted in Figures 3 and 10 by the following five model learning algorithms: $\mathtt{L}^*_{\mathtt{M}}$ (SHAHBAZ; GROZ, 2009); our $\partial\mathtt{L}^*_{\mathtt{M}}$; an adaptive approach, referred to as Adp, where $\mathtt{L}^*_{\mathtt{M}}$ starts with suffixes from a previous version (HUISTRA; MEIJER; VAN DE POL, 2018); and two straightforward implementations of the Dynamic $\mathtt{L}^*$ (CHAKI *et al.*, 2008) algorithm for Mealy machines, referred to as $\mathtt{DL}^*_{\mathtt{M}}$ and $\mathtt{DL}^*_{\mathtt{M}+}$. The latter differs by restoring the properties of closedness and consistency to avoid the loss of prefixes $s_1, s_2 \in S_R$, where $s_1 \cong s_2$ and $\exists(i,e) \in (I_u, E_r), T_{R'}(s_1 \cdot i, e) \neq T_{R'}(s_2 \cdot i, e)$. In this example, the $\partial\mathtt{L}^*_{\mathtt{M}}$ algorithm posed fewer MQs than the other methods.

Table 7 – Reuse approaches and numbers of queries

Algorithm	Reuse		Restore properties	SUL	OT	Number of	
	Prefixes	Suffixes				MQs	EQs
L^*_M	-	-	-	v_{ref}	-	18	2
				v_{updt}	-	48	2
Adp	No	Complete	No	v_{updt}	$\mathtt{OT}_r$	48	1
$\mathtt{DL}^*_\mathtt{M}$	Indiscriminate		No	v_{updt}	$\mathtt{OT}_r$	76	2
$\mathtt{DL}^*_{\mathtt{M}+}$			Yes	v_{updt}	$\mathtt{OT}_r$	81	1
$\partial\mathtt{L}^*_\mathtt{M}$	On-the-fly		No	v_{updt}	$\mathtt{OT}_r$	43	1

Source: Elaborated by the author.

3.2 Empirical evaluation

According to Huistra, Meijer and van de Pol (2018), the more the states of an SUL have been changed, the lower is the number of suffixes with good quality. Therefore, a higher number of irrelevant queries should be expected to be posed by the state-of-the-art adaptive learning algorithms, especially when older versions are reused.

3.2.1 Research questions

To evaluate adaptive learning techniques in different settings, we have extended the LearnLib framework (ISBERNER; HOWAR; STEFFEN, 2015) with the algorithms from Table 7. Thus, we investigated if our $\partial\mathtt{L}^*_\mathtt{M}$ algorithm was more efficient than three state-of-the-art adaptive algorithms (RQ1.1) and the impact of temporal distance in their effectiveness (RQ1.2).

In Table 8, we show the hypotheses formulated about the influence of the temporal distance between versions on the number of queries, denoted by $\Delta\mathtt{T}$; and the average difference between the number of MQs and EQs posed by adaptive learning algorithms and the $\mathtt{L}^*_\mathtt{M}$ for traditional model learning, respectively denoted by $\mu\mathtt{MQ}$ and $\mu\mathtt{EQ}$.

Table 8 – Hypotheses - Learning to Reuse Models

Measure	Hypotheses	Description
$\mu\mathtt{MQ}$	$H_0^{\mu\mathtt{MQ}}$	The $\partial\mathtt{L}^*_\mathtt{M}$ requires an equivalent $\mu\mathtt{MQ}$
	$H_1^{\mu\mathtt{MQ}}$	The $\partial\mathtt{L}^*_\mathtt{M}$ requires a higher $\mu\mathtt{MQ}$
	$H_2^{\mu\mathtt{MQ}}$	The $\partial\mathtt{L}^*_\mathtt{M}$ requires a lower $\mu\mathtt{MQ}$
$\mu\mathtt{EQ}$	$H_0^{\mu\mathtt{EQ}}$	The $\partial\mathtt{L}^*_\mathtt{M}$ requires an equivalent $\mu\mathtt{EQ}$
	$H_1^{\mu\mathtt{EQ}}$	The $\partial\mathtt{L}^*_\mathtt{M}$ requires a higher $\mu\mathtt{EQ}$
	$H_2^{\mu\mathtt{EQ}}$	The $\partial\mathtt{L}^*_\mathtt{M}$ requires a lower $\mu\mathtt{EQ}$
$\Delta\mathtt{T}$	$H_0^{\Delta\mathtt{T}}$	The $\partial\mathtt{L}^*_\mathtt{M}$ is influenced by the temporal distance
	$H_1^{\Delta\mathtt{T}}$	The $\partial\mathtt{L}^*_\mathtt{M}$ is not influenced by the temporal distance

Source: Elaborated by the author.

As a measure of software evolution, we have used the temporal distance between versions in terms of their release dates. We have opted for this metric as structural changes (e.g., changed transitions) in black-box settings are unknown, and behavioral metrics (e.g., percentage of failed test cases) may mislead minor modifications closer to initial states compared to significant changes (WALKINSHAW; BOGDANOV, 2013). As a measure of effectiveness, we have used the number of MQs and EQs posed by each adaptive learning algorithm compared to traditional learning using the L^*_M algorithm (SHAHBAZ; GROZ, 2009) in terms of numbers of resets and input symbols.

For each scenario $\langle v_l, \mathtt{OT_r} \rangle$ where v_l is an SUL and $\mathtt{OT_r}$ is an observation table built from a reference version v_r, we used the Mann-Whitney-Wilcoxon (MWW) to check if there was *statistical significance* ($p < 0.01$) between the difference of numbers of queries posed by each adaptive algorithms. To measure the *scientific significance* (KAMPENES *et al.*, 2007) and the probability of one algorithm outperforming another (ARCURI; BRIAND, 2011), we used the Vargha-Delaney's $\hat{A}$ effect size (VARGHA; DELANEY, 2000; WOHLIN *et al.*, 2012). If $\hat{A}_{c,t} < 0.5$, then the treatment t poses more queries than the control c. If $\hat{A}_{c,t} = 0.5$, they are equivalent. To categorize the magnitude, we used the intervals between $\hat{A}_{c,t}$ and 0.5 (HESS; KROMREY, 2004; TORCHIANO, 2017): $0 \leq negligible < 0.147 \leq small < 0.33 \leq medium < 0.474 \leq large \leq 0.5$. Finally, we used Pearson's correlation coefficient to evaluate the relationship between the temporal distance between versions $\langle v_l, v_r \rangle$ to the numbers of MQs and EQs.

3.2.2 Subject systems

As our evolving system, we have used 18 Mealy machines learned in a large scale analysis of several versions of OpenSSL (DE RUITER, 2016), an open-source and commercial-grade cryptographic toolkit (OpenSSL Foundation, Inc., 2018a). In Figure 13, we depict the versioning schema for over 14 years of development branches from the server-side of OpenSSL.

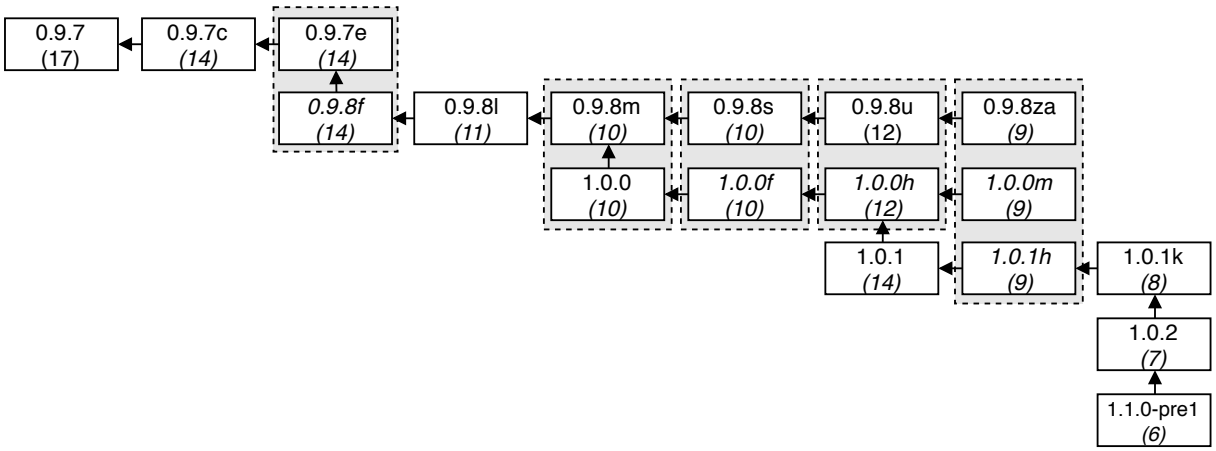


Figure 13 – OpenSSL server-side: 18 FSMs versions used as SUL

Source: Adapted from de Ruiter (2016).

In this versioning scheme, SULs are denoted by white boxes with arrows pointing out to their previous release in the branch, the number of implemented states are shown in parentheses, and dashed areas indicate groups of versions with behavioral overlaps (i.e., equivalent FSMs). These are important information to quantify the distance between versions.

3.2.3 Experimental design

Let $\langle v_l, \mathtt{OT}_r \rangle$ be a learning scenario where v_l is the SUL, and $\mathtt{OT}_r$ is an observation table built from a reference version v_r. For all 18 versions of the OpenSSL project and their precedents, we measured the difference between the numbers of resets and symbols in MQs and EQs posed by each adaptive algorithm and $\mathtt{L}^*_\mathtt{M}$ and their temporal distance in years. Positive difference values indicate that *adaptive learning posed more queries* than traditional learning.

For each reused reference version v_r, we generated 500 observation tables $\mathtt{OT}_\mathtt{r} = (S_r, E_r, T_r)$ composed by prefix-closed state cover sets S_r created using randomized DFS, and $E_r = I_u \cup W_r$, where W_r is a W set for v_r; and calculated the μMQs and μEQs. For processing CE, we used the *Suffix1by1* (IRFAN; ORIAT; GROZ, 2010), and the CLOSE_FIRST strategy to close tables (LearnLib, 2018). In order to build EQs, we used the Wp method for conformance testing (FUJIWARA *et al.*, 1991) with an upper bound equal to $m = 2$.

3.2.4 Experiment artifacts

For the sake of reproducibility and repeatability, we have made publicly available a lab package with a variety of artifacts (e.g., source code, test scripts, FSMs). The lab package is at the link <https://github.com/damascenodiego/DynamicLstarM/releases/tag/iFM2019>. This repository is organized as a Java project that can be opened using the Java Development Kit version 1.8 (ORACLE, 2014) and the Eclipse IDE (ECLIPSE, 2019).

The coding artifacts of this project are in the `br.usp.icmc.labes.mealyInference` package. In this package, we have implemented java classes to support a command-line interface (CLI) developed to automate our experiments execution. The `mealyInference/mylearn.jar` jar file is a compiled version of the `partial-Dynamic` $\mathtt{L}^*_\mathtt{M}$ algorithm. This software has been developed using the libraries LearnLib and AutomataLib (RAFFELT; STEFFEN, 2006).

In folder `mealyInference/Experiments/2019_02_iFM`, we have included our experiment results and experiment scripts. The raw outputs of our experiments are available in the file `mealyInference/Experiments/2019_02_iFM/run_2019021.zip`. To replicate our experiment, there are the `mealyInference/Experiments/2019_02_iFM/exp_nordsec16.sh` and `mealyInference/Experiments/2019_02_iFM/util.R` scripts. FSM models of the subject systems are found in folder *mealyInference/Benchmark/Nordsec16*.

3.3 Analysis of results

In this section, we analyze the μMQs and μEQs posed by the adaptive algorithms and their relationship to the temporal distance within $\langle v_l, v_r \rangle$. We calculated the average differences between the numbers of symbols and resets in MQs and EQs posed by each adaptive algorithm and learning from scratch using the $\mathtt{L}^*_{\mathtt{M}}$ algorithm.

By analyzing the release dates of all versions (OpenSSL Foundation, Inc., 2018b), we identified a strong positive correlation ($r = 0.72$) between the temporal distance and variation in numbers of states implemented in each version. In Figure 14, we show the correlation between the temporal distance and variation in numbers of states implemented between all pairs of OpenSSL versions.

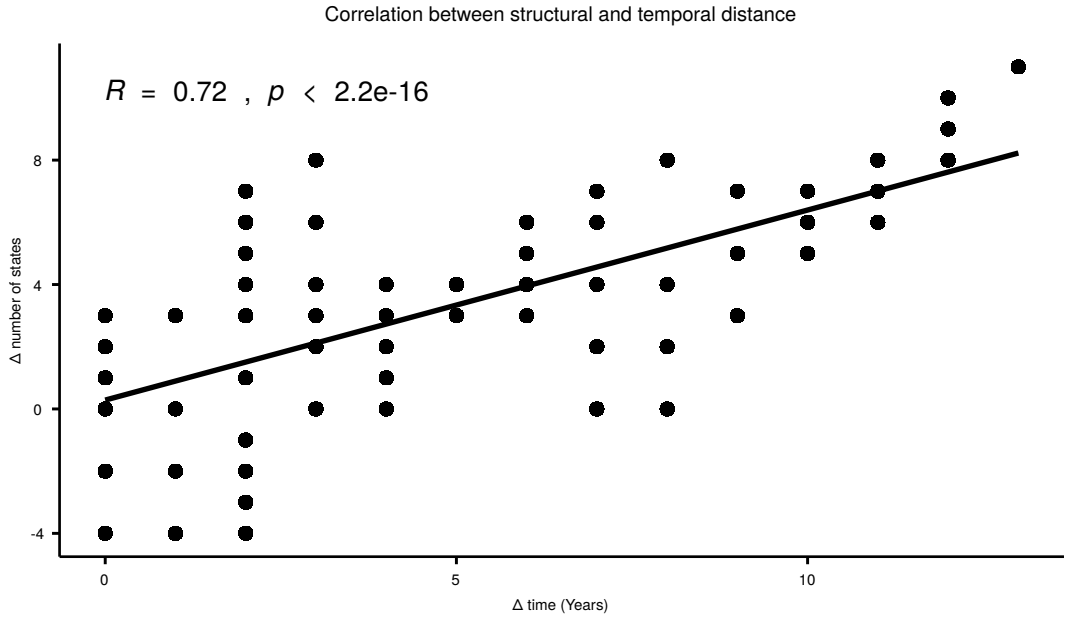


Figure 14 – Pearson's correlation between variation in number of states and temporal distance

Source: Research data.

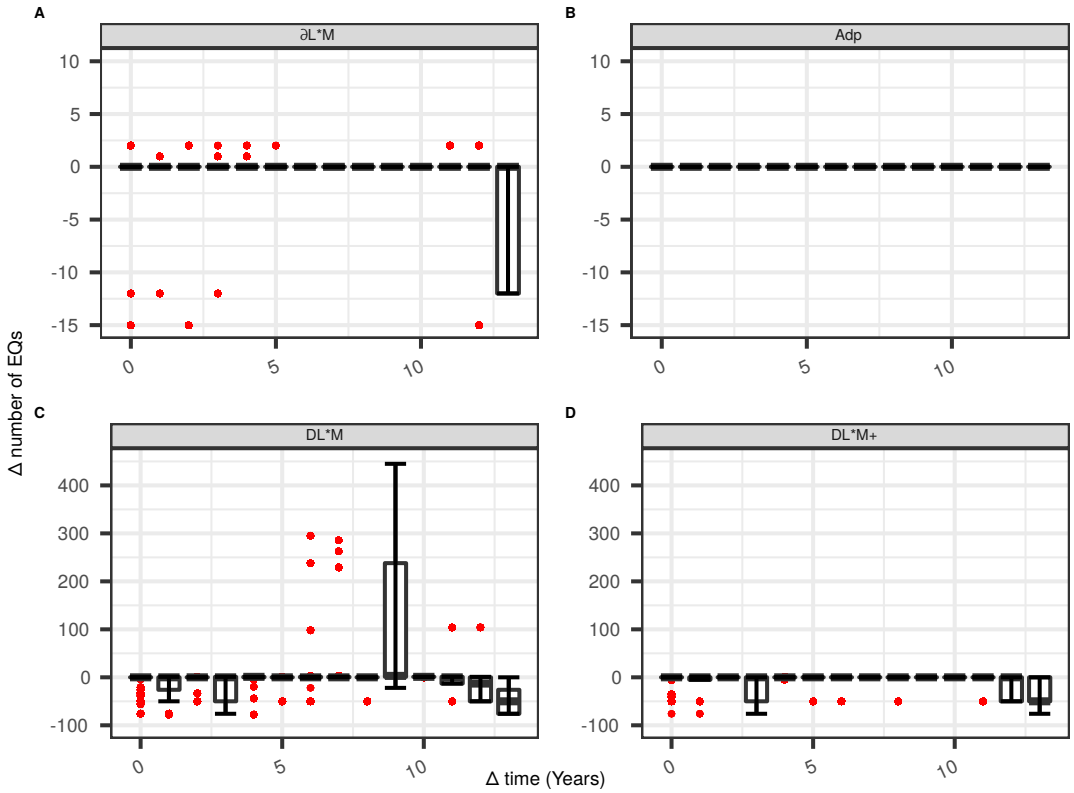
These results corroborate de Ruiter (2016) findings that the OpenSSL project has improved over time, and recent versions became more succinct in terms of their size, i.e., variation of number of states. These findings indicated that the OpenSSL project represents an interesting case study for evolving systems that can pose challenges to adaptive learning algorithms, such as the reuse of older versions may impact the performance of adaptive learning, realistic size and structure, the possibility of infinite behavior, and the existence of states with similar or identical behavior in different versions.

3.3.1 Average difference of EQs

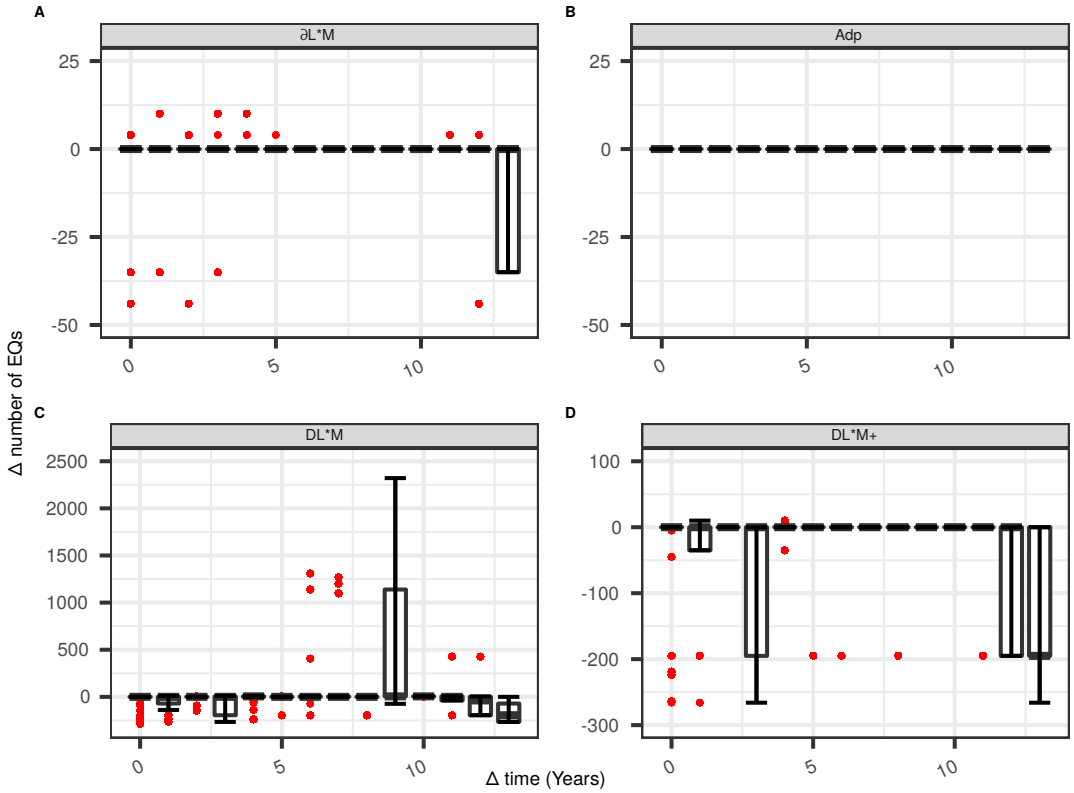
In Figures 15a and 15b, we depict boxplots for the numbers of resets and symbols in the μEQs as a function of the temporal distance between $\langle v_l, v_r \rangle$. To keep the figure uncluttered, we calculated the boxplots for time windows of one year. Outliers are depicted as red dots.

Figure 15 – Boxplots of the μEQs posed by adaptive and traditional learning

(a) Number of resets in the μEQs



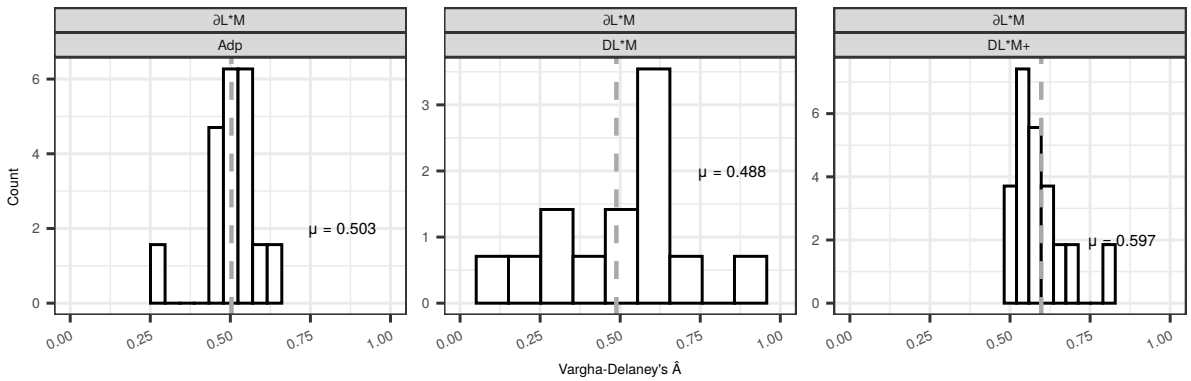
(b) Number of symbols in the μEQs



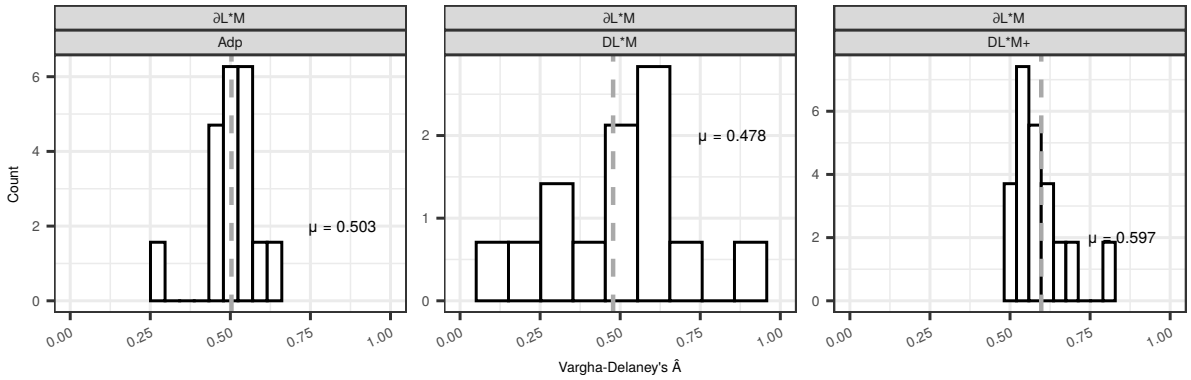
Source: Research data.

Figure 16 – Histograms of the effect sizes for EQs posed by the adaptive algorithms

(a) Number of resets



(b) Number of symbols



Source: Research data.

By analyzing the μEQs and the learned observation tables, we found that the set of suffixes composed by I_u has been a good set of separating sequences. The I_u set has enabled us to distinguish most of the states implemented in the OpenSSL versions. Thus, as our reused observation tables included I_u, the resulting μEQs happened to be quite similar. Hence, the whiskers in our boxplots were presented very close to the averages.

In Figures 16a and 16b, we show histograms to the effect size for the numbers of resets and symbols in EQs, where $\partial \mathrm{L}^*_{\mathrm{M}}$ is the control method. The MWW test indicated a statistically significant difference ($p < 0.01$) between $\partial \mathrm{L}^*_{\mathrm{M}}$ and the other adaptive algorithms; however, as the histograms indicate, the effect sizes were mostly categorized as *negligible*.

Additionally, Pearson's correlation coefficient indicated a *very weak* to *no correlation* between μEQs and temporal distance. The number of rounds was approximately the same, i.e., *one* round for all versions with less than 14 states and *two to five* for all other versions. These findings support the hypothesis $H_0^{\mu\mathrm{EQ}}$ that our adaptive learning algorithm $\partial \mathrm{L}^*_{\mathrm{M}}$ required an μEQs similar to traditional learning.

3.3.2 Average difference of MQs

In Figures 17a and 17b, we depict boxplots to numbers of resets and symbols in the μMQs as a function of the temporal distance between v_l and v_r. To keep the figure uncluttered, we calculated the boxplots for time windows of one year. Outliers are depicted as red dots.

By analyzing the μMQs, we found that Adp posed around 50 additional MQs when versions older than four years were reused. For the $\mathtt{DL}^*_{\mathtt{M}}$ and $\mathtt{DL}^*_{\mathtt{M}+}$ algorithms, we have found an increment on the μMQs of up to 800 extra queries. Our results indicated significant increments in the total number of MQs when the temporal distance was maximum (i.e., 14 years).

For the existing adaptive learning algorithms, we found a *strong* to *very strong* correlation between μMQs and the temporal distance within $\langle v_l, v_r \rangle$. Thus, our findings corroborate to Huistra, Meijer and van de Pol (2018) where the quality of the reused sequences has been a factor that can undermine the performance of adaptive learning techniques. Consequently, the existing adaptive learning algorithms posed a large number of MQs composed of *redundant* prefixes and *deprecated* suffixes.

Differently from the state-of-the-art adaptive learning algorithms, our $\partial\mathtt{L}^*_{\mathtt{M}}$ algorithm turned out to be more robust than the other adaptive techniques. In Figures 18a and 18b, we show histograms for the effect sizes of the numbers of resets and symbols posed by the $\partial\mathtt{L}^*_{\mathtt{M}}$ algorithm compared to the three other adaptive techniques.

For the $\partial\mathtt{L}^*_{\mathtt{M}}$ algorithm compared to the other algorithms, we found a significant difference ($p < 0.01$) on the number of MQs, and a *weak positive* correlation between the μMQs and temporal distance, with effect sizes mostly categorized as *large*. Thus, our results have favored the hypothesis $H_2^{\mu\mathtt{MQ}}$ that identified the $\partial\mathtt{L}^*_{\mathtt{M}}$ as the algorithm showing the lowest μEQs compared to the other adaptive techniques. Thus, it has answered the **RQ1** by placing our on-the-fly technique as more efficient than the existing adaptive algorithms in terms of MQs.

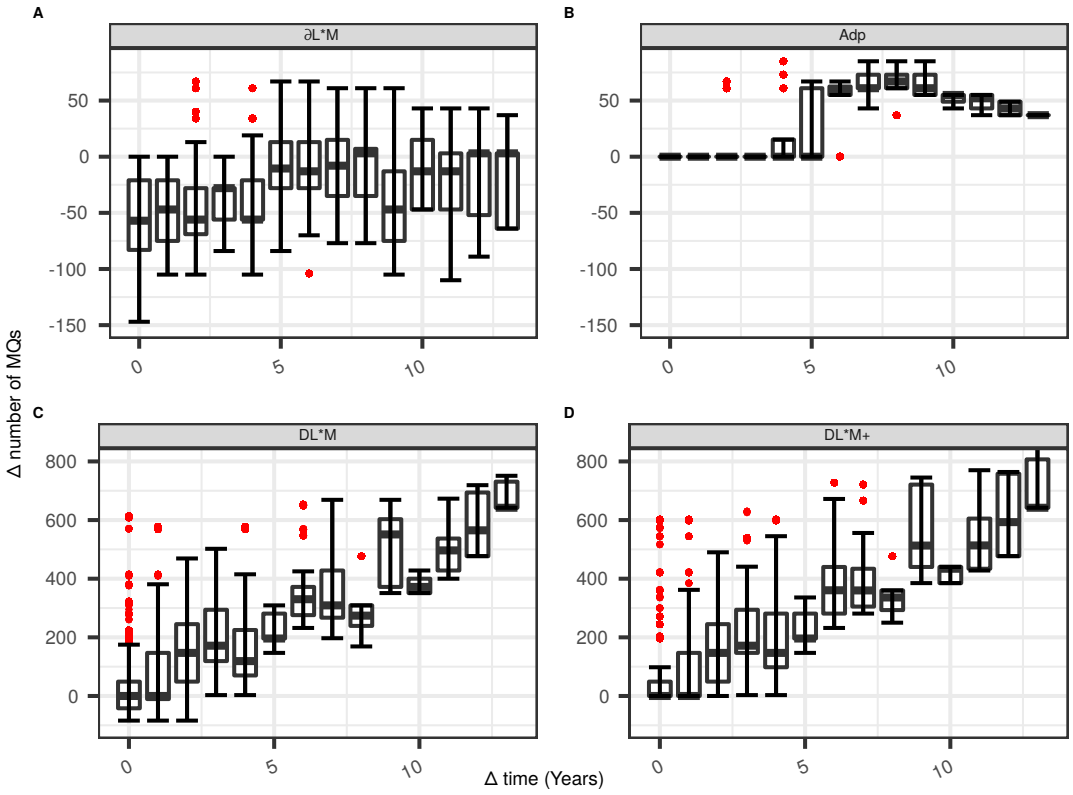
3.3.3 Benefits of adaptive learning vs. temporal distance

For all the adaptive learning algorithms, Pearson's correlation coefficients have indicated *very weak* positive correlation (≤ 0.20) between the number of EQs and temporal distance. These findings corroborate the mostly constant μEQ seen in Figures 15a and 15b.

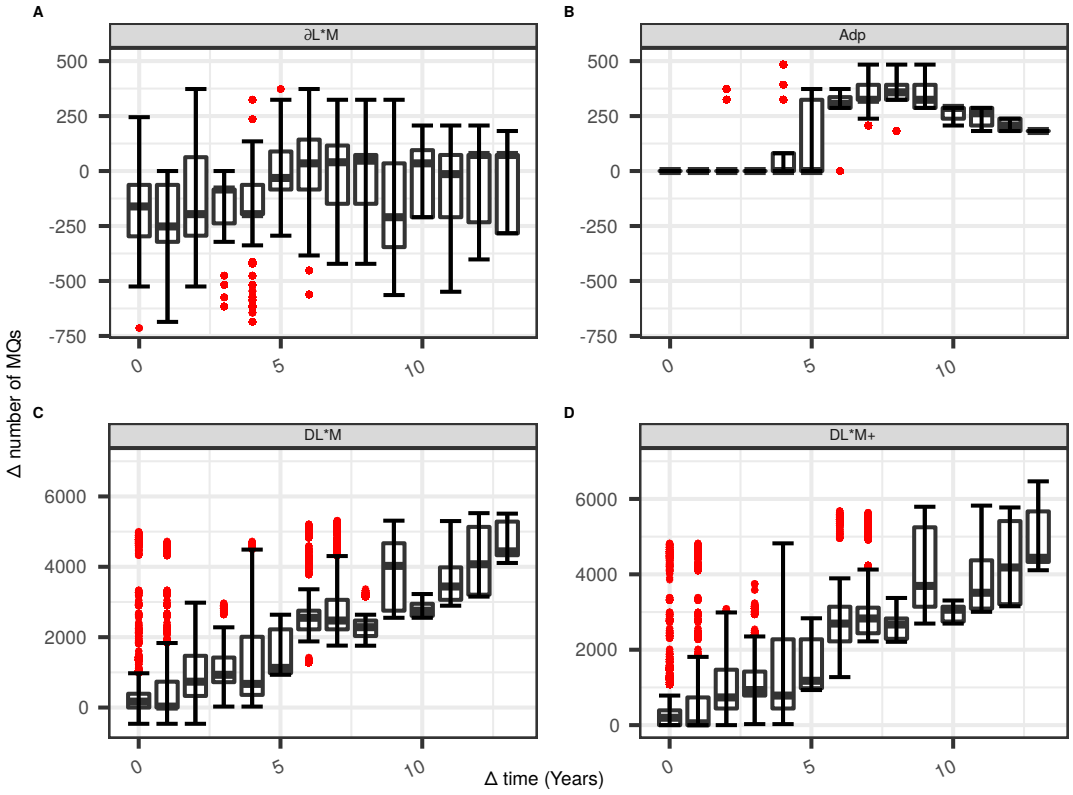
For the algorithms Adp, $\mathtt{DL}^*_{\mathtt{M}}$ and $\mathtt{DL}^*_{\mathtt{M}+}$, we observed the Pearson's coefficients indicated strong positive correlations (0.73, 0.78 and 0.80, respectively) between the number of MQs and temporal distance. Alternatively, for our $\partial\mathtt{L}^*_{\mathtt{M}}$ algorithm, we observed weak positive correlation (0.33) between the number of MQs and temporal distance. Thus, we confirm the hypothesis $H_1^{\Delta\mathtt{T}}$ that the $\partial\mathtt{L}^*_{\mathtt{M}}$ algorithm is not influenced by the temporal distance between versions and affirmatively answer **RQ2** by showing that the effectiveness of existing adaptive learning techniques is indeed affected by the temporal distance between versions.

Figure 17 – Boxplots of the μMQs posed by adaptive and traditional learning

(a) Number of resets



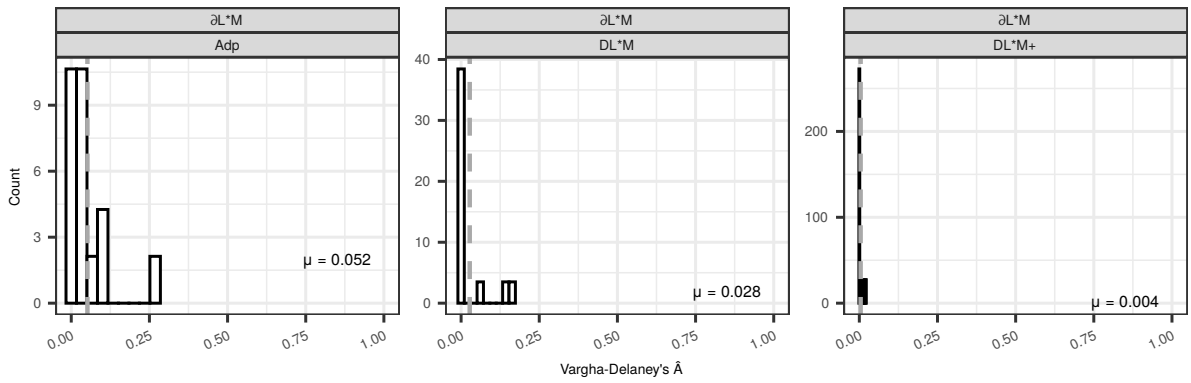
(b) Number of symbols



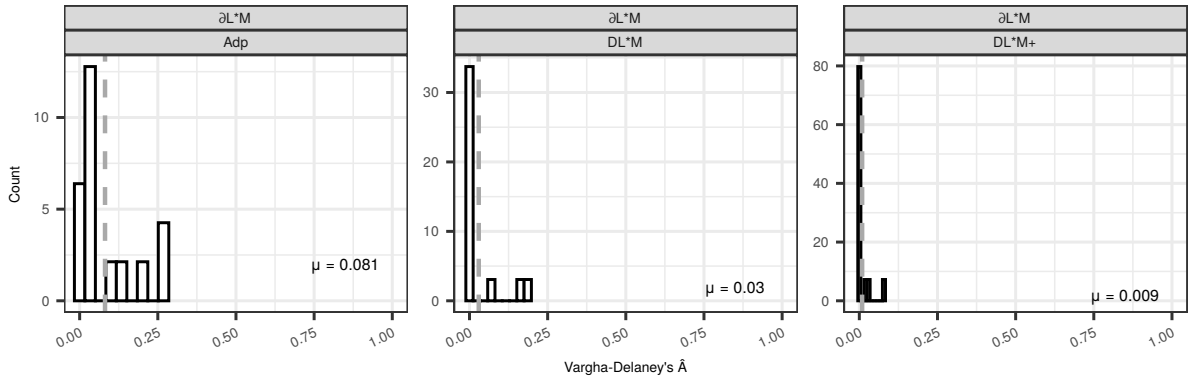
Source: Research data.

Figure 18 – Histograms of the effect sizes for MQs posed by the adaptive algorithms

(a) Number of resets



(b) Number of symbols



Source: Research data.

3.4 Threats to validity

Internal validity: Threats to internal validity concern with the influences that can affect the causal relationship between the treatment and outcomes. One element that forms a threat to internal validity is the temporal distance between versions as a measure of software evolution. We found a strong positive correlation ($r = 0.72$) between temporal distance and difference in the numbers of states of the underlying FSMs. Thus, the temporal distance shall be a reasonable measure, at least for this particular case. Failed test-cases may not be good measures as they do not reflect the points of failure in the SUL semantics, e.g., minor changes close to initial states compared to major changes on the language.

External validity: Threats to external validity concern with generalization. To guarantee the reliability of our experiment results, we relied on the LearnLib framework (ISBERNER; HOWAR; STEFFEN, 2015) to implement adaptive learning algorithms. Our study is based on FSM models representing multiple versions of the OpenSSL toolkit, and it poses a threat to external validity. However, since the OpenSSL project has realistic size and structure, we believe that our results are generalizable to other reactive systems (SMEENK *et al.*, 2015).

3.5 Discussion

What are the implications for researchers? Our study complements the scientific literature by introducing an approach inspired by the limitations of the state-of-the-art adaptive algorithms (GROCE; PELED; YANNAKAKIS, 2002; CHAKI *et al.*, 2008; WINDMÜLLER *et al.*, 2013; HUISTRA; MEIJER; VAN DE POL, 2018). Additionally, as the proposal of new methods motivates empirical studies with existing algorithms, we have empirically compared our technique against these adaptive algorithms. Finally, our research artifacts have been open-sourced so that interested people can study our strategies and gain insights for improvements.

What are the implications for practitioners? Recent studies have shown model learning as suitable for addressing a wide range of software engineering tasks (AICHERNIG *et al.*, 2018). However, as promising in theory, they are also slippery and inefficient in real systems (DUHAIBY *et al.*, 2018). Therefore, we believe that our study can pave the way towards more efficient applications of model learning in subject systems with realistic size and structure. Our experimental artifacts have been implemented using the LearnLib project (LearnLib, 2017), a state-of-the-art Java framework for automata learning.

What types of models may it work/not work? Currently, our approach has been designed for SULs that incorporate changes in the behavioral level. Thus, the addition or deletion of symbols in the input domain may hurdle our strategy's performance. Two possible issues that may be faced are an excessive pruning of the prefix tree (i.e., during the on-the-fly exploration) or the experiment cover tree (i.e., during the second step of our algorithm). As a result, our algorithm's performance may become comparable to traditional learning, where the initial setup starts from the empty sequence. A possible improvement for this limitation could be the reuse of transfer or synchronizing sequences from non-initial states in the learning process.

3.6 Final remarks

Real systems pass through changes along their life-cycle. Thus, as we often do not know how states may have changed, behavioral models tend to become outdated, incomplete, or even deprecated. To deal with these issues, recent studies have proposed the application of active model learning to derive behavioral models automatically.

Adaptive model learning is a variant of active learning which attempts to speed up model inference by reusing transfer and separating sequences from previously learned observation tables. However, software evolution tends to undermine the performance of state-of-the-art adaptive learning. Hence, it may pose *redundant* and *deprecated* sequences. To date, there is a lack of studies about the pros and cons of adaptive model learning and how much extra effort it may pose when low-quality models are reused. We fill this research gap by performing an empirical analysis of adaptive methods and comparing them against our novel technique.

We have introduced a novel adaptive algorithm that explores observation tables *on-the-fly* to avoid irrelevant MQs, called ∂L^*_M. Using 18 versions of the OpenSSL toolkit, we showed that state-of-the-art adaptive algorithms mostly show a strong positive correlation between the number of MQs and temporal distance between the reused and learned versions.

Alternatively, our ∂L^*_M algorithm presented a weak positive correlation between temporal distance and MQs. Thus, our algorithm turned out to be less sensitive to software evolution and more efficient than current approaches for adaptive learning. Also, our ∂L^*_M algorithm posed fewer MQs compared to three state-of-the-art adaptive learning algorithms.

In this study, we move towards addressing adaptive learning algorithms' performance issues for evolving systems and mitigating the reuse of *redundant* and *deprecated* sequences. Additionally, we fill the research gap of studies comparing adaptive learning methods using subject systems with realistic size and structure, the possibility of infinite behavior, and the existence of states with similar or identical behavior in different versions.

The `partial-Dynamic` L^*_M algorithm has been published as a regular paper at the 15th International Conference on integrated Formal Methods (iFM) that was held in Bergen, Norway (DAMASCENO; MOUSAVI; SIMAO, 2019b). For the sake of reproducibility and repeatability, we have open-sourced our code artifacts, FSMs, and test scripts in a lab package available on GitHub at the page <https://damascenodiego.github.io/DynamicLstarM/>.

This chapter presents one of the earliest investigations of this PhD research. Early designs of this idea have emerged from meetings with prof. Dr. Mohammad Reza Mousavi and prof. Dr. Adenilso Simao and pilot experiments performed in the first half of 2017. This research has been carried out using computational resources of the Center for Mathematical Sciences Applied to Industry (CeMEAI) funded by FAPESP (grant 2013/07375-0). Additionally, we are grateful to the VALidation and Verification (VALVE) research group from the University of Leicester (England), and the Software Engineering Lab (LabES) from the University of Sao Paulo (Brazil) for their insightful suggestions and technological infrastructure.

CHAPTER

4

FAMILY MODEL LEARNING FOR PRODUCT LINES

Analyzing software product lines (SPL) on a product-based basis is very demanding and cumbersome, due to the number of possible products (THÜM *et al.*, 2014a), crosscutting features (SCHAEFER *et al.*, 2012), and the need to cater for possible feature interactions (APEL *et al.*, 2013). Hence, family-based approaches have been developed to facilitate the analysis of SPLs without going individually through each and every product (BENDUHN *et al.*, 2015; CLASSEN *et al.*, 2013; BEOHAR; MOUSAVI, 2014; FRAGAL; SIMAO; MOUSAVI, 2017). Such family-based approaches pave the way for efficient model-based analysis of SPL and typically involve a variability-aware behavioral specification referred to as a *family* model (THÜM *et al.*, 2014a) or *150% model* (BEUCHE; SCHULZE; DUVIGNEAU, 2016).

Family models are annotated with feature constraints to express the combination of features involved in product-specific concerned parts of the model (THÜM *et al.*, 2014a). Thus, using SAT solvers (BERRE; PARRAIN, 2010), family models are amenable to family model-based testing (UTTING; PRETSCHNER; LEGEARD, 2012) and family model checking (BAIER; KATOEN, 2008) where redundant analysis are mitigated. Additionally, the cost of family-based analysis is mainly determined by the number and size of features and the amount of feature sharing, rather than the number of valid products (THÜM *et al.*, 2014a).

Model-based techniques designed explicitly to family models have enabled efficient test case generation (ATLEE *et al.*, 2015; BEOHAR; MOUSAVI, 2016; FRAGAL; SIMAO; MOUSAVI, 2017) and model checking (SABOURI; KHOSRAVI, 2013; TER BEEK; DE VINK; WILLEMSE, 2017). Nevertheless, the creation and maintenance of test models are known to be difficult, time consuming and error-prone (UTTING; PRETSCHNER; LEGEARD, 2012). Additionally, the traceability between the family- and variability models can be complex due to crosscutting features (SCHAEFER *et al.*, 2012). Thus, as new requirements emerge and products evolve, the lack of maintenance may render outdated models (WALKINSHAW, 2013).

Motivated by these issues, in this chapter we discuss how the creation and maintenance of family models can be performed using techniques for automata learning (WALKINSHAW; BOGDANOV, 2013), feature model analysis (BENAVIDES; SEGURA; RUIZ-CORTÉS, 2010), and product sampling (VARSHOSAZ *et al.*, 2018). To address these tasks, we have introduced the $FFSM_{Diff}$ algorithm, a technique to learn family models by comparing and merging product models into a succinct virtual representation that incorporates variability constraints to express product-specific behaviors. The representation is said *virtual* as it can also describe optional or mutually exclusive behavior (BENAVIDES; SEGURA; RUIZ-CORTÉS, 2010).

Our algorithm extends a technique for analyzing the effectiveness of automata learning algorithms (WALKINSHAW; BOGDANOV, 2013) to the domain of product lines by incorporating variability to indicate product-specific behaviors as feature constraints. Hence, we compare product FSMs to map feature constraints to their states and transitions, and merge them into a featured finite state machine (FFSM) (FRAGAL; SIMAO; MOUSAVI, 2017; FRAGAL *et al.*, 2018). An FFSM is a family-based formalism for unifying Mealy Machines (BROY *et al.*, 2005; GILL, 1962) specifying valid products of an SPL into a single representation by annotating states and transitions with feature constraints (FRAGAL *et al.*, 2018).

We have also employed the T-wise product sampling criteria (VARSHOSAZ *et al.*, 2018) as a configuration query (`CQ`) oracle in family model learning. A `CQ` oracle is an entity that we have designed to search for subsets of individual products to be learned and, hence, reduce the costs for building precise family models. We incorporate similarity measures developed by Walkinshaw and Bogdanov (2013) to quantify the precision of family models learned from sampled sets of valid product configurations.

To evaluate our approach, we have used an extensive benchmark of abstract representation of SPLs (CLASSEN, 2010; SAMIH *et al.*, 2014; FRAGAL; SIMAO; MOUSAVI, 2017). First, we have evaluated the succinctness of the learned family models concerning the size of individual product models and hand-crafted specifications. Second, we have analyzed the correlation between degree of reuse and succinctness of learned models. Finally, we have measured the precision of family models learned from products sampled by different `CQ` oracles. Thus, we have addressed the following research questions (RQ):

(RQ2.1) Is our approach effective in learning succinct family models compared to the total size of the products under learning?

(RQ2.2) Is the size of the learned family models influenced by the configuration similarity degree of the products under learning?

(RQ2.3) Is our approach effective in learning succinct family models compared to the total size of the hand-crafted models?

(RQ2.4) Is our approach effective in learning precise family models by sampling compared to exhaustive learning?

This chapter builds upon two manuscripts (DAMASCENO; MOUSAVI; SIMAO, 2019a; DAMASCENO; MOUSAVI; SIMAO, 2020) that we have recently published as a full paper at the 23rd International Systems and Software Product Line Conference held in Paris, France, and submitted for a special issue on "Configurable Systems" in the Empirical Software Engineering journal[1]. To analyze our algorithm, we have performed an extensive analysis using six abstract representations of SPL from academic benchmarks (CLASSEN, 2010; SAMIH *et al.*, 2014; FRAGAL; SIMAO; MOUSAVI, 2017), among which there is an industrial situational awareness system for helicopters flying in degraded visual environments (SAMIH *et al.*, 2014). Furthermore, we have evaluated the precision of models learned from products sampled using `CQ` oracles.

To our knowledge, this is the first study in learning variability-aware behavioral models and evaluating the performance of product sampling to learn precise family models at a reduced cost. Our approach can be helpful to domain engineering by supporting the inclusion of new application requirements, SPL re-engineering (FENSKE; THüM; SAAKE, 2013), evolution (MARQUES *et al.*, 2019), and traceability analysis (VALE *et al.*, 2017).

The remainder of this chapter is organized as follows: In Section 4.1, we introduce our $FFSM_{Diff}$ algorithm to learn FFSMs from product specifications. In Section 4.2, we present a process that incorporates product sampling for efficient family model learning. In Section 4.3, we discuss an empirical evaluation to assess the effectiveness of our techniques. In Section 4.4, we present the analysis of results obtained in our empirical evaluation. In Section 4.5, we enumerate the threats to validity that we addressed in this study. In Section 4.6, we present a discussion about our results. In Section 4.7, we close this chapter with our final remarks.

4.1 Learning family models from product specifications

Family models have been exploited as a theoretical foundation for efficient SPL analysis (SABOURI; KHOSRAVI, 2013; BEOHAR; VARSHOSAZ; MOUSAVI, 2016; TER BEEK; DE VINK; WILLEMSE, 2017). Albeit reasonably efficient, the creation of models is known to be time-consuming, and error-prone, especially if there are crosscutting features and large models (SCHAEFER *et al.*, 2012). Additionally, software maintenance is often neglected due to its cost, and, hence it may render outdated specifications (WALKINSHAW, 2013).

In this section, we introduce the $FFSM_{Diff}$ algorithm, a fully automated technique that integrates feature model analysis into the process of structural comparison of state-based models, discussed in Chapter 2; to annotate states and transitions with feature constraints to learn succinct FFSM models (FRAGAL; SIMAO; MOUSAVI, 2017; FRAGAL *et al.*, 2018) from individual product specifications of SPLs. Although our technique is discussed in terms of FFSMs, it can be extended to other family-based modeling approaches (BENDUHN *et al.*, 2015), such as featured transition systems (FTS) (CLASSEN *et al.*, 2013; BEOHAR; MOUSAVI, 2014).

[1] This journal manuscript is currently in the *Rebuttal Phase*

4.1.1 The $FFSM_{Diff}$ algorithm

The $FFSM_{Diff}$ allows (i) to learn a fresh FFSM model from two product-specific FSMs and (ii) to incorporate novel product-specific behavior into an existing FFSM model. The former task can be useful when no FFSM exists a priori for a product line and the latter when there is a new product configuration $\rho_u \notin \Lambda_r$ that is not specified into an FFSM FF_r specifying a set of configurations Λ_r from a product line.

For both cases, we assume that the feature model, the product-specific FSMs and configurations of the products under learning are known a priori. For the product FSMs, these are assumed to be previously hand-crafted, or learned using some variant of model learning (ANGLUIN, 1987; SHAHBAZ; GROZ, 2009) and satisfy the basic properties for FSM-based testing and model learning, i.e., determinism, completeness, initially connectedness, and minimality.

4.1.2 Learning a fresh FFSM from two products specifications

Let $M_r = \langle S_r, s_{0_r}, I_r, O_r, D_r, \delta_r, \lambda_r \rangle$ and $M_u = \langle S_u, s_{0_u}, I_u, O_u, D_u, \delta_u, \lambda_u \rangle$ be the FSMs of two products p_r and p_u that implement configurations $\rho_r = (\bigwedge_{f \in p_r} f) \wedge (\bigwedge_{f \notin p_r} \neg f)$ and $\rho_u = (\bigwedge_{f \in p_u} f) \wedge (\bigwedge_{f \notin p_u} \neg f)$. To learn a fresh FFSM from product-specific state machines, i.e., M_r and M_u, there are two assumptions: (i) M_r and M_u are complete, deterministic, initially connected and minimal FSMs built a priori, and (ii) their respective feature model and configurations ρ_r and ρ_u are known a priori.

We employ feature model analysis to identify and annotate conditional states and transitions identified using a technique to compare state machines (WALKINSHAW; BOGDANOV, 2013). Hence, to learn fresh FFSMs from two product-specific FSMs, we proceed as follows:

Definition 4.1.1 (FFSM learned from two configurations)**.** An FFSM learned from $\langle M_r, M_u \rangle$ is a tuple $FF = \langle F, \Lambda, C, c_0, Y, O, \Gamma \rangle$, where

- $F = (p_r \cup p_u)$ is the set of features implemented by the two products
- $\Lambda = \{\rho_r, \rho_u\}$ is the set of two configurations analyzed,
- $C \subseteq (S_r \cup S_u \cup (S_r \times S_u)) \times B(F)$ is the set of conditional states where

$$\begin{aligned} \forall s_i \in S_r, s_j \in S_u \mid (s_i, s_j) \in KPairs \cdot ((a,b), \rho_r | \rho_u) \in C, \\ \forall s_i \in S_r, \nexists s_j \in S_u \mid (s_i, s_j) \in KPairs \cdot (s_i, \rho_r) \in C, \\ \forall s_j \in S_u, \nexists s_i \in S_r \mid (s_i, s_j) \in KPairs \cdot (s_j, \rho_u) \in C \end{aligned} \tag{4.1}$$

- $c_0 = ((s_{0_r}, s_{0_u}), true) \in C$ is the initial conditional state,
- $Y \subseteq (I_r \cup I_u) \times B(F)$ is a finite set of conditional input symbols,

- $O = (O_r \cup O_u)$ is the finite set of output symbols
- $\Gamma \subseteq C \times Y \times O \times C$ is the set of conditional transitions where
  - two transitions $(s_i, x) \in D_r$ and $(s_j, x) \in D_u$ are unified in the same conditional transition if

$$\begin{aligned} \forall (s_i,x) \in D_r, (s_j,x) \in D_u \mid \lambda_r(s_i,x) = \lambda_u(s_j,x) = o, \\ \delta_r(s_i,x) = s_k, \delta_u(s_j,x) = s_l, \\ (s_i,s_j),(s_k,s_l) \in KPairs \cdot \\ ((s_i,s_j),\phi),(x,(\rho_u|\rho_r)),o,((s_k,s_l),\phi'')) \in \Gamma \end{aligned} \tag{4.2}$$

  - otherwise, for two transitions $(s_i, x) \in D_r$ and $(s_j, y) \in D_u$, there are two independent conditional transitions defined as follows:

$$\begin{aligned} \forall (s_i,x) \in D_r, (s_j,x) \in D_u \mid \lambda_r(s_i,x) = o_r, \delta_r(s_i,x) = s_k, \\ \lambda_u(s_j,y) = o_u, \delta_u(s_j,y) = s_l, \cdot \\ ((s_i,\phi_r),(x,\rho_r),o_r,(s_k,\phi_r'')) \in \Gamma, \\ ((s_j,\phi_u),(y,\rho_u),o_u,(s_l,\phi_u'')) \in \Gamma \end{aligned} \tag{4.3}$$

Condition 4.1 ensures that product states are either unified into one conditional or two distinct conditional states and, hence, annotated with the disjunction of its concerned product configurations or the individual configuration involved in a specific product instance, respectively. Condition 4.2 denotes when two transitions shall be unified due to their matching labels and conditional states. Finally, Condition 4.3 describes the case where two transitions cannot be merged, and two distinct conditional transitions shall be created for each product configuration.

Example 4.1.1 (FFSM learned from two product configurations)**.** In Figure 19, we depict a fragment of an FFSM learned from the two product FSMs shown in Figures 5 and 6. In this example, the product-specific states `Pong Game` and `Bowling Game` were merged into one conditional state `Bowling*Pong` where there is one conditional transition with input symbol `Exit` for each configuration. The feature constraint $(W \wedge \neg S \wedge \neg B \wedge \neg N)$ is an example of a simplified configuration for the product in Figure 5.

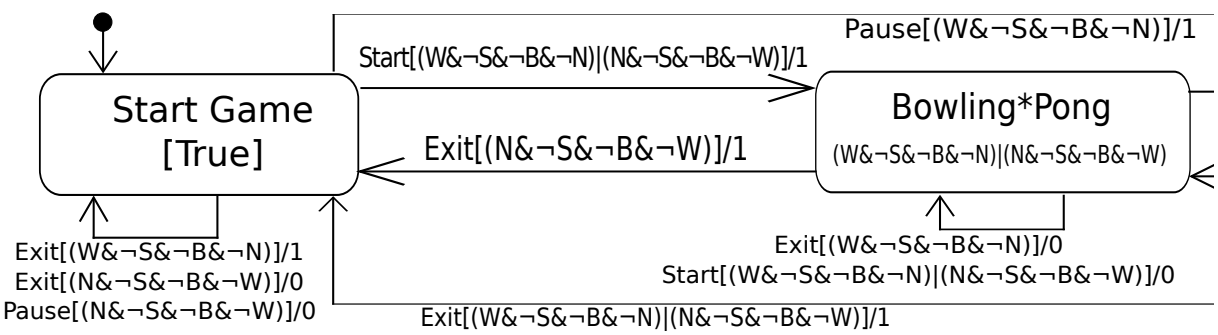


Figure 19 – Fragment of the FFSM learned for the AGM SPL

Source: Elaborated by the author.

To guarantee the mapping between the initial states of the products, we start by assuming the state pair (s_{0_r}, s_{0_u}) as one of the landmarks returned by the *identifyLandmarks*() function from Algorithm 2. Then, we begin searching for all pairs likely to be equivalent, given the threshold t and ratio r parameters. To reduce the complexity of feature constraints, we simplify product configurations by discarding core features (BENAVIDES; SEGURA; RUIZ-CORTÉS, 2010) from their associated formulae.

4.1.3 Including new product behavior into an existing FFSM

Let $FF_r = \langle F_r, \Lambda_r, C_r, c_{0_r}, Y_r, O_r, \Gamma_r \rangle$ be an FFSM model learned from a set of product configurations Λ_r. If the FFSM model FF_r does not include the behavior of a product-specific FSM $M_u = \langle S_u, s_{0_u}, I_u, O_u, D_u, \delta_u, \lambda_u \rangle$ representing the state machine implemented by a configuration $\rho_u \notin \Lambda_r$, then a new FFSM model FF that includes the product behavior from ρ_u can be learned by comparing and merging $\langle FF_r, M_u \rangle$.

To include a new product behavior into an existing FFSM, there are three required assumptions: (i) FF_r and M_u are complete, deterministic, initially connected, and minimal models built a priori, (ii) configurations ρ_u is known in advance, and (iii) the FSM and FFSM models under learning share a feature model that is known a priori. Thus, we extend the Definition 4.1.1 to describe how an existing family model incorporates novel product behavior described in terms of a product-specific FSM.

Definition 4.1.2 (FFSM learned from FF_r and configuration ρ_u)**.** An FFSM learned from $\langle FF_r, M_u \rangle$ is a tuple $FF = \langle F, \Lambda, C, c_0, Y, O, \Gamma \rangle$ where FF_r is a reference FFSM and M_u is the FSM specifying an updated product p_u where

- $F = F_r \cup \{p_u\}$ is the set of features in FF_r and implemented by p_u
- $\Lambda = \Lambda_r \cup \{\rho_u\}$ are the configurations in FF_r and implemented by p_u,
- $C \subseteq (S_r \cup S_u \cup (S_r \times S_u)) \times B(F)$ is the set of conditional states where

$$\begin{aligned} \forall (s_i, \phi_a) \in C_r, s_j \in S_u \mid (s_i, s_j) \in KPairs \cdot ((s_i, s_j), \phi_a | \rho_u) \in C, \\ \forall (s_i, \phi_a) \in C_r, \nexists s_j \in S_u \mid (s_i, s_j) \in KPairs \cdot (s_i, \phi_a) \in C, \\ \forall s_j \in S_u, \nexists s_i \in S_r \mid (s_i, s_j) \in KPairs \cdot (s_j, \rho_u) \in C \end{aligned} \tag{4.4}$$

- $c_0 = ((c_{0_r}, s_{0_u}), true) \in C$ is the initial conditional state,
- $Y \subseteq (Y_r \cup I_u) \times B(F)$ is a finite set of conditional input symbols,
- $O = (O_r \cup O_u)$ is the finite set of output symbols
- $\Gamma \subseteq C \times Y \times O \times C$ is the set of conditional transitions where

– two transitions $((s_i,\phi_i),(x,\phi_r),o,(s_k,\phi_k)) \in \Gamma_r$ and $(s_j,x) \in D_u$ are combined into the same conditional states if

$$\begin{aligned} \forall((s_i,\phi_i),(x,\phi_r),o,(s_j,\phi_j)) \in \Gamma_r,(s_k,x) \in D_u \mid \lambda_u(s_j,x) = o, \\ \delta_u(s_j,x) = s_l, \\ (s_i,s_j),(s_k,s_l) \in KPairs \cdot \\ ((s_i,s_j),(\phi|\rho_u)),(x,(\phi_r|\rho_u)),o,((s_k,s_l),(\phi''|\rho_u)) \in \Gamma \end{aligned} \tag{4.5}$$

– otherwise, for a conditional transition $((s_i,\phi_i),(x,\phi_r),o_r,(s_k,\phi_k)) \in \Gamma_r$ and a transition $(s_j,y) \in D_u$, there are two conditional transitions defined as follows:

$$\begin{aligned} \forall(s_i,\phi_i) \in C_r,(s_j,x) \in D_u \mid \lambda_u(s_j,y) = o_u, \\ \delta_u(s_j,y) = b_2 \cdot \\ ((s_i,\phi_r),(x,\phi_r),o_r,(s_k,\phi_r'')) \in \Gamma \\ ((s_j,\phi_u),(y,\rho_u),o_u,(s_l,\phi_u'')) \in \Gamma \end{aligned} \tag{4.6}$$

4.2 Learning by product sampling

Family model learning aims at building featured FSMs with presence conditions indicating feature-specific and family-wide behavior in terms of conditional states and transitions (DAMASCENO; MOUSAVI; SIMAO, 2019a). Since family models are expected to represent all product variants within a single artifact (SCHAEFER *et al.*, 2012), the most straightforward strategy should be exhaustive learning, i.e., where all valid products are analyzed in a brute-force fashion. However, this is only feasible for small product lines (THÜM *et al.*, 2014a).

For large and complex SPLs, we propose an approach that incorporates a `CQ` oracle that chooses an individual subset of products that are to be learned to reduce the costs to efficiently build precise family models that collectively cover the behavior of a product line. Therefore, reasonable statements on the product line's behavior are made from a potentially smaller subset of valid configurations (THÜM *et al.*, 2014a; VARSHOSAZ *et al.*, 2018),

Product sampling techniques, such as T-wise (PERROUIN *et al.*, 2010; JOHANSEN; HAUGEN; FLEUREY, 2011), should collectively cover the behavior of an SPL (VARSHOSAZ *et al.*, 2018). Hence, they should pave the way for family model learning with reasonable precision and execution costs lower than in exhaustive learning. In Algorithm 3, we depict our learning strategy that relies on a `CQ` oracle to find a subset of product configurations to be learned.

Let $C_{smpl} = \{\rho_1,\rho_2,\ldots,\rho_n\}$ be the list of sampled configurations to be learned using a technique $sample()$ that samples a subset of valid configurations $C_{smpl} = \{\rho_1,\rho_2,\ldots,\rho_n\}$ satisfying arbitrary coverage criteria for a feature model FM. In Algorithm 3, this sampling step is depicted in Line 2. For learning by sampling, we start building a fresh family model FF_1 for the first pair of product models M_1 and M_2. This fresh family model FF_1 is our initial hypothesis shown in Line 5.

As product FSMs may not be available, a model construction step $findFSM()$ shall find an FSMs M_i for each configuration $\rho_i \in C_{smpl}$. Product-specific FSM models shall be either found hand-crafted or automatically extracted by means of automata learning (ANGLUIN, 1987; VAANDRAGER, 2017). Additionally, these models must be mapped to their respective configurations so their conditional states and transitions can be annotated.

Algorithm 3 – Configuration query oracle for learning family models by sampling

```
1:  procedure CONFIGURATIONQUERYORACLE(FM, k, t, r)
2:      C_smpl = {ρ_1, ρ_2, ..., ρ_n} = sample(FM);      ▷ List of sampled configurations
3:      M_1 = findFSM(ρ_1, FM);                          ▷ Find a product FSM M_i for ρ_i
4:      M_2 = findFSM(ρ_2, FM);
5:      FF_1 = FFSM_Diff(M_1, M_2, k, t, r);             ▷ Learn fresh FFSM
6:      for j ∈ {2, ..., n − 1} do
7:          M_{j+1} = findFSM(ρ_{j+1}, FM);
8:          FF_j = FFSM_Diff(FF_{j−1}, M_{j+1}, k, t, r);   ▷ Increment family model
9:      end for
10:     return return(FF_n);
11: end procedure
```

Source: Elaborated by the author.

After learning a fresh FFSM FF_1, the algorithm starts in Line 6 an iterative step where novel product-specific behavior expressed in terms of product state machines M_{j+1} are included in partial family models FF_j. These family models are called *partial* as they describe only a subset of valid product instances. Eventually, product FSMs M_{j+1} may not exist *a priori*, and, hence, the $findFSM()$ may require automata learning to build product FSMs. At the end of this process, a family model FF_n is learned from a subset of product configurations C_{smpl}.

Sample-based approaches are known to improve the efficiency of SPL analysis by eliminating products that may already be covered by other products (THÜM *et al.*, 2014a). However, they may be incomplete and miss product-specific behaviors (VARSHOSAZ *et al.*, 2018). Higher-order feature interaction coverage criteria are known by their improved fault detection capabilities (STEFFENS *et al.*, 2012; PETKE *et al.*, 2013). Thus, for larger values of T, the T-wise sampling criteria should lead to more precise models. In the next section, we present a range of experiments to quantify the precision of models learned through exhaustive analysis and product sampling techniques.

4.3 Empirical evaluation

Several studies have underscored the importance of feature interaction coverage in product sampling (VARSHOSAZ *et al.*, 2018). Thus, we have designed a set of experiments to investigate the problem of family model learning and whether feature interaction coverage metrics can alleviate this task. We discuss the methodology, research questions, subject systems, and

experiment design of our empirical analysis in the next sections. For the sake of reproducibility and repeatability, we have made a lab package with a variety of artifacts (e.g., source code, scripts, FFSMs, FTSs, FSMs, feature models) available online on GitHub at the repository <https://github.com/damascenodiego/learningFFSM/>.

4.3.1 Research questions

To evaluate the $FFSM_{Diff}$ algorithm, we have defined four RQs to investigate whether: our approach is effective in learning succinct family models (RQ2.1); the size of the learned family models influenced by the similarity between product configurations (RQ2.2); our technique is effective in learning succinct family models compared to hand-crafted models (RQ2.3); our strategy relying on product sampling is effective in learning precise family models (RQ2.4); In Table 9, we present our hypotheses about each RQ.

RQ	Hypotheses	Description
RQ2.1	$H_0^{RQ2.1}$	The size of learned FFSMs is equal to the total size of the pairs of products under learning
	$H_1^{RQ2.1}$	The size of learned FFSMs is smaller than the total size of the pairs of products under learning
RQ2.2	$H_0^{RQ2.2}$	The size of learned FFSMs is not influenced by configuration similarity
	$H_1^{RQ2.2}$	The size of learned FFSMs is influenced by configuration similarity
RQ2.3	$H_0^{RQ2.3}$	The learned FFSMs are larger than hand-crafted models
	$H_1^{RQ2.3}$	The learned FFSMs have at most the same size as hand-crafted models
RQ2.4	$H_0^{RQ2.4}$	The FFSMs learned by sampling configurations are less precise than those learned by exhaustive analysis
	$H_1^{RQ2.4}$	The FFSMs learned by sampling configurations can be as precise as those learned by exhaustive analysis

Table 9 – Hypotheses - Learning From Difference and By Sampling

Source: Elaborated by the author.

4.3.2 Methodology

According to Thüm *et al.* (2014a), the effectiveness of family-based analysis shall be mainly influenced by the number and size of features and the amount of reuse among configurations, rather than the number of valid configurations. Therefore, for our technique to qualify as an effective family-based learning technique, we expect to learn *succinct* FFSMs where states and transitions are annotated with simplified product configurations.

By succinct, we mean that the FFSMs learned are smaller than the products under learning and hand-crafted models, especially if there is high feature sharing. By simplified, we mean that product configurations are modified by removing core features found using SAT solvers (BERRE; PARRAIN, 2010).

Furthermore, we also expected that family models learned from product configurations sampled by T-wise criteria shall collectively cover the behavior of a product line and be at least as precise as those models recovered using exhaustive learning. Thus, we designed a set of experiments to measure how succinct and precise are the learned family models.

As a measure of *succinctness*, we have used the size of the FFSMs learned from product pairs. We describe *size* in terms of *number of transitions* as it is one of the factors that influence the complexity of model-based techniques (BROY *et al.*, 2005; BAIER; KATOEN, 2008) and that is used to interpret the language and structure of FSMs (WALKINSHAW; BOGDANOV, 2013). To complement our analysis, we have included the number of states of learned models.

To measure the *statistical significance*, we have used the Mann-Whitney (MW) test to check if there was significant difference ($p < 0.05$) between the sizes of the learned FFSM and the reference models, i.e., the product pair or the hand-crafted family model. To measure the *scientific significance* (KAMPENES *et al.*, 2007; ARCURI; BRIAND, 2011), we have used the Vargha-Delaney's (VD) $\hat{A}$ effect size (VARGHA; DELANEY, 2000) to assess the probability of the learned FFSMs being more succinct than the reference model. If $\hat{A} < 0.5$, the learned FFSM is smaller than the pair of products. If $\hat{A} = 0.5$, they have equivalent sizes. To categorize the magnitude of the $\hat{A}$ effect size, we have used the intervals between $\hat{A}$ and 0.5 recommended by Hess and Kromrey (2004) and implemented in the `effsize` package (TORCHIANO, 2017): $negligible < 0.147 \leq small < 0.33 \leq medium < 0.474 \leq large$.

As a measure of *configuration similarity*, we have used the Hamming distance between product configurations with respect to feature selection (AL-HAJJAJI *et al.*, 2017). Thus, we have analyzed the impact of configuration similarity over the succinctness of learned family models using the Pearson's correlation coefficient between *the ratio of the size of the learned FFSM to the total size of the product pairs*, on the one hand, and *the similarity between configurations*, on the other hand. Thus, the normalized size would range between 0.5, if both product FSMs are equivalent; and 1.0, otherwise.

As a measure of *precision*, we have used the model precision proposed by Walkinshaw and Bogdanov (2013) to calculate how many transitions from all valid product FSMs were included into models learned by sampling. We have used the attenuation ratio as $k = 0.5$, the threshold of most equivalent pairs as $t = 0.4$, and the ratio for best matches as $r = 1.4$. To analyze feature models and calculate configuration similarity, we have used the SAT4J solver (BERRE; PARRAIN, 2010) and the FeatureIDE framework (THÜM *et al.*, 2014b). To solve the linear equations that would match states and transitions, we have used the Apache Commons Mathematics library (Apache, 2016).

4.3.3 Experiment design

To answer the **RQ2.1**, we have used the $FFSM_{Diff}$ algorithm to learn family models from all pairs of product configurations and combine their FSM models into FFSMs, as shown in Figure 20. Then, we have checked whether there were significant and relevant differences between the sizes of the learned FFSM, the pair of products under learning.

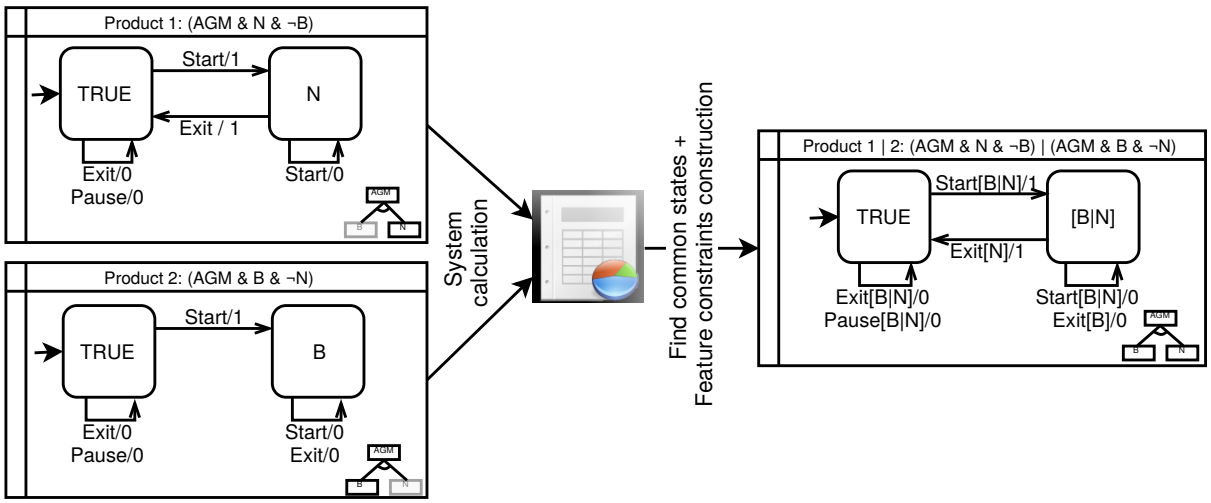


Figure 20 – Experiment design

Source: Elaborated by the author.

To answer the **RQ2.2**, we have used the Pearson's correlation coefficient between *the normalized size of learned FFSMs and the configuration similarity*. To answer the **RQ2.3**, we have compared the size of the models learned in the analysis of the RQ2.1 against the size of the hand-crafted family models.

Let $\{\rho_0, \rho_1, \ldots, \rho_m\} \subseteq B(F)$ be a set of valid configurations, such that the product derived from ρ_i has at most the same number of states as ρ_{i+1}, i.e., they are sorted by their FSM size. Thus, to complement our answer to **RQ2.3**, we have analyzed the size of family models recovered by including in the FFSMs resulting from all products $\bigcup_{i=0}^{j-1}(\rho_i)$ with the FSM of the next product ρ_j; and compared against the size of its hand-crafted version.

Finally, to answer the **RQ2.4**, we have used the FeatureIDE framework (THÜM *et al.*, 2014b) to generate subsets of valid products satisfying the feature-wise (aka. 1-wise), pair-wise (aka. 2-wise), 3-wise, 4-wise and all-valid configurations criteria. Particularly, we have used the Chvatal algorithm for the set-covering problem (CHVATAL, 1979).

The Chvatal algorithm has been adapted by Johansen, Haugen and Fleurey (2011) for T-wise product sampling that is available in the FeatureIDE workbench (THÜM *et al.*, 2014b). In Figure 21, we illustrate our experiment for the RQ2.4.

Let $C_{smpl} = \{\rho_1, \rho_2, \ldots, \rho_m\} \subseteq B(F)$ be a subset of valid configurations generated by some arbitrary sampling criteria, such that they are sorted by configuration similarity (AL-

HAJJAJI *et al.*, 2017). For each sampled subset, we iteratively learn a partial FFSM by merging the FSMs of all configurations $\bigcup_{i=1}^{j-1}(\rho_i)$ with its next configuration ρ_j ordered by configuration similarity, as described in Algorithm 3.

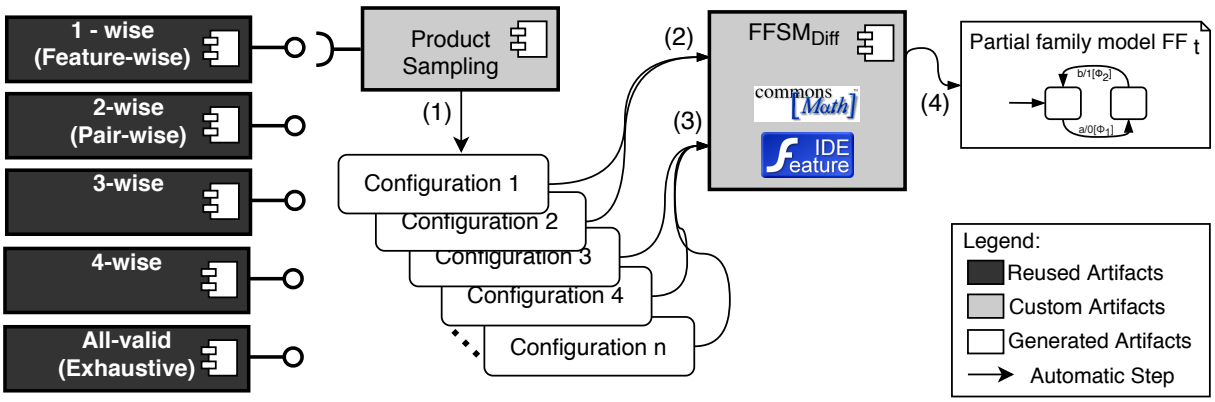


Figure 21 – Experiment design - Learning FFSMs by sampling

Source: Elaborated by the author.

The process of learning by sampling, shown in Figure 21, works as follows: (1) finding sets of product configurations satisfying each of the T-wise coverage criteria, (2) learning a fresh FFSM from the two first product configurations, and (3) increment family models by including the remainder product behavior, as in Line 6 of the Algorithm 3. At the end of this process, a partial family model FF_t satisfying a given T-wise coverage criteria is obtained.

Once a partial family model FF_t satisfying some T-wise coverage is learned, we evaluate the quality of such model by means of its precision (WALKINSHAW; BOGDANOV, 2013). To evaluate model precision, we used the $FFSM_{Diff}$ itself and the same t,k,r parameters used for learning family models to calculate how many transitions from all valid products, including the ones not analyzed, are expressed in the FFSM learned by sampling.

4.3.4 Experiment artifacts

The artifacts supporting the experiments shown in this chapter are available in a GitHub repository at the link <https://github.com/damascenodiego/learningFFSM/>. Currently, this repository contains tags, i.e., `splc19` and `EMSE`, for each of our studies (DAMASCENO; MOUSAVI; SIMAO, 2019a; DAMASCENO; MOUSAVI; SIMAO, 2020), respectively.

The $FFSM_{Diff}$ repository is organized as a Java project where there is one main package available, namely `uk.le.ac`. In this package, there are code artifacts that we have developed to (i) read/write FSMs; (ii) solve the systems of linear equations to compare the FSMs and FFSMs under learning; and (iii) merge FSM models and annotate FFSMs. The project is organized using the Eclipse IDE (ECLIPSE, 2019) and Java v1.8 (ORACLE, 2014).

To read and write FSM files, we designed the `ProductMealy` class by extending the `CompactMealy` class from LearnLib (RAFFELT; STEFFEN, 2006). Thus, we extended basic operations over FSMs (e.g., reset, transition/output functions) to product FSM models implementing our `IConfigurableFSM` interface and product configurations.

Using the `Apache Commons Math` library (Apache, 2016), we have implemented our algorithm to find state pairs most likely to be equivalent. It starts mapping the initial states (s_{0_r}, s_{0_u}) as our first landmark. Hence, we start searching for pairs likely to be equivalent, given the t and r parameters. Based on empirical observations and recommendations by Walkinshaw and Bogdanov (2013), we have set the attenuation ratio as $k = 0.5$, the threshold of most equivalent pairs as $t = 0.4$, and the ratio for best matches as $r = 1.4$.

To represent FFSMs, we designed the `FeaturedMealy` class by extending the `FastNFA` class, one of the LearnLib building blocks, to represent non-deterministic models. We opted for this representation because, if we ignore the presence conditions, an FFSMs can be modeled as a non-deterministic FSM. To represent conditional states/transitions, we designed the classes `ConditionalState` and `ConditionalTransition` with collections of `Node` objects. The `Node` class is a building block from `FeatureIDE` to represent feature constraints.

In the folder `lib`, we have some of the libraries used in our project. These include `FeatureIDE` (THÜM *et al.*, 2014b), and `Apache Commons Math` (Apache, 2016). Other libraries (e.g., LearnLib and AutomataLib (RAFFELT; STEFFEN, 2006)) are by the Apache Maven. The file `learnFFSM.jar` is a compiled version of the $FFSM_{Diff}$ algorithm.

In the folder `FFSM_diff/Benchmark_SPL/`, we have added the subject systems and their respective models (i.e., FSMs, FFSMs), feature models, visual representations and test scripts. In the folder `FFSM_diff/Benchmark_SPL/exp_splc2019`, there is an RStudio project (RSTUDIO, 2019) with the results obtained in our first study (DAMASCENO; MOUSAVI; SIMAO, 2019a). To replicate this set of experiments, you shall use the test scripts `run_<ID>.py` and `run_<ID>_pairs.py` in folder `Benchmark_SPL/exp_splc2019/scripts`.

In the folder `FFSM_diff/Benchmark_SPL/exp_emse`, we have the plots and tabulated results from our extended analysis (DAMASCENO; MOUSAVI; SIMAO, 2020). The raw results are found in the file `FFSM_diff/Benchmark_SPL/wise2learn.zip`. The experimental bash scripts are named as `emse_*.py`. There is an RStudio (RSTUDIO, 2019) project at `FFSM_diff/Benchmark_SPL/exp_emse/learningFFSMs.Rproj` and an R script for statistical analysis at `FFSM_diff/Benchmark_SPL/exp_emse/script.r`.

4.3.5 Subject systems

In order to evaluate our hypotheses, we used 105 Mealy machines derived from six SPLs from previous studies (DAMASCENO; MOUSAVI; SIMAO, 2019a; DEVROEY *et al.*, 2015). Although these case studies are abstract representations of SPLs, they comprise many

non-trivial aspects, such as infinite behavior and states with similar or identical behavior in different products (SAMIH *et al.*, 2014; DAMASCENO; MOUSAVI; SIMAO, 2019a). In Table 10, we present the SPLs in terms of numbers of features and valid configurations from the feature model, and states and transitions in the family model.

SPL		Feature model		Family model	
ID	Name	Features	Valid conf.	States	Transitions
AGM	Arcade Game Maker	13	6	6	35
VM	Vending Machine	9	20	14	197
WS	Wiper System	8	8	13	112
AEROUC5	Aero UC5	7	9	25	450
CPTERMINAL	Card Payment	13	30	11	176
MINEPUMP	Minepump	9	32	25	575

Table 10 – Description of the SPLs under learning - Feature and family models

Source: Elaborated by the author.

All the FSMs used in this study were derived from FFSMs using the model derivation operator Δ_ρ (FRAGAL; SIMAO; MOUSAVI, 2017). Thus, we had reference FFSMs to assess the learned models, i.e.,, the hand-crafted product-line FFSMs, the intermediate FFSMs. Each of the SPLs is described in the following sections.

The Arcade Game Maker SPL

The Arcade Game Maker (AGM) SPL is a well-known pedagogical example that describes arcade game machines with different game rules. In our version of the AGM SPL (FRAGAL; SIMAO; MOUSAVI, 2017), we have support to three alternative games and one optional feature to *Save* the game. The FSMs derived from the AGM FFSM have at least three states for the game modes *start*, *running* and *paused*. If *Save* feature is included, a fourth state is added. The feature model for the AGM SPL has been shown in Figure 7.

The Vending Machine SPL

The Vending Machine (VM) is an abstract representation of a product-line that we hand-crafted (DAMASCENO; MOUSAVI; SIMAO, 2019a) as FFSM/FSM models inspired by featured transition systems from a collection of illustrative examples of SPLs (CLASSEN, 2010). In Figure 22, we depict the feature model for the VM SPL.

In the VM SPL, product instances shall feature at most three beverages (i.e., *Coffee* - `COF`, *Tea* - `TEA`, and *Cappuccino* - `CAP`), support one currency (i.e., *Dollar* - `DOL` or *Euro* - `EUR`) and include one optional *Ringtone* - `TON` that can be played when a beverage is completed. The VM SPL constitutes an interesting case as it resulted on FSMs with distinct languages. Among

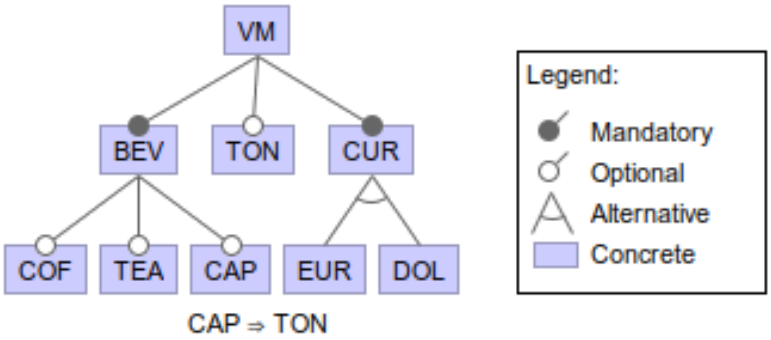


Figure 22 – The VM feature model

Source: Elaborated by the author.

the FSMs derived, we highlight two main differences: the addition of states for each of beverage and changes in the language of outgoing transitions from the initial state for each currency.

The Wiper System SPL

The Wiper System (WS) is another abstract behavioral representation of SPL handcrafted (DAMASCENO; MOUSAVI; SIMAO, 2019a) based on a product-line from an academic catalog of SPLs (CLASSEN, 2010). In Figure 23, we depict the feature model of the WS SPL.

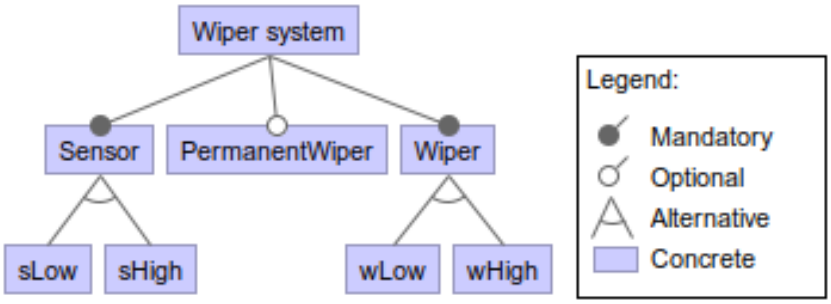


Figure 23 – The WS feature model

Source: Elaborated by the author.

Our WS SPL has two subsystems: the `Sensor` to detect rain and the `Wiper` itself; available in two qualities, namely, high and low; and one optional feature for permanent movement `PermanentWiper`. A high-quality sensor `sHigh` can discriminate between heavy and light rain, whereas a low-quality sensor `sLow` can only distinguish between rain and no rain. Similarly, the `wHigh` and `wLow` quality wipers can operate at two and one speeds, respectively. These features lead to significant changes in the structure and language of its derived FSMs.

The Aero UC5 SPL

The Aero UC5 (AEROUC5) model has been originally introduced by Samih *et al.* (2014) as a set of extended Markov models designed by engineers. The AEROUC5 is an

industrial situational awareness system for helicopters flying in degraded visual environments. The AEROUC5 feature model has been originally designed with 25 features and more than 5 million valid configurations (VIBeS, 2016a).

By inspecting the actual family model that was given as a featured transition system (CLASSEN *et al.*, 2013), we found that only a few features were, in fact, annotating the model. Thus, we have opted to discard those features that did not have impact on the behavioral model. Our adapted version of this feature model contains four features related to (i) display real object or (ii) 3D conformal visual cues on a head-tracked Helmet, and (iii) marking intended lading positions or (iv) obstacles on the ground using an Obstacle Warning System. In Figure 24, we show our version for the AEROUC5 SPL.

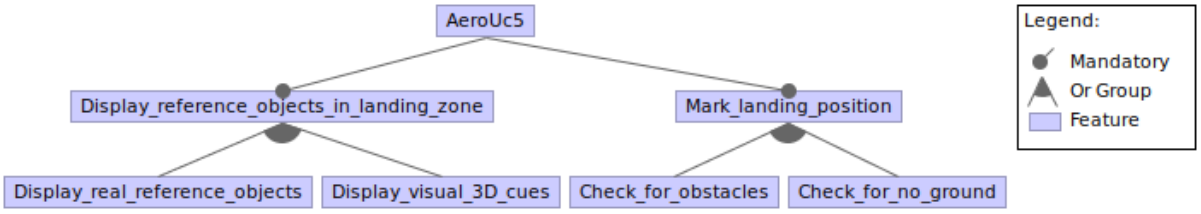


Figure 24 – The Aero UC5 feature model

Source: Elaborated by the author.

To instantiate complete FSMs from the FTS, we have used the VIBeS tool to derive projections and identify the transitions valid for all valid configurations. For each valid configurations, we have created FSM transitions with output as 1 when a state included some FTS transition. For the missing transitions, we have created self-loop transitions returning 0. This family model is intended to be a more realistic subject as it has been designed by engineers and is one of our largest model in terms of number of states and transitions.

The Card Payment Terminal SPL

The Card Payment Terminal (CPTERMINAL) SPL is another model originally designed as an FTS (CLASSEN *et al.*, 2013). In Figure 25, we depict the feature model for the Card Payment Terminal product line.

This product line has been defined by a software engineer based on EMV and PCI norms and its family model describes the behavior of one terminal that accepts card payment with `DirectDebit` and/or `CreditCard`. It supports a card owner authentication method (i.e., `Signature` and optionally `PIN` code), and synchronous (`Online`) or asynchronous (`Offline`) connection to the payment service (VIBeS, 2016b). To derive FSMs, we have used the same approach applied to the AEROUC5 SPL.

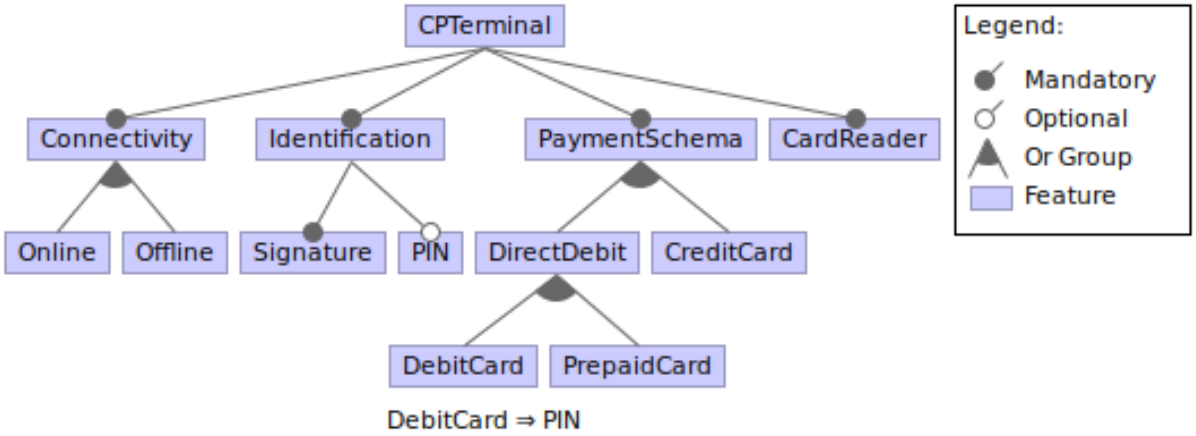


Figure 25 – The Card Payment Terminal feature model

Source: Elaborated by the author.

The Minepump SPL

The Minepump (MINEPUMP) product line has been presented by Classen *et al.* (2013). The purpose of this system is to keep a mine shaft clear of water while avoiding the danger of methane related explosions. In Figure 26, we show the feature model for the Minepump SPL.

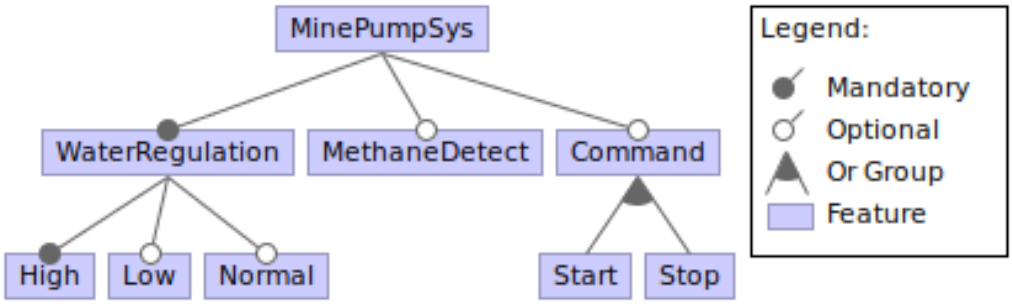


Figure 26 – The Minepump feature model

Source: Elaborated by the author.

To monitor the mine shaft, it uses the `WaterRegulator` and `MethaneDetect` features. The system is activated once the water level reaches a preset threshold, but only if the methane level is below a critical limit. Similarly to the AEROUC5 and CPTERMINAL SPLs, the product FSMs for the Minepump SPL were derived from an FTS model (VIBeS, 2016c).

4.4 Analysis of results

In this section, we discuss the main results of our experiments in terms of the RQs and Hypotheses shown in Table 9. For the sake of space, we will only plot and highlight the main findings of our experiments. The full set of plots and tabulate results are available on GitHub at the repository <https://github.com/damascenodiego/learningFFSM> .

4.4.1 Size of learned family models vs. products under learning

Regarding the succinctness of the learned FFSMs, we observed that on average all learned FFSMs presented fewer transitions than their respective pairs of products under learning. In Figure 27, we show boxplots for the sizes of the learned FFSMs and the total size of the pairs of products under learning in terms of number of transitions. Dashed lines indicate the number of transitions in the original hand-crafted model.

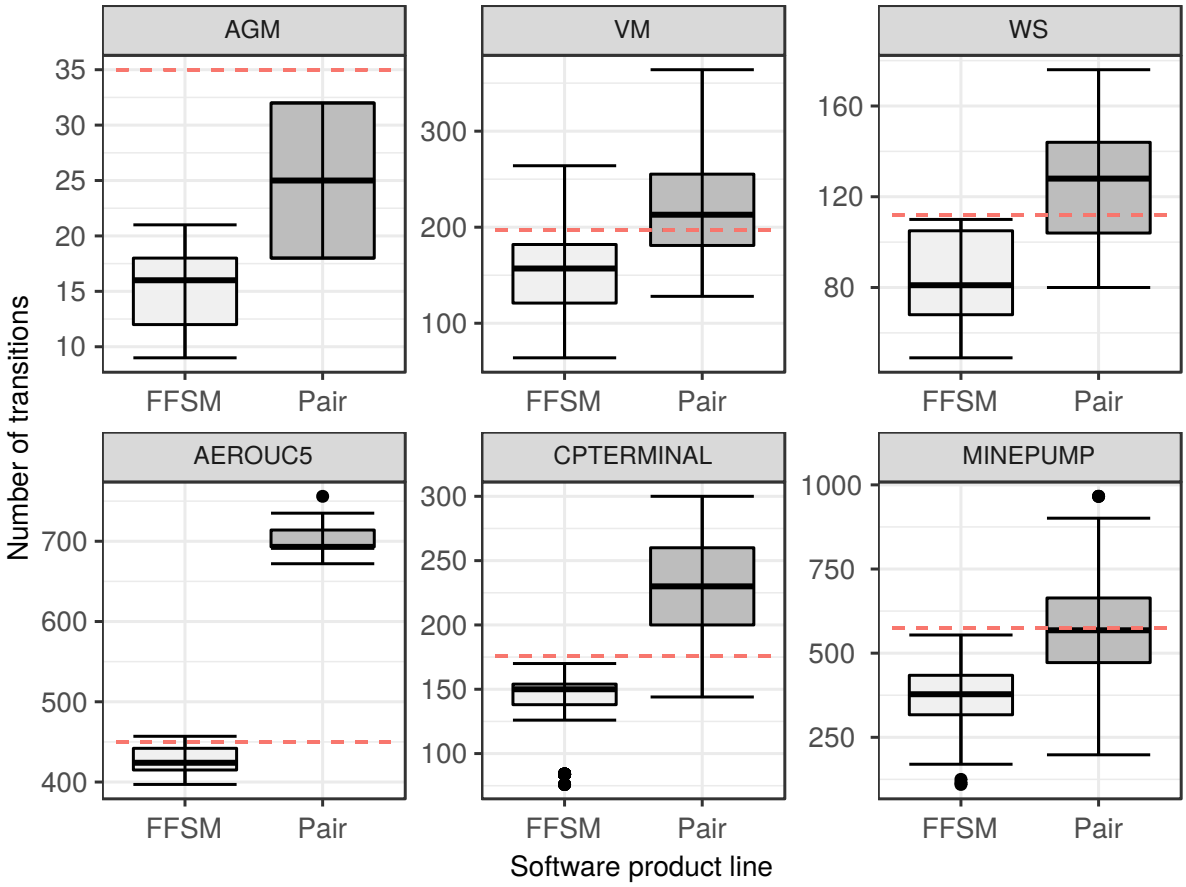


Figure 27 – Number of transitions in the learned FFSMs and pairs of products

Source: Elaborated by the author.

In terms of number of states, we also found that the learned FFSMs had fewer states than their pairs of products under learning. Figure 28 shows the boxplots for the numbers of states in the learned FFSMs and the total number of states in the pair of products under learning. Dashed lines indicate the number of states in the original hand-crafted model.

To assess the statistical difference and significance of our results, we ran the MW test and VD's $\hat{A}$ effect size to check the significance ($p < 0.05$) and magnitude of the difference between the sizes of the learned FFSMs and the pairs of products under learning. In Table 11, we present the p-values and effect sizes comparing the sizes of our learned family models against the pairs of product under learning in terms of states and transitions.

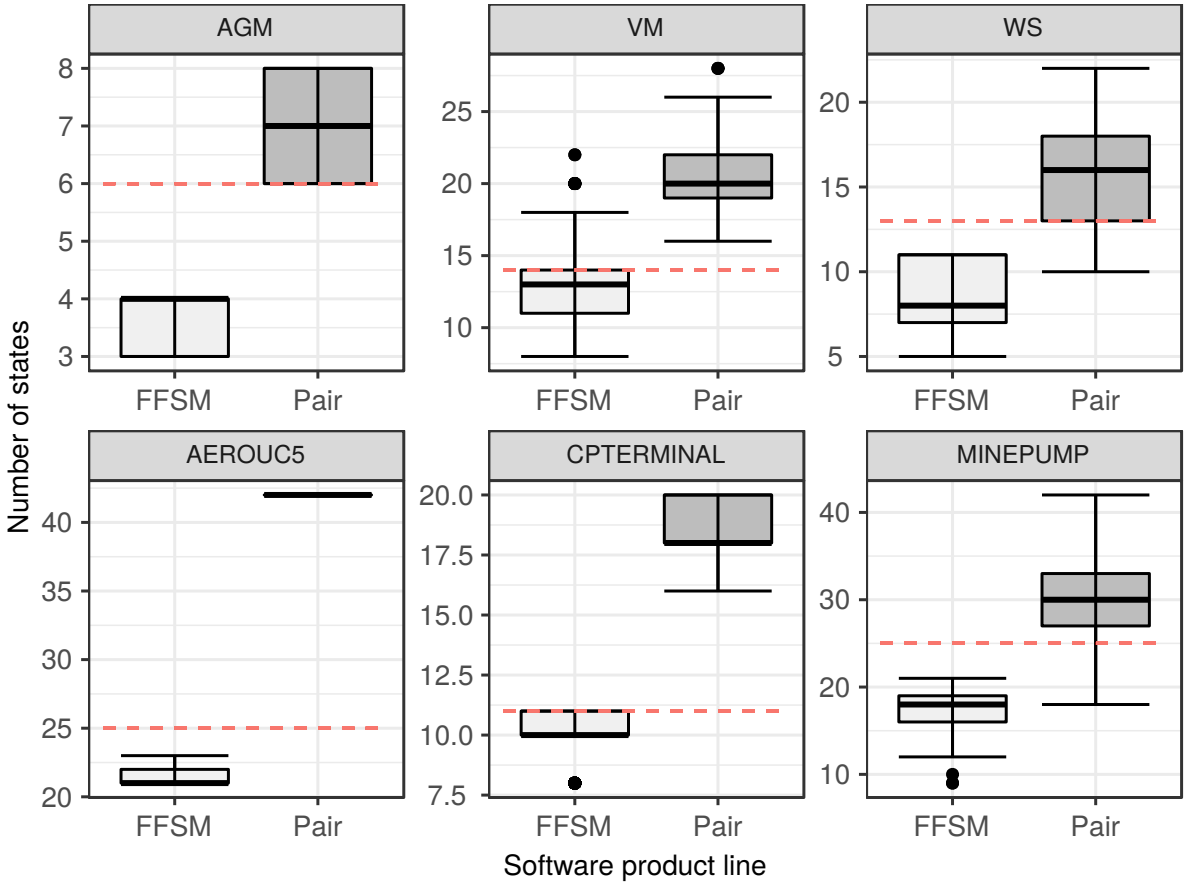


Figure 28 – Number of states in the learned FFSMs and pairs of products

Source: Elaborated by the author.

# of	Test	AGM	VM	WS	CPTERMINAL	MINEPUMP	AEROUC5
States	MW	1.07E-08	9.21E-61	2.92E-09	2.07E-160	1.06E-171	6.77E-19
	VD	0.000	0.037	0.050	0.000	0.005	0.000
Transitions	MW	1.73E-06	6.44E-31	3.55E-06	2.30E-135	1.73E-101	2.14E-16
	VD	0.075	0.174	0.146	0.032	0.120	0.000

Table 11 – Mann-Whitney test and Vargha-Delaney's effect size: learned FFSM vs. Product pair

Source: Elaborated by the author.

As indicated by Figures 27 and 28, as well as by Table 11, there were statistically significant differences between the sizes of the learned FFSMs and the pair of products under learning. For the effect sizes, we also found that the differences had a *large* magnitude. Thus, our results support the hypothesis $H_1^{RQ2.1}$ that the sizes of learned FFSMs are at most equal to the total size of products under learning.

4.4.2 Size of learned family models vs. configuration similarity

In addition to analyzing the size of FFSMs learned against the products under learning, we analyzed the relationship between learned family model size and configuration similarity

using the Pearson's correlation coefficient. In Figure 29, we show a set of scatter plots for the configuration similarity degree against the size of the learned FFSMs for all pairs of products of each SPL.

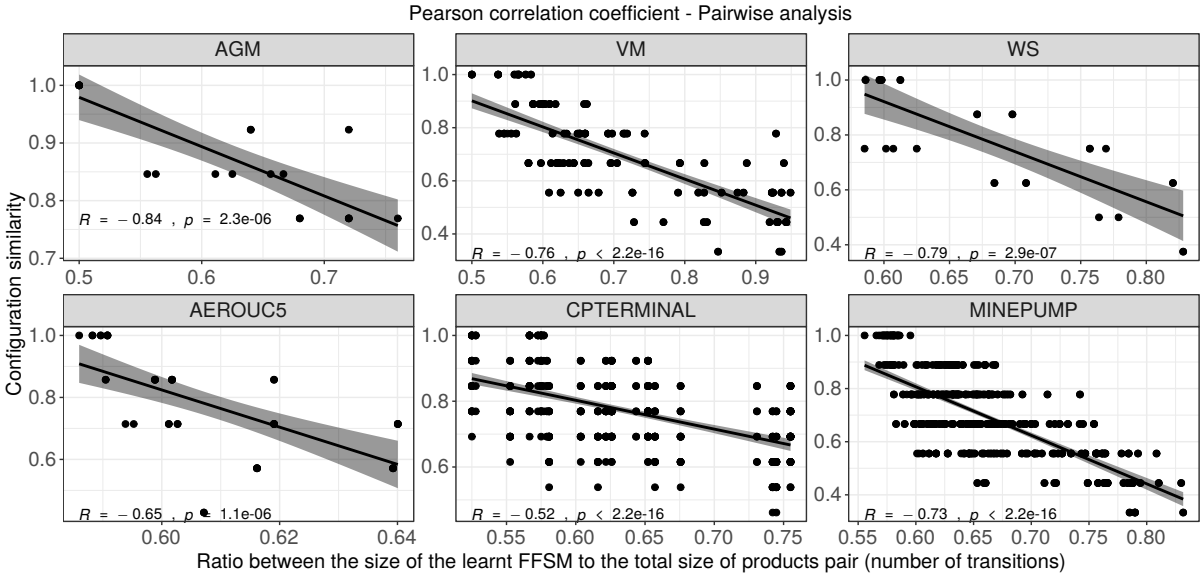


Figure 29 – Scatter plots for the relationship between the normalized size of the learned FFSM and configuration similarity

Source: Elaborated by the author.

A configuration similarity equals to 1.0 means that both products have the same feature configuration. A ratio between the size of the learned FFSM and the total size of products equals to 0.5 means that the products analyzed implement equivalent behavior; otherwise they have some variability expressed by distinct transitions.

By analyzing the Pearson correlation coefficient, we found *strong* negative correlations between FFSM size and configuration similarity for the VM, WS, AEROUC5 and MINEPUMP product lines; *very strong* negative correlation for the AGM product line; and *moderate* negative correlation for the CPTERMINAL. These results indicate that FFSMs learned from products implementing similar configurations were more succinct than those built from products implementing fewer common features. Thus, our results support the hypothesis $H_1^{RQ2.2}$ that the size of the learned FFSMs is influenced by configuration similarity.

4.4.3 Size of learned family models vs. hand-crafted models

To evaluate the succinctness of the learned FFSMs, we also compared the size of hand-crafted models against the FFSMs learned from pairs of products. In Figures 27 and 28, we show the boxplot for size of learned FFSMs from product pairs. A dashed line indicates the size of the original hand-crafted FFSM.

To compare the sizes of the hand-crafted models and the FFSMs learned from product pairs, we used the Mann-Whitney test and $\hat{A}$ effect size. Table 12 shows the results for the Mann-Whitney test and effect size comparing the size of the learned FFSMs against the size of hand-crafted models.

# of	Test	AGM	VM	WS	CPTERMINAL	MINEPUMP	AEROUC5
States	MW	1.16E-09	1.35E-09	1.80E-12	3.38E-76	1.54E-199	6.77E-19
	VD	0	0.35	0	0.22	0	0
Transitions	MW	3.06E-09	1.48E-54	2.83E-12	1.50E-175	1.14E-198	1.51E-10
	VD	0	0.09	0	0	0	0.13

Table 12 – Mann-Whitney test and Vargha-Delaney's effect size: learned FFSM vs. Hand-crafted model

Source: Elaborated by the author.

By analyzing the results of the MW test, we found statistically significant differences ($p < 0.01$) between the sizes of FFSMs learned from all SPLs. The VD effect sizes indicated differences of *large* magnitude where FFSMs learned from product pairs included fewer transitions than their hand-crafted versions. These findings persisted for the number of states, except for the VM SPL where we found a small magnitude on the difference between the number of states of the FFSM models learned from product pairs. Thus, our results support the hypothesis $H_1^{RQ2.3}$ that learned FFSMs have at most the same size as hand-crafted FFSMs.

These results corroborate with our findings (DAMASCENO; MOUSAVI; SIMAO, 2019a) where two FFSMs were recovered with fewer states than their original models, i.e., AGM and WS. We inspected the learned and hand-crafted FFSMs models and found that two states presented similar outgoing transitions and, hence, they could be merged. In Figure 30, we show the size of the learned FFSMs from all products of three of our SPLs.

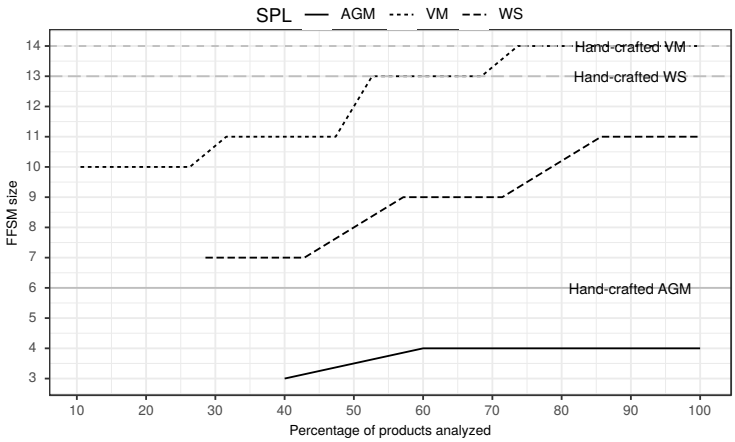


Figure 30 – Size of the recovered FFSMs

Source: Damasceno, Mousavi and Simao (2019a).

As shown in Table 10, the hand-crafted FFSMs for the AGM and WS SPLs presented 6 and 13 conditional states, respectively. In Damasceno, Mousavi and Simao (2019a), the FFSMs

learned from AGM and WS, in their turn, included 4 and 11 states, respectively. Moreover, we found that the FFSMs learned from the AGM SPL coincided with the alternative representation reported by Fragal (2017) with fewer states (DAMASCENO; MOUSAVI; SIMAO, 2019a). In this alternative representation shown in Figure 31, all three conditional states are composed into one state annotated with the disjunction of the alternative features.

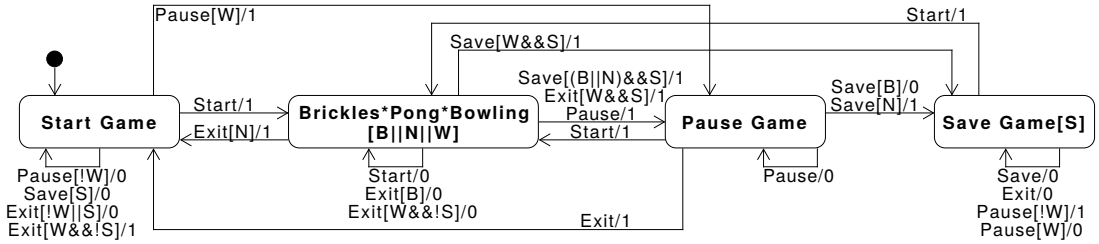


Figure 31 – Alternative FFSM for the AGM SPL with fewer states

Source: Fragal (2017).

The FFSMs hand-crafted and learned for the VM SPL presented the same size. In this particular case, the order that the products have been analyzed led to an FFSMs learned with the same size of its hand-crafted version. These findings indicate that selecting “good" product configurations could help to learn more succinct FFSMs. Thus, product sampling (VARSHOSAZ *et al.*, 2018) could potentially improve the overall behavior incorporated into an existing FFSM, as we see in the next section.

4.4.4 Precision of family models learned by sampling

For each $T \in \{1,2,3,4\}$, we have used the T-wise sampling criteria to generate subsets of valid product configurations and learn FFSM models by sampling. To evaluate the precision of learning by sampling, we used the all-valid configurations criteria to derive all products FSMs and build reference FFSMs for each SPL. In Table 13, we depict the sizes of the subsets of products generated by each configuration sampling criteria.

SPL	Size of the sampled subset generated by T-wise				
	Feature-wise	Pair-wise	3-wise	4-wise	All-valid
AGM	3	6	6	6	6
VM	2	6	13	19	20
WS	2	5	8	8	8
AEROUC5	3	6	9	9	9
CPTERMINAL	3	8	16	24	30
MINEPUMP	3	7	13	24	32

Table 13 – Number of configurations in the subsets generated by each criteria

Source: Elaborated by the author.

To evaluate the precision of the models *learned by sampling*, we have used the $FFSM_{Diff}$ to measure the proportion of transitions from the analyzed models (i.e., learned by sampling) that

are also in the reference models (i.e., individual FSMs of all valid products). Thus, a precision equals to 1 indicates that all transitions from all valid products are included in the FFSM learned by sampling.to In Figure 32, we show the precision of the FFSM learned from each sampling criteria and the full set of all valid products.

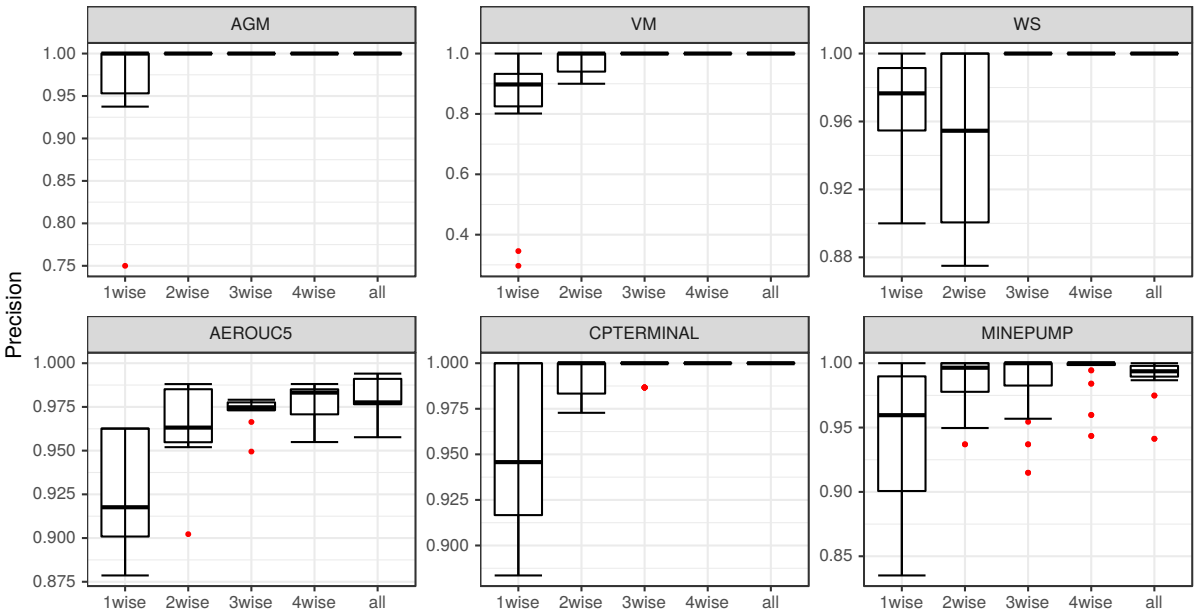


Figure 32 – Model precision by sampling criteria

Source: Elaborated by the author.

As our results indicate, model precision turned out to be larger for higher values of T. For most of the product lines, excluding the AGM, we found a significant difference between the models learned by feature-wise sampling and all-valid configurations, i.e., exhaustive analysis. By comparing exhaustive analysis against feature-wise sampling, we found effect sizes categorized as medium to large with the exhaustive criteria reaching higher precision.

Higher interaction strengths are known for their improved fault detection capabilities (STEFFENS *et al.*, 2012; PETKE *et al.*, 2013). Similarly, our results corroborate to these findings as they indicate that family models learned by 3- and 4-wise sampling tend to be more precise than those built by feature-wise and pair-wise sampling.

As shown in Table 13, the 3- and 4-wise sampling criteria generated the same number of configurations as the exhaustive criteria for the AGM, WS and AEROUC5 product lines. Thus, similar precision levels should be expected. The results for the MW and VD's tests corroborate these findings as they indicated no significant difference between the precision of models learned by 3-wise, 4-wise and all-valid sampling criteria.

For the VM, CPTERMINAL and MINEPUMP product lines, we found that models learned by 3-wise and 4-wise sampling reached precision levels similar to those learned by using the exhaustive criteria. For these product lines, we found either no significant differences or

effect sizes categorized as negligible for small between the models learned by the 3-wise, 4-wise and all-valid sampling criteria. These findings indicate that product sampling can be helpful at reducing the costs for recovering family models from product families without analyzing all-valid products.

For the AEROUC5 and MINEPUMP product lines, we found that FFSMs learned by exhaustive analysis did not reach precision levels equal to 1. We associate this to a possibly large number of state pairs with similar scores returned by the *identifyLandmarks*() function. Thereafter, multiple mappings between state pairs have been found by our algorithm and the selected pairs deemed transitions as removed and affected precision. These results support our hypothesis $H_1^{RQ2.4}$ that FFSMs learned by sampling can be at least as precise as those learned by running exhaustive analysis.

4.5 Threats to validity

In this section, we discuss the threats to validity and limitations for the experiments shown in this chapter.

Conclusion validity: These threats concern to the relationship between treatment and outcome of our investigation. To ensure the reliability of our measures and treatment implementation, we have a setup in place based on widely used tools for state-machine learning (RAFFELT; STEFFEN, 2006), SAT solving (BERRE; PARRAIN, 2010), and feature-oriented software development and analysis for SPLs (THÜM *et al.*, 2014b).

External validity: These concerns relate to the generalization of our results to industrial subjects. Our results have been obtained using six product lines, where one of them has been inspired by a real system (SAMIH *et al.*, 2014); the small number of real product lines poses a threat to external validity. Another variable that will form a threat to external validity is the behavioral variability inherent to the valid products of our subject systems. For some of our subjects, the behavioral difference between products has made the exhaustive analysis the only criteria able to recover family models fully precise.

Internal validity: These concerns relate to the phenomena that can affect the causal relationship between the treatment and outcomes. One variable that will form a threat to internal validity is the *order* of incorporating product FSMs into product line FFSMs. Originally, we have considered one single order for recovering FFSMs by incrementally incorporating products, i.e., products sorted by their FSM size (DAMASCENO; MOUSAVI; SIMAO, 2019a). In this case, product configuration prioritization (HENARD *et al.*, 2014) could be employed. However, the impact of prioritization techniques in family model learning is out of the scope of this PhD Thesis.

Construct validity: These are concerned with the ability to draw correct conclusions

about the treatment and outcomes. Two factors that will form threats to construct validity are the nature of the hand-crafted FFSMs and the subsets sampled by T-wise criteria. Highly specialized engineers are more likely to come up with better subsets of products configurations and more compact models than professionals with less experience. In addition to that, T-wise criteria may still sample large subsets, compared to the set of all-valid products. In these cases, domain-specific expertise may be useful to optimize family model learning.

4.6 Discussion

What are the implications for practitioners? While exhaustive learning may be suitable for small product lines, for large SPLs, it becomes impractical. Our sampling-based learning technique paves the way for efficient and effective reverse engineering techniques for software product lines. Additionally, we believe that it can complement automata learning (IRFAN; ORIAT; GROZ, 2013; STEVENSON; CORDY, 2014; AICHERNIG *et al.*, 2018) and reverse engineering feature models (HASLINGER; LOPEZ-HERREJON; EGYED, 2011; RYSSEL; PLOENNIGS; KABITZSCH, 2011; AL-MSIE'DEEN *et al.*, 2014).

What types of systems may it work/not work? In product sampling, there is an assumption that sampled products shall collectively cover the behavior of product families. Thus, if there is no such behavioral overlap, then products learned by sampling may never be precise enough. Hence, a possible alternative could be the application of an iterative `CQ` oracle for *testing* if a partial family model already includes the states and transitions of a given SUL and *learning* any unseen behavior.

How are the different notions of variability represented? Currently, our approach annotates state and transitions using the disjunction of simplified configurations. As a result of this design decision, the representation of feature constraints is limited to a unique format (i.e., OR with ANDs). To overcome this limitation, more sophisticated presence-condition simplification techniques (VON RHEIN *et al.*, 2015) could be used to reduce the complexity of feature constraints. Other possible solutions are the usage of feature model refactoring and specialization (BENAVIDES; SEGURA; RUIZ-CORTÉS, 2010) to come up with the constrains for conditional state and transitions.

4.7 Final remarks

Explicit models are important assets to the analysis and development of high-quality systems. They can help developers on program comprehension, software refactoring, model checking, and model-based testing. However, in the lack of proper maintenance, software models often become outdated, incomplete, or even deprecated. The problem of outdated and deprecated models can arise in the setting of SPLs and hamper the application of family-based analysis.

Family-based analysis operates on a single artifact, referred to as a family model, annotated with feature constraints to express variability in terms of states and transitions specific to product configurations. Albeit reasonably efficient, family-based analysis is challenging because creation and maintenance of family models tend to be time consuming and error-prone, especially if there are crosscutting features. To tackle this issue, we have introduced $FFSM_{Diff}$, an automated technique to learn succinct FFSMs from sets of product FSMs.

Our technique incorporates variability to compare and merge FSMs and annotate states and transitions with feature constraints. Our technique copes with two tasks: (i) learning fresh FFSMs from two product FSMs, and (ii) including novel product behavior expressed as an FSM into an existing FFSM. We have built upon our expertise on the $FFSM_{Diff}$ algorithm product sampling techniques to reduce the cost of exhaustive learning and obtain accurate family models.

To evaluate our technique, we used 105 Mealy machines derived from six SPLs from previous studies and measured the effectiveness of our algorithm in terms of the size of the learned FFSMs and the amount of feature reuse. Our results have supported the hypothesis that families of FSMs can be effectively merged into succinct FFSMs, especially if there is high feature reuse among products. Moreover, we have found that family models learned by product sampling can be as precise as those learned by exhaustive analysis. These results pave the way for more efficient family model recovery from product lines. Therefore, this study can pave the way to several family-based analysis techniques without family models specified *a priori*, such as SPL re-engineering (FENSKE; THüM; SAAKE, 2013), SPL evolution (MARQUES *et al.*, 2019), traceability analysis (VALE *et al.*, 2017), and model-based regression testing (RUNESON; ENGSTRöM, 2012).

The $FFSM_{Diff}$ algorithm has been originally published as a full paper at the 23rd International Systems and Software Product Line Conference (SPLC) (DAMASCENO; MOUSAVI; SIMAO, 2019a). An extended version of this paper has been submitted for a Special Issue on "Configurable Systems" in the Journal of Empirical Software Engineering (DAMASCENO; MOUSAVI; SIMAO, 2020), which is currently at the *rebuttal phase*.

The ideas presented in this chapter have been designed and implemented during a research visit at the University of Leicester (UK) under the supervision of prof. Dr. Mohmammad Reza Mousavi and prof. Dr. Adenilso Simao from January/2019 to December/2019. The artifacts used in this study are available in a lab package that has been open-sourced on GitHub at the repository <https://github.com/damascenodiego/learningFFSM/>.

CHAPTER

5

CONCLUSION

Software models are key assets in a range of activities in the software development life cycle. Nevertheless, as modifications take place in coding artifacts, specification models often become outdated once they lack proper maintenance. Hence, this phenomenon may hinder the application of model-based software engineering principles, such as model-based testing, model checking, and program comprehension.

This PhD Thesis improves upon the state of the art of model-based software engineering by introducing theoretical and experimental contributions to address automata model learning for evolving systems. Thus, it introduces and empirically evaluates solutions to the research problem retaken below:

Research Problem

Given an *evolving system* that has changed *over time* (*in space*) where its versioning scheme (variability model) is known, but version-specific FSMs (family models) are *unavailable* or *outdated*, how can we *efficiently* and *effectively* learn (family-based) *state machines* specifying its behavior?

The remainder of this chapter is organized as follows: The contributions to address the research problem of this PhD Thesis and research collaborations are revisited in Section 5.1. In Section 5.2, we present a list of works related to my PhD Thesis. In Section 5.3, we discuss the limitations of this PhD Thesis. Finally, in Section 5.4, we conclude this work enumerating a non-exhaustive list of future works and possible extensions to this PhD Thesis.

5.1 Contributions of this PhD Thesis

The overall contribution of this PhD Thesis is an extensive investigation of the problem of automata model learning in the setting of evolving systems that can incorporate software modifications *over time* and software variability *in space*. Particularly, we have addressed our research problem following three main branches refered to as `Learning to reuse`, `Learning from difference`, and `Learning by sampling`, respectively. We briefly summarize the branches of contributions of this PhD Thesis as follows:

1. **Learning to Reuse:** We have introduced the `partial-Dynamic` L^*_M algorithm, an adaptive technique that mitigates the cost for re-learning FSMs from evolving systems. We have empirically shown that our technique is less sensitive to the side-effects of *redundant* and *deprecated* sequences and more efficient than state-of-the art adaptive learning. This algorithm suports the efficient model-based analysis of systems that evolve *over time*. The `partial-Dynamic` L^*_M algorithm has been published as a regular paper at the 15th International Conference on integrated Formal Methods (iFM) held in Bergen, Norway (DAMASCENO; MOUSAVI; SIMAO, 2019b). Furthermore, an extended abstract presenting an overview and indicating future works for this PhD Thesis has been published and presented in the PhD Symposium of the iFM 2019 (DAMASCENO, 2019).

2. **Learning from Difference:** We have designed the $FFSM_{Diff}$ algorithm, an automated technique to learn family models by comparing, annotating, and merging product-specific FSM models. Using abstract representations of product lines from academic benchmarks, we have empirically shown that our algorithm can effectively combine families of product FSMs into succinct family models given as FFSMs, especially if there is high feature reuse among products. Therefore, our technique has been able to learn succinct family models out of product-specific FSMs efficiently, and include novel product-specific behavior into existing family models. A full paper discussing this contribution has been published at the 23rd International Systems and Software Product Line Conference held in Paris, France (DAMASCENO; MOUSAVI; SIMAO, 2019a).

3. **Learning by Sampling:** We have extended our understanding of $FFSM_{Diff}$ by incorporating product sampling into the process of family model learning. Using 105 FSMs derived from six product lines of academic benchmarks, we have shown that product sampling can lead to models as precise as those learned by exhaustive analysis. These results have been submitted as a journal paper for a Special Issue on "Configurable Systems" in the Empirical Software Engineering Journal (DAMASCENO; MOUSAVI; SIMAO, 2020).

Other publications

Apart from the **three published papers** (DAMASCENO; MOUSAVI; SIMAO, 2019b; DAMASCENO, 2019; DAMASCENO; MOUSAVI; SIMAO, 2019a) and the journal manuscript submitted to the Empirical Software Engineering Journal (DAMASCENO; MOUSAVI; SIMAO, 2020), the author of this PhD Thesis has co-authored other **five papers** in collaboration with researchers from the **University of Leicester**, where he has spent *one year and two months* as a visiting PhD student researcher; and the **Federal University of Pará**, where he has formely attended an Institutional Scientific Initiation Scholarship Program (PIBIC) and earned his Bsc degree in Computer Science. The five papers are listed below with brief descriptions of their content.

4. **Trusted Autonomous Vehicles: an Interactive Exhibit** This *conference paper* reports a collaboration with the University of Leicester on the design of an *interactive exhibit* to illustrate basic technologies employed in autonomous vehicles and principles to ensure their quality (Araujo *et al.*, 2019). The authors also report on a science outreach involving this exhibit at the Royal Society Summer Science Exhibition 2019, held in London, UK.

5. **Similarity testing for role-based access control systems:** This manuscript is a *journal paper* resulting from my MSc dissertation. In this paper, Damasceno, Masiero and Simao (2018) have introduced and empirically evaluated a similarity-based test prioritization criteria for Role-Based Access Control (RBAC) systems.

6. **Data Analysis of Multiplex Sequencing at SOLiD Platform:** This *journal paper* is a publication in collaboration with my former PIBIC supervisors from the Federal University of Pará. In this paper, Lobato *et al.* (2018) have introduced a probabilistic model to analyze and characterize the reliability of DNA sequencing datasets.

7. **A parallel algorithm for test prioritization based on similarity using OpenMPI:** This *short paper* reports a small experiment comparing test prioritization algorithms implemented using OpenMPI (DAMASCENO; SOUZA; SIMAO, 2017). These experiments have been performed in a course of *Parallel Programming* that I attended during my PhD.

8. **Evaluating Test Characteristics and Effectiveness of FSM-based Testing Methods on RBAC Systems:** This *conference paper* is another result of my MSc dissertation. In this paper, Damasceno, Masiero and Simao (2016) have analyzed the characteristics test suites generated by FSM-based testing methods for RBAC policies specified as FSM models. This work has earned the **3rd Best Paper Award** at the 30th Brazilian Symposium on Software Engineering 2016.

5.2 Related work

In this section, we discuss the related works and how this PhD Thesis can be helpful in their respective contexts. Studies related to this PhD Thesis are in the fields of state-machine model learning, product sampling, family-based analysis, comparison of state models, reverse engineering feature models, and product line evolution.

State-machine learning

State-machine learning, also known as automata learning (ANGLUIN, 1987), has been a popular technique in software analysis and testing. State-machine learning algorithms have been harnessed for black-box model checking (PELED; VARDI; YANNAKAKIS, 1999), analyzing network protocols (AARTS *et al.*, 2012; FITERĂU-BROŞTEAN; HOWAR, 2017), software evolution (HUNGAR; NIESE; STEFFEN, 2003; DE RUITER; POLL, 2015), automatic test generation (RAFFELT *et al.*, 2009), and generalization of failure models (CHAPMAN *et al.*, 2015; KUNZE *et al.*, 2016). For an overview of model learning, we refer the interested reader to Irfan, Oriat and Groz (2013), Stevenson and Cordy (2014), and Aichernig *et al.* (2018).

The problem of learning models from evolving systems becomes more complex as it has to cope with products that may have their own models, requirements and code. This PhD Thesis improves upon the state of the art by supporting the tasks of learning models from systems evolving *over time* and incorporating variability *in space*.

Product sampling for software product lines

Due to the number of valid configurations that usually grows exponentially with the number of features, the exhaustive analysis of SPLs is impractical (THÜM *et al.*, 2014a). To alleviate this issue, sampling techniques have been used to select products systematically covering the behavior of SPLs and hence, reveal faults in other products (PERROUIN *et al.*, 2010).

According to Varshosaz *et al.* (2018), product sampling techniques often rely on feature models (KANG *et al.*, 1990) and SAT solvers (BERRE; PARRAIN, 2010) to distinguish valid from invalid configurations (BENAVIDES; SEGURA; RUIZ-CORTÉS, 2010). In order to sample product configurations, techniques can use meta-heuristics (e.g., genetic algorithms (ENSAN; BAGHERI; GAŠEVIĆ, 2012; LOPEZ-HERREJON *et al.*, 2014)), coverage criteria (e.g., T-wise (PERROUIN *et al.*, 2010)), manual selection and semi-automatic selection.

In this PhD Thesis, we employed product sampling to generate subsets of valid configurations satisfying the T-wise coverage criteria. We used the Chvatal algorithm (CHVATAL, 1979) implemented in the FeatureIDE workbench (THÜM *et al.*, 2014b) for product sampling. This algorithm has been adapted by Johansen, Haugen and Fleurey (2011) to select subsets of products covering T-wise feature combinations.

Family-based analysis of SPLs

Family-based analysis operates on domain artifacts and incorporates knowledge about valid feature combinations, given a feature model. Thus, not every individual product has to be analyzed (THÜM *et al.*, 2014a), as opposed to traditional analysis strategies that are influenced by the number of valid feature combinations (BRABRAND *et al.*, 2012). To achieve this goal, family-based analysis techniques rely on *family models*. For an overview of techniques for family-model analysis, testing and modeling, we refer the reader to recent surveys such as Thüm *et al.* (2014a), Benduhn *et al.* (2015), and Beohar, Varshosaz and Mousavi (2016).

Family models have been exploited as a theoretical foundation to perform efficient model-based testing of SPLs (ATLEE *et al.*, 2015; BEOHAR; MOUSAVI, 2016) and family model checking (SABOURI; KHOSRAVI, 2013; TER BEEK; DE VINK; WILLEMSE, 2017). Moreover, they have been used to automate the generation of specifications for individual products (ASIRELLI *et al.*, 2012), to efficiently validate families of products (FRAGAL; SIMAO; MOUSAVI, 2017), and to describe fine-grained differences among product variants (SCHAEFER *et al.*, 2010).

The contributions of this PhD Thesis are complementary to the techniques aforementioned as it can give insights about how to optimize family model learning to large SPLs. Our techniques have been discussed in terms of FFSMs, but they can be extended to other family-based notations (CLASSEN *et al.*, 2013; BEOHAR; MOUSAVI, 2014).

Comparison of state-based models

The comparison of FSMs is important for software engineering tasks (WALKINSHAW; BOGDANOV, 2013) such as conformance testing (BROY *et al.*, 2005), and performance analysis of state-machine learning techniques (ANGLUIN, 1987; VAANDRAGER, 2017). In this PhD Thesis, we aimed at finding the differences between two models (i.e., FSM-FSM, FFSM-FSM) and labeling commonalities and variability with feature constraints. Studies related to ours have been conducted by Walkinshaw and Bogdanov (2013) and Nejati *et al.* (2012).

Walkinshaw and Bogdanov (2013) have designed and evaluated two approaches to compute the precise difference between labeled transition systems (LTS) in terms of their *language* and *structure*. To compare the *language* of two state-based models, they have proposed an approach based on the proportion of test sequences (CHOW, 1978; VASILEVSKII, 1973) that are classified in the same way by two models M_r and M_u. Thus, performance metrics (e.g., precision, recall, and F-measure) can be used to compare the languages of LTS models. A major issue in comparing the language of FSMs is that some minor differences may mask structural similarities. To address this issue, the authors have modeled the problem of structural comparison of FSMs as a system of linear equations. These two approaches for structural and language similarity are *complementary* as two models may have similar state transition structure, but

completely different languages, or vice-versa.

Nejati *et al.* (2012) presented an approach for matching and merging Statecharts (HAREL, 1987). Their approach relies on two operators for matching and merging transitions. The latter uses static and behavioral properties to match state pairs. The former produces a combined model in which variant behaviors are parameterized using guards on their transitions where temporal properties are preserved. The authors have developed a tool that implements both Match and Merge operators and allows their seamless application. Finally, they have shown that that relying on both operators produces higher precision than relying on them independently.

In this PhD Thesis, we introduced an approach for comparing product FSMs and building family models inspired by Walkinshaw and Bogdanov (2013). To achieve this, we extended their approach by annotating common and distinct states with feature constraints and evaluating how product sampling can reduce the costs for recovering family models. Product-lines may have an exponential number of valid configurations. Thus, sampling techniques can help reduce the effort required to recover family models.

Reverse engineering feature models

Feature models play a central role in the variability management for SPLs (POHL; BÖCKLE; VAN DER LINDEN, 2005). Feature models can be used to detect invalid relationships or product configurations, core or dead features, redundancies, and enumerate or quantify all valid products of an SPL (BENAVIDES; SEGURA; RUIZ-CORTÉS, 2010). Unfortunately, software variants are often created in unstructured ways and may lack feature models as their construction is time-consuming (HASLINGER; LOPEZ-HERREJON; EGYED, 2011).

Several approaches have been proposed to automatically build feature models from sets of product configurations (HASLINGER; LOPEZ-HERREJON; EGYED, 2011; RYSSEL; PLOENNIGS; KABITZSCH, 2011; AL-MSIE'DEEN *et al.*, 2014). Approaches based on Formal Concept Analysis have been promising as they can detect interdependencies and hierarchies between features (AL-MSIE'DEEN *et al.*, 2014).

This PhD Thesis addresses the task of "reverse engineering" family models through exhaustive learning and product sampling. In this PhD thesis, there is an assumption that a feature model is known a priori for the SPL. However, we believe that our strategy can be extended to cope with non-existent feature models and learn family and feature models at once. Further investigations on combining feature and family model learning are still required.

Software product line evolution

The tasks of product-line reengineering and refactoring are vital to the maintenance and evolution of their software products. For an overview of product-line evolution, refactoring and

reengineering, we refer the readers to Laguna and Crespo (2013), Fenske, Thüm and Saake (2013), and Marques *et al.* (2019).

A variety of artifacts have been studied in SPL evolution, but feature models are by far the most researched ones (MARQUES *et al.*, 2019). Recent studies have shown that there is a need for reengineering approaches tailored for agile processes (MARQUES *et al.*, 2019), and migration of SPL paradigms (LAGUNA; CRESPO, 2013). Several studies have investigated model learning techniques to cope with traditional software evolution and regression testing (SERY; FEDYUKOVICH; SHARYGINA, 2015; HUISTRA; MEIJER; VAN DE POL, 2018). However, to the best of our knowledge, there are no works investigating model learning in the setting of SPLs. Combined with state-machine learning (ANGLUIN, 1987), we believe that our algorithm can support model-based regression testing in SPLs (RUNESON; ENGSTRöM, 2012) and family model checking (SABOURI; KHOSRAVI, 2013; TER BEEK; DE VINK; WILLEMSE, 2017) in agile processes (NEUBAUER *et al.*, 2012).

5.3 Research limitations and Assumptions

In this section, we describe the assumptions and limitations of the contributions in this PhD Thesis. It should be noticed that the specific limitations of each chapter of this PhD Thesis have been already reported in their respective sections of threats to validity. Thus, this section focuses on broad limitations and assumptions that may support future works. Possible extensions of this PhD Thesis are reported in the next final section.

In Chapter 3, we introduced an adaptive learning algorithm for evolving systems referred to as `partial-Dynamic` $\mathtt{L}^*_{\mathtt{M}}$. In this study, the following assumptions have been required:

1. **There is a MAT:** As we rely on the principle of adaptive learning, the *MAT framework* (ANGLUIN, 1987) shall be applicable to build black-box state machines by providing inputs and observing outputs (VAANDRAGER, 2017).

2. **SULs can be represented as Mealy machines:** Our evolving system constitutes a set of *releases* modeled as complete deterministic Mealy machines (GILL, 1962). This is reasonable as it is a suitable abstraction for representing reactive systems (GILL, 1962; BROY *et al.*, 2005; CHOW, 1978) and the semantics of richer notations (HAREL, 1987; CASSEL FALK HOWAR, 2015).

3. **There is a versioning scheme:** We require our SUL to be tracked using a versioning scheme (PRESTON-WERNER, 2013; COGHLAN; STUFFT, 2013) that indicates precedence/antecedence relationships between versions. This is acceptable as it is common practice in software projects (SPINELLIS, 2005; DECAN; MENS, 2019).

4. **There are some reference FSMs:** We rely on the assumption that previous versions shall eventually have their respective FSM specifications while precedent releases may have models outdated or unavailable (WALKINSHAW, 2013). Thus, by means of our technique, we steer learning to preserved states and avoid irrelevant queries while re-building models from systems that evolve *over time*.

In Chapter 4, we leveraged the concept of model learning for software product lines by introducing the $FFSM_{Diff}$ algorithm to learn family models and analyzing the incorporation of product sampling in this task. The following assumptions have been required:

1. **Products can be represented as Mealy machines:** As in our previous investigation, we have assumed that product instances can be modeled as complete deterministic Mealy machines (GILL, 1962). If a product-specific FSM is unavailable, model learning can be used address this lack of specification. However, our technique can still be extended to other state-based notations, such as transition systems (CLASSEN *et al.*, 2013), timed automata (CORDY *et al.*, 2012), and statecharts (FRAGAL; SIMAO; MOUSAVI, 2019).

2. **Products have their respective configurations known:** To support the process of state and transition annotation, we assume that product configurations are known.

3. **Products have a common variability model that is known *a priori*:** To optimize the process of state and transition annotation, we assume that there is a known variability model that is shared among the products under learning.

4. **Product sampling collectively covers the product lines' behavior:** This is a common assumption for product sampling techniques (VARSHOSAZ *et al.*, 2018). If this is not the case, then a possible alternative could be the adoption of an iterative `CQ` oracle that alternates between testing and learning. Thus, partial family models learned from subsets of configurations can be validated within unseen products.

5.4 Future work and possible extensions

Some of the possible extensions and future works to the research contributions of this PhD Thesis include: Adaptive strategies for discrimination tree-based learning algorithms, Active family model learning, Incremental Configuration Queries, and Fingerprinting evolving systems. The future work are discussed in the next sections and illustrated in Figure 33. Each of these future work are respectively indicated using *double-dotted-dashed*, *dotted*, *dashed*, and *single-dotted-dashed* lines. Arrows arrows and extra boxes indicate the future work and how they are associated with the Research Problems addressed in this PhD Thesis.

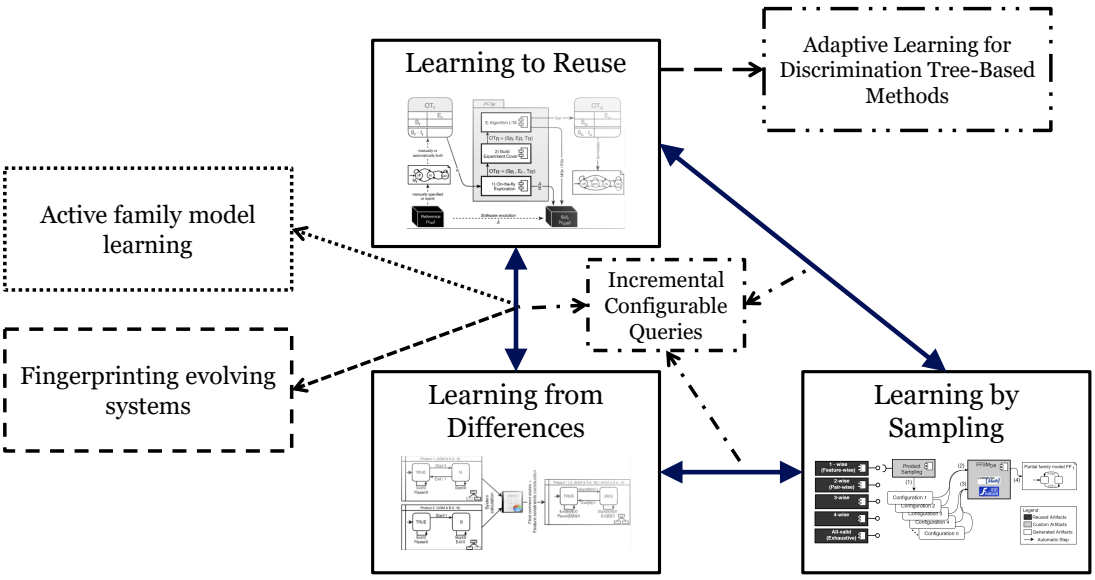


Figure 33 – Future Work and their relationship with the Research Problems of this PhD Thesis

Source: Elaborated by the author.

Adaptive learning for discrimination tree-based algorithms

One branch of future research consists of extending the principles of adaptive learning to *discrimination tree*-based algorithms, such as the TTT algorithm (ISBERNER; HOWAR; STEFFEN, 2014b). Discrimination tree-based algorithms differ from observation table-based algorithms, such as the L^*_M (SHAHBAZ; GROZ, 2009), by their redundancy-free and improved space complexity.

A possible way of extending such principles could be employing sequential Equivalence Oracles (YANG *et al.*, 2019), where execution traces from the SUL feed a passive learning algorithm and models passively learned are used as candidate hypotheses from which `EQ`s can be derived. This approach could be easily applied to evolving systems and make the concept of *adaptive learning* algorithm-independent. The relationship of this future work with the research problems tackled in this PhD thesis is depicted in Figure 33 using *double-dotted-dashed* lines.

Active family model learning

Our two algorithms improve upon the state-of-the-art of automata learning and family model learning as both independent and complementary strategies. One possibility for upon the combination of these two strategies stands on an idea we vision as the *active family model learning* framework. In Figure 34, we show our vision of active family model learning.

Our vision of *active family model learning* stands for a framework where SPLs can have their family models incrementally harvested by re-using *partial family models*, i.e., family models describing subsets of valid product instances from SPLs, to steer an active model learning and testing procedure. We envisage that such variability-aware model learning framework will

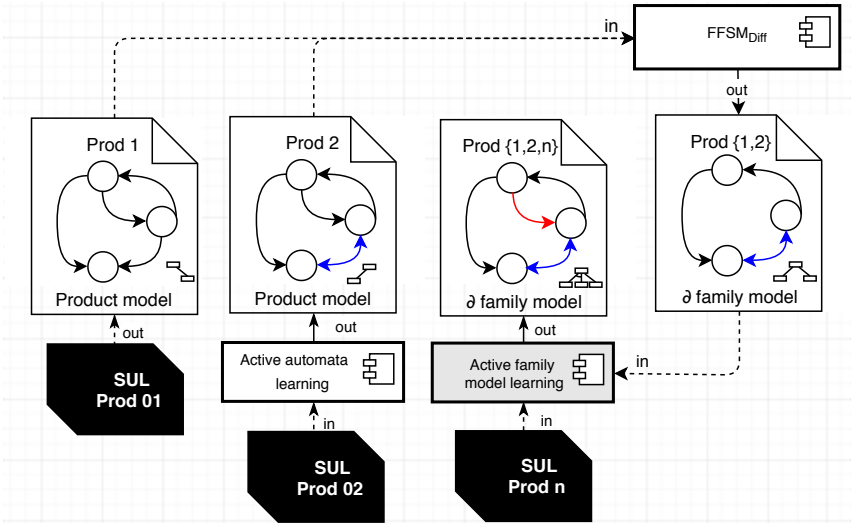


Figure 34 – Active family model learning

Source: Damasceno (2019).

scale better than exhaustively and independently applying adaptive learning to product instances or software releases. Concepts such as configurable test cases (FRAGAL, 2017) could support the process of deriving EQs and steer learning to conditional states shared among different product configurations. The relationship of this future work with the research problems tackled in this PhD thesis is depicted in Figure 33 using *dashed* lines.

Incremental Configurable Queries

To complement the idea of active model learning, the concept of configurable queries that we have discussed in terms of the Chvatal algorithm (CHVATAL, 1979) for T-wise sampling could be improved with incremental sampling techniques (AL-HAJJAJI *et al.*, 2016). Using the IncLing algorithm (AL-HAJJAJI *et al.*, 2016), users can incorporate products in the sample and extends this set towards a representative set of products satisfying some T-wise criteria (PERROUIN *et al.*, 2010). Thus, subsets of product models that have already been incorporated in some partial family model could be discarded from the learning process. Alternatively, search-based techniques could be used for product sampling (ENSAN; BAGHERI; GAŠEVIĆ, 2012) and matching and merging large product models (AL-KHIATY; AHMED, 2017). The relationship of this future work with the research problems tackled in this PhD thesis is depicted in Figure 33 using *single-dotted-dashed* lines.

Fingerprinting evolving systems

Another possible application to our model learning approaches is software fingerprinting (PELLEGRINO *et al.*, 2017). Software fingerprinting is an important task in cybersecurity as it supports authorship attribution, clone detection, library identification, and vulnerability analysis,

to name a few possible applications (ALRABAEE *et al.*, 2020).

In the context of evolving systems, software versions could be modeled in terms of a feature model with features connected by `exclusive-OR` relationships denoting its versions. Once the models of an evolving system have been unified into a featured FSM, this family model could play the role of an oracle to identify what versions (features) could be implemented in a black-box evolving system of unknown version (configuration). The relationship of this future work with the research problems tackled in this PhD thesis is depicted in Figure 33 using *dashed* lines.

BIBLIOGRAPHY

AARTS, F.; FITERAU-BROSTEAN, P.; KUPPENS, H.; VAANDRAGER, F. Learning register automata with fresh value generation. In: ____. **Theoretical Aspects of Computing - ICTAC 2015: 12th International Colloquium, Cali, Colombia, October 29-31, 2015, Proceedings**. Cham: Springer International Publishing, 2015. p. 165–183. ISBN 978-3-319-25150-9. Available: <http://dx.doi.org/10.1007/978-3-319-25150-9_11>. Citation on page 39.

AARTS, F.; HEIDARIAN, F.; KUPPENS, H.; OLSEN, P.; VAANDRAGER, F. Automata learning through counterexample guided abstraction refinement. In: ____. **FM 2012: Formal Methods: 18th International Symposium, Paris, France, August 27-31, 2012. Proceedings**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. p. 10–27. ISBN 978-3-642-32759-9. Available: <https://doi.org/10.1007/978-3-642-32759-9_4>. Citations on pages 25, 39, and 102.

AFSHAN, S.; MCMINN, P.; STEVENSON, M. Evolving readable string test inputs using a natural language model to reduce human oracle cost. In: **2013 IEEE Sixth International Conference on Software Testing, Verification and Validation**. [S.l.: s.n.], 2013. p. 352–361. ISSN 2159-4848. Citation on page 35.

AICHERNIG, B. K.; MOSTOWSKI, W.; MOUSAVI, M. R.; TAPPLER, M.; TAROMIRAD, M. Model learning and model-based testing. In: BENNACEUR, A.; HÄHNLE, R.; MEINKE, K. (Ed.). **Machine Learning for Dynamic Software Analysis: Potentials and Limits: International Dagstuhl Seminar 16172, Dagstuhl Castle, Germany, April 24-27, 2016, Revised Papers**. Cham: Springer International Publishing, 2018. p. 74–100. ISBN 978-3-319-96562-8. Available: <https://doi.org/10.1007/978-3-319-96562-8_3>. Citations on pages 24, 25, 27, 38, 70, 97, and 102.

AL-HAJJAJI, M.; KRIETER, S.; THüM, T.; LOCHAU, M.; SAAKE, G. Incling: Efficient product-line testing using incremental pairwise sampling. In: **Proceedings of the 2016 ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences**. New York, NY, USA: Association for Computing Machinery, 2016. (GPCE 2016), p. 144–155. ISBN 9781450344463. Available: <https://doi.org/10.1145/2993236.2993253>. Citation on page 108.

AL-HAJJAJI, M.; LITY, S.; LACHMANN, R.; THÜM, T.; SCHAEFER, I.; SAAKE, G. Delta-oriented product prioritization for similarity-based product-line testing. In: **2017 IEEE/ACM 2nd International Workshop on Variability and Complexity in Software Design (VACE)**. [S.l.: s.n.], 2017. p. 34–40. Citations on pages 51, 82, and 84.

AL-KHIATY, M. A.-R.; AHMED, M. Matching UML class diagrams using a hybridized greedy-genetic algorithm. In: **2017 12th International Scientific and Technical Conference on Computer Sciences and Information Technologies (CSIT)**. IEEE, 2017. p. 161–166. ISBN 978-1-5386-1638-3 978-1-5386-1639-0. Available: <http://ieeexplore.ieee.org/document/8098759/>. Citation on page 108.

AL-MSIE'DEEN, R. A.; HUCHARD, M.; SERIAI, A.-D.; URTADO, C.; VAUTTIER, S. Reverse Engineering Feature Models from Software Configurations using Formal Concept Analysis.

In: BERTET, K.; RUDOLPH, S. (Ed.). **CLA: Concept Lattices and their Applications**. [S.l.: s.n.], 2014. v. 1252, p. 95–106. Citations on pages 97 and 104.

ALRABAEE, S.; DEBBABI, M.; SHIRANI, P.; WANG, L.; YOUSSEF, A.; RAHIMIAN, A.; NOUH, L.; MOUHEB, D.; HUANG, H.; HANNA, A. **Binary Code Fingerprinting for Cybersecurity: Application to Malicious Code Fingerprinting**. Springer International Publishing, 2020. (Advances in Information Security, v. 78). ISBN 978-3-030-34237-1 978-3-030-34238-8. Available: <http://doi.org/10.1007/978-3-030-34238-8>. Citation on page 109.

AMMANN, P.; OFFUTT, J. **Introduction to Software Testing**. 1. ed. New York, NY, USA: Cambridge University Press, 2008. ISBN 0521880386, 9780521880381. Citations on pages 33 and 34.

ANGLUIN, D. Learning regular sets from queries and counterexamples. **Information and Computation**, v. 75, n. 2, p. 87 – 106, 1987. ISSN 0890-5401. Available: <http://www.sciencedirect.com/science/article/pii/0890540187900526>. Citations on pages 24, 25, 33, 38, 39, 40, 43, 47, 55, 76, 80, 102, 103, and 105.

Apache. **Commons Math: The Apache Commons Mathematics Library**. 2016. <http://commons.apache.org/proper/commons-math/>. [Online; accessed 28-Mar-2019]. Citations on pages 29, 82, and 85.

APEL, S.; KOLESNIKOV, S.; SIEGMUND, N.; KÄSTNER, C.; GARVIN, B. Exploring feature interactions in the wild: The new feature-interaction challenge. In: **Proceedings of the 5th International Workshop on Feature-Oriented Software Development**. New York, NY, USA: ACM, 2013. (FOSD '13), p. 1–8. ISBN 978-1-4503-2168-6. Available: <http://doi.acm.org/10.1145/2528265.2528267>. Citations on pages 50 and 73.

Araujo, H. L. S.; Damasceno, C. D. N.; Dimitrova, R.; Kefalidou, G.; Mehtarizadeh, M.; Mousavi, M. R.; Onime, J.; Ringert, J. O.; Rojas, J. M.; Verdezoto, N. X.; Wali, S. Trusted autonomous vehicles: an interactive exhibit. In: **2019 IEEE International Conferences on Ubiquitous Computing Communications (IUCC) and Data Science and Computational Intelligence (DSCI) and Smart Computing, Networking and Services (SmartCNS)**. [S.l.]: IEEE, 2019. p. 386–393. [Online]. Citation on page 101.

ARCURI, A.; BRIAND, L. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In: **Proceedings of the 33rd International Conference on Software Engineering**. NY, USA: ACM, 2011. (ICSE '11), p. 1–10. ISBN 978-1-4503-0445-0. Citations on pages 62 and 82.

ASIRELLI, P.; TER BEEK, M. H.; FANTECHI, A.; GNESI, S. A compositional framework to derive product line behavioural descriptions. In: ____. **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**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. p. 146–161. ISBN 978-3-642-34026-0. Available: <http://dx.doi.org/10.1007/978-3-642-34026-0_12>. Citation on page 103.

ATLEE, J. M.; BEIDU, S.; FAHRENBERG, U.; LEGAY, A. Merging Features in Featured Transition Systems. In: **Proceedings of the 12th Workshop on Model-Driven Engineering, Verification and Validation co-located with ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems**. Ottawa, Canada: [s.n.], 2015. (CEUR

Workshop Proceedings, v. 1514), p. 38–43. Available: <https://hal.inria.fr/hal-01237661>. Citations on pages 73 and 103.

BAIER, C.; KATOEN, J.-P. **Principles of Model Checking (Representation and Mind Series)**. [S.l.]: The MIT Press, 2008. ISBN 026202649X, 9780262026499. Citations on pages 26, 50, 73, and 82.

BAINCZYK, A.; SCHIEWECK, A.; ISBERNER, M.; MARGARIA, T.; NEUBAUER, J.; STEFFEN, B. Alex: Mixed-mode learning of web applications at ease. In: ____. **Leveraging Applications of Formal Methods, Verification and Validation: Discussion, Dissemination, Applications: 7th International Symposium, ISoLA 2016, Imperial, Corfu, Greece, October 10-14, 2016, Proceedings, Part II**. Cham: Springer International Publishing, 2016. p. 655–671. ISBN 978-3-319-47169-3. Available: <https://doi.org/10.1007/978-3-319-47169-3_51>. Citation on page 25.

BARR, E. T.; HARMAN, M.; MCMINN, P.; SHAHBAZ, M.; YOO, S. The oracle problem in software testing: A survey. **IEEE Transactions on Software Engineering**, n. 5, p. 507–525, May 2015. ISSN 0098-5589. Citations on pages 34 and 35.

BEKRAR, S.; BEKRAR, C.; GROZ, R.; MOUNIER, L. Finding software vulnerabilities by smart fuzzing. In: **Proceedings of the 2011 Fourth IEEE International Conference on Software Testing, Verification and Validation**. Washington, DC, USA: IEEE Computer Society, 2011. (ICST '11), p. 427–430. ISBN 978-0-7695-4342-0. Available: <http://dx.doi.org/10.1109/ICST.2011.48>. Citation on page 35.

BENAVIDES, D.; SEGURA, S.; RUIZ-CORTÉS, A. Automated analysis of feature models 20 years later: A literature review. **Information Systems**, v. 35, n. 6, p. 615 – 636, 2010. ISSN 0306-4379. Available: <http://www.sciencedirect.com/science/article/pii/S0306437910000025>. Citations on pages 29, 48, 49, 51, 74, 78, 97, 102, and 104.

BENDUHN, F. **Representing Variability in Product Lines: A Survey of Modeling and Specification Techniques**. Master's Thesis (Master's Thesis) — University of Magdeburg, 2014. Available: <http://wwwiti.cs.uni-magdeburg.de/iti_db/publikationen/ps/auto/thesisBenduhn14.pdf>. Citation on page 28.

BENDUHN, F.; THÜM, T.; LOCHAU, M.; LEICH, T.; SAAKE, G. A survey on modeling techniques for formal behavioral verification of software product lines. In: **Proceedings of the Ninth International Workshop on Variability Modelling of Software-intensive Systems**. New York, NY, USA: ACM, 2015. (VaMoS '15), p. 80:80–80:87. ISBN 978-1-4503-3273-6. Available: <http://doi.acm.org/10.1145/2701319.2701332>. Citations on pages 29, 73, 75, and 103.

BEOHAR, H.; MOUSAVI, M. R. Input-output conformance testing based on featured transition systems. In: **Proceedings of the 29th Annual ACM Symposium on Applied Computing - SAC '14**. New York, New York, USA: ACM Press, 2014. p. 1272–1278. ISBN 9781450324694. Available: <http://doi.acm.org/10.1145/2554850.2554949>. Citations on pages 26, 29, 73, 75, and 103.

____. Input–output conformance testing for software product lines. **Journal of Logical and Algebraic Methods in Programming**, v. 85, n. 6, p. 1131 – 1153, 2016. ISSN 2352-2208. {NWPT} 2013. Available: <http://www.sciencedirect.com/science/article/pii/S2352220816301171>. Citations on pages 73 and 103.

BEOHAR, H.; VARSHOSAZ, M.; MOUSAVI, M. R. Basic behavioral models for software product lines: Expressiveness and testing pre-orders. **Science of Computer Programming**, v. 123, p. 42 – 60, 2016. ISSN 0167-6423. Available: <http://www.sciencedirect.com/science/article/pii/S0167642315001288>. Citations on pages 75 and 103.

BERG, T.; RAFFELT, H. 19 model checking. In: ____. **Model-Based Testing of Reactive Systems: Advanced Lectures**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. p. 557–603. ISBN 978-3-540-32037-1. Available: <http://dx.doi.org/10.1007/11498490_25>. Citations on pages 24, 33, and 38.

BERRE, D. L.; PARRAIN, A. The SAT4J library, Release 2.2, System Description. **Journal on Satisfiability, Boolean Modeling and Computation**, IOS Press, v. 7, p. 59–64, 2010. Citations on pages 26, 49, 51, 73, 82, 96, and 102.

BERTOLINO, A.; DAOUDAGH, S.; EL KATEB, D.; HENARD, C.; LE TRAON, Y.; LONETTI, F.; MARCHETTI, E.; MOUELHI, T.; PAPADAKIS, M. Similarity testing for access control. **Information and Software Technology**, v. 58, p. 355 – 372, 2015. ISSN 0950-5849. Citation on page 51.

BEUCHE, D.; SCHULZE, M.; DUVIGNEAU, M. When 150% is too much: Supporting product centric viewpoints in an industrial product line. In: **Proceedings of the 20th International Systems and Software Product Line Conference**. New York, NY, USA: Association for Computing Machinery, 2016. (SPLC '16), p. 262–269. ISBN 9781450340502. Available: <https://doi.org/10.1145/2934466.2934493>. Citations on pages 26 and 73.

BINDER, R. V. **Testing Object-oriented Systems: Models, Patterns, and Tools**. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999. ISBN 0-201-80938-9. Citations on pages 23, 33, and 55.

BOLLIG, B.; HABERMEHL, P.; KERN, C.; LEUCKER, M. Angluin-style learning of nfa. In: **Proceedings of the 21st International Jont Conference on Artifical Intelligence**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2009. (IJCAI'09), p. 1004–1009. Available: <http://dl.acm.org/citation.cfm?id=1661445.1661605>. Citation on page 39.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **The Unified Modeling Language User Guide, The 2nd Edition**. [S.l.]: Addison-Wesley Professional, 2005. (Addison-Wesley Object Technology Series). ISBN 0321267974. Citation on page 23.

BRABRAND, C.; RIBEIRO, M.; TOLêDO, T.; BORBA, P. Intraprocedural dataflow analysis for software product lines. In: **Proceedings of the 11th Annual International Conference on Aspect-oriented Software Development**. New York, NY, USA: ACM, 2012. (AOSD '12), p. 13–24. ISBN 978-1-4503-1092-5. Available: <http://doi.acm.org/10.1145/2162049.2162052>. Citations on pages 52 and 103.

BROY, M.; JONSSON, B.; KATOEN, J.-P.; LEUCKER, M.; PRETSCHNER, A. **Model-Based Testing of Reactive Systems: Advanced Lectures**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. 557–603 p. ISBN 978-3-540-32037-1. Available: <http://dx.doi.org/10.1007/11498490_25>. Citations on pages 24, 33, 34, 35, 37, 38, 55, 59, 74, 82, 103, and 105.

BURDY, L.; CHEON, Y.; COK, D. R.; ERNST, M. D.; KINIRY, J. R.; LEAVENS, G. T.; LEINO, K. R. M.; POLL, E. An overview of jml tools and applications. **International Journal on Software Tools for Technology Transfer**, v. 7, n. 3, p. 212–232, Jun 2005. ISSN 1433-2787. Available: <https://doi.org/10.1007/s10009-004-0167-4>. Citation on page 35.

CARTAXO, E. G.; MACHADO, P. D. L.; NETO, F. G. O. On the use of a similarity function for test case selection in the context of model-based testing. **Software Testing, Verification and Reliability**, John Wiley & Sons, Ltd., v. 21, n. 2, p. 75–100, 2011. ISSN 1099-1689. Citation on page 51.

CASSEL FALK HOWAR, B. J. S. Ralib: A learnlib extension for inferring efsms. **DIFTS 2015**, 2015. Available: <http://www.faculty.ece.vt.edu/chaowang/difts2015/papers/paper_5.pdf>. Citations on pages 24, 38, 39, and 105.

CASSEL, S.; HOWAR, F.; JONSSON, B.; STEFFEN, B. Active learning for extended finite state machines. **Formal Aspects of Computing**, v. 28, n. 2, p. 233–263, Apr 2016. ISSN 1433-299X. Available: <https://doi.org/10.1007/s00165-016-0355-5>. Citation on page 39.

CHAKI, S.; CLARKE, E.; SHARYGINA, N.; SINHA, N. Verification of evolving software via component substitutability analysis. **Formal Methods in System Design**, v. 32, n. 3, p. 235–266, 2008. ISSN 1572-8102. Available: <https://doi.org/10.1007/s10703-008-0053-x>. Citations on pages 25, 27, 28, 41, 42, 43, 44, 55, 59, 60, and 70.

CHAPMAN, M.; CHOCKLER, H.; KESSELI, P.; KROENING, D.; STRICHMAN, O.; TAUTSCHNIG, M. Learning the language of error. In: ____. **Automated Technology for Verification and Analysis: 13th International Symposium, ATVA 2015, Shanghai, China, October 12-15, 2015, Proceedings**. Cham: Springer International Publishing, 2015. p. 114–130. ISBN 978-3-319-24953-7. Available: <http://dx.doi.org/10.1007/978-3-319-24953-7_9>. Citations on pages 25 and 102.

CHOW, T. S. Testing software design modeled by finite-state machines. **IEEE Transactions on Software Engineering**, IEEE Press, Piscataway, NJ, USA, v. 4, n. 3, p. 178–187, May 1978. ISSN 0098-5589. Citations on pages 24, 37, 38, 40, 103, and 105.

CHVATAL, V. A greedy heuristic for the set-covering problem. **Mathematics of Operations Research**, INFORMS, v. 4, n. 3, p. 233–235, 1979. ISSN 0364765X, 15265471. Available: <http://www.jstor.org/stable/3689577>. Citations on pages 83, 102, and 108.

CLASSEN, A. Other, **Modelling with FTS: a Collection of Illustrative Examples**. 2010. Available: <https://researchportal.unamur.be/en/publications/modelling-with-fts-a-collection-of-illustrative-examples>. Citations on pages 29, 30, 74, 75, 86, and 87.

CLASSEN, A.; CORDY, M.; SCHOBBENS, P. Y.; HEYMANS, P.; LEGAY, A.; RASKIN, J. F. Featured transition systems: Foundations for verifying variability-intensive systems and their application to ltl model checking. **IEEE Transactions on Software Engineering**, v. 39, n. 8, p. 1069–1089, Aug 2013. ISSN 0098-5589. Citations on pages 26, 28, 29, 73, 75, 88, 89, 103, and 106.

CLEMENTS, P. C.; NORTHROP, L. **Software Product Lines: Practices and Patterns**. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001. (SEI Series in Software Engineering). ISBN 0-201-70332-7. Citations on pages 26, 28, 29, 34, and 48.

COGHLAN, N.; STUFFT, D. **PEP 440 – Version Identification and Dependency Specification**. 2013. <https://www.python.org/dev/peps/pep-0440/>. [Online; accessed 21-Feb-2020]. Citation on page 105.

CORDY, M.; SCHOBBENS, P.-Y.; HEYMANS, P.; LEGAY, A. Behavioural modelling and verification of real-time software product lines. In: **Proceedings of the 16th International Software Product Line Conference - Volume 1**. New York, NY, USA: Association for Computing Machinery, 2012. (SPLC '12), p. 66–75. ISBN 9781450310949. Available: <https://doi.org/10.1145/2362536.2362549>. Citation on page 106.

DAMASCENO, C. D. N. Learning from families: Inferring behavioral variability from software product lines. In: **PhD Symposium at Integrated Formal Methods**. [S.l.: s.n.], 2019. Citations on pages 30, 100, 101, and 108.

DAMASCENO, C. D. N.; MASIERO, P. C.; SIMAO, A. Evaluating test characteristics and effectiveness of fsm-based testing methods on rbac systems. In: **Proceedings of the 30th Brazilian Symposium on Software Engineering**. New York, NY, USA: ACM, 2016. (SBES '16), p. 83–92. ISBN 978-1-4503-4201-8. Available: <http://doi.acm.org/10.1145/2973839.2973849>. Citation on page 101.

DAMASCENO, C. D. N.; MASIERO, P. C.; SIMAO, A. Similarity testing for role-based access control systems. **Journal of Software Engineering Research and Development**, v. 6, n. 1, p. 1, 2018. ISSN 2195-1721. Available: <https://doi.org/10.1186/s40411-017-0045-x>. Citations on pages 51 and 101.

DAMASCENO, C. D. N.; MOUSAVI, M. R.; SIMAO, A. 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 - Volume A**. New York, NY, USA: Association for Computing Machinery, 2019. (SPLC '19), p. 52–63. ISBN 9781450371384. Available: <https://doi.org/10.1145/3336294.3336307>. Citations on pages 30, 75, 79, 84, 85, 86, 87, 93, 94, 96, 98, 100, and 101.

____. Learning to reuse: Adaptive model learning for evolving systems. In: AHRENDT, W.; TARIFA, S. L. T. (Ed.). **Integrated Formal Methods**. Cham: Springer International Publishing, 2019. p. 138–156. ISBN 978-3-030-34968-4. Citations on pages 28, 56, 71, 100, and 101.

____. Learning by sampling: Evaluating t-wise sampling criteria for learning family models. **Empirical Software Engineering**, 2020. (Submitted - **Status:** Rebuttal Phase with Major Reviews). Available: <https://link.springer.com/journal/10664>. Citations on pages 30, 75, 84, 85, 98, 100, and 101.

DAMASCENO, C. D. N.; SOUZA, P. S. L.; SIMAO, A. Um algoritmo paralelo para priorização de testes baseada em similaridade usando openmpi. In: **Anais Estendidos da Escola Regional de Alto Desempenho de São Paulo (ERAD-SP)**. Porto Alegre, RS, Brasil: SBC, 2017. p. 37–40. Citation on page 101.

DE RUITER, J. A tale of the openssl state machine: A large-scale black-box analysis. In: BRUMLEY, B. B.; RÖNING, J. (Ed.). **Secure IT Systems**. Cham: Springer International Publishing, 2016. p. 169–184. ISBN 978-3-319-47560-8. Citations on pages 28, 35, 56, 62, and 64.

DE RUITER, J.; POLL, E. Protocol state fuzzing of tls implementations. In: **Proceedings of the 24th USENIX Conference on Security Symposium**. Berkeley, CA, USA: USENIX Association, 2015. (SEC'15), p. 193–206. ISBN 978-1-931971-232. Available: <http://dl.acm.org/citation.cfm?id=2831143.2831156>. Citations on pages 25, 35, and 102.

DECAN, A.; MENS, T. What do package dependencies tell us about semantic versioning? **IEEE Transactions on Software Engineering**, p. 1–1, 2019. ISSN 2326-3881. Citation on page 105.

DEUTSCH, M. S. Tutorial series 7 software project verification and validation. **Computer**, v. 14, n. 4, p. 54–70, April 1981. ISSN 0018-9162. Citations on pages 23 and 33.

DEVROEY, X.; PERROUIN, G.; LEGAY, A.; SCHOBBENS, P.-Y.; HEYMANS, P. Covering spl behaviour with sampled configurations: An initial assessment. In: **Proceedings of the Ninth International Workshop on Variability Modelling of Software-intensive Systems**. New York, NY, USA: ACM, 2015. (VaMoS '15), p. 59:59–59:66. ISBN 978-1-4503-3273-6. Available: <http://doi.acm.org/10.1145/2701319.2701325>. Citation on page 85.

DEZA, M. M.; DEZA, E. Distances and similarities in data analysis. In: **Encyclopedia of Distances**. [S.l.]: Springer Berlin Heidelberg, 2013. p. 291–305. ISBN 978-3-642-30957-1. Citation on page 51.

DIAS-NETO, A. C.; TRAVASSOS, G. H. A picture from the model-based testing area: Concepts, techniques, and challenges. In: ZELKOWITZ, M. V. (Ed.). **Advances in Computers**. Elsevier, 2010, (Advances in Computers, v. 80). p. 45 – 120. Available: <http://www.sciencedirect.com/science/article/pii/S0065245810800026>. Citation on page 24.

DUHAIBY, O. al; MOOIJ, A.; VAN WEZEP, H.; GROOTE, J. F. Pitfalls in applying model learning to industrial legacy software. In: MARGARIA, T.; STEFFEN, B. (Ed.). **Leveraging Applications of Formal Methods, Verification and Validation. Industrial Practice**. Cham: Springer International Publishing, 2018. p. 121–138. ISBN 978-3-030-03427-6. Citations on pages 25, 34, 35, 55, and 70.

ECLIPSE. **Eclipse desktop and Web IDEs**. 2019. <https://www.eclipse.org/ide/>. [Online; accessed 19-May-19]. Citations on pages 63 and 84.

ELBAUM, S.; ROTHERMEL, G.; KANDURI, S.; MALISHEVSKY, A. G. Selecting a cost-effective test case prioritization technique. **Software Quality Journal**, v. 12, n. 3, p. 185–210, 2004. ISSN 1573-1367. Available: <http://dx.doi.org/10.1023/B:SQJO.0000034708.84524.22>. Citation on page 35.

ENSAN, F.; BAGHERI, E.; GAŠEVIĆ, D. Evolutionary search-based test generation for software product line feature models. In: RALYTË, J.; FRANCH, X.; BRINKKEMPER, S.; WRYCZA, S. (Ed.). **Advanced Information Systems Engineering**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. p. 613–628. ISBN 978-3-642-31095-9. Citations on pages 51, 102, and 108.

ERNST, M. D.; PERKINS, J. H.; GUO, P. J.; MCCAMANT, S.; PACHECO, C.; TSCHANTZ, M. S.; XIAO, C. The daikon system for dynamic detection of likely invariants. **Science of Computer Programming**, v. 69, n. 1, p. 35 – 45, 2007. ISSN 0167-6423. Special issue on Experimental Software and Toolkits. Available: <http://www.sciencedirect.com/science/article/pii/S016764230700161X>. Citation on page 35.

FENG, L.; LUNDMARK, S.; MEINKE, K.; NIU, F.; SINDHU, M. A.; WONG, P. Y. H. Case studies in learning-based testing. In: ____. **Testing Software and Systems: 25th IFIP WG 6.1 International Conference, ICTSS 2013, Istanbul, Turkey, November 13-15, 2013, Proceedings**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 164–179. ISBN 978-3-642-41707-8. Available: <http://dx.doi.org/10.1007/978-3-642-41707-8_11>. Citation on page 25.

FENSKE, W.; THüM, T.; SAAKE, G. A taxonomy of software product line reengineering. In: **Proceedings of the Eighth International Workshop on Variability Modelling of Software-Intensive Systems**. New York, NY, USA: ACM, 2013. (VaMoS '14), p. 1–8. ISBN 978-1-4503-2556-1. Citations on pages 29, 75, 98, and 105.

FITERĂU-BROŞTEAN, P.; HOWAR, F. Learning-based testing the sliding window behavior of tcp implementations. In: ____. **Critical Systems: Formal Methods and Automated Verification: Joint 22nd International Workshop on Formal Methods for Industrial Critical Systems and 17th International Workshop on Automated Verification of Critical Systems, FMICS-AVoCS 2017, Turin, Italy, September 18–20, 2017, Proceedings**. Cham: Springer International Publishing, 2017. p. 185–200. ISBN 978-3-319-67113-0. Available: <https://doi.org/10.1007/978-3-319-67113-0_12>. Citations on pages 25 and 102.

FRAGAL, V. H. **Automatic generation of configurable test-suites for software product lines**. Phd Thesis (PhD Thesis) — Universidade de São Paulo, 2017. [Online] <http://www.teses.usp.br/teses/disponiveis/55/55134/tde-10012019-085746/>. Citations on pages 52, 53, 94, and 108.

FRAGAL, V. H.; SIMAO, A.; MOUSAVI, M. R. 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**. Cham: Springer International Publishing, 2017. p. 210–227. ISBN 978-3-319-57666-4. Available: <http://dx.doi.org/10.1007/978-3-319-57666-4_13>. Citations on pages 26, 29, 30, 52, 53, 73, 74, 75, 86, and 103.

FRAGAL, V. H.; SIMAO, A.; MOUSAVI, M. R. Hierarchical featured state machines. **Science of Computer Programming**, v. 171, p. 67–88, 2019. ISSN 0167-6423. Citations on pages 53 and 106.

FRAGAL, V. H.; SIMAO, A.; MOUSAVI, M. R.; TURKER, U. C. Extending HSI Test Generation Method for Software Product Lines. **The Computer Journal**, v. 62, n. 1, p. 109–129, 05 2018. ISSN 0010-4620. Citations on pages 53, 74, and 75.

FUJIWARA, S.; BOCHMANN, G. v.; KHENDEK, F.; AMALOU, M.; GHEDAMSI, A. Test selection based on finite state models. **IEEE Transactions on Software Engineering**, v. 17, n. 6, p. 591–603, Jun 1991. ISSN 0098-5589. Citations on pages 37, 40, and 63.

GILL, A. **Introduction to the Theory of Finite State Machines**. New York: McGraw-Hill, 1962. Citations on pages 24, 35, 52, 74, 105, and 106.

GODFREY, M. W.; GERMAN, D. M. On the evolution of lehman's laws. **J. Softw. Evol. Process**, John Wiley & Sons, Inc., USA, v. 26, n. 7, p. 613–619, Jul. 2014. ISSN 2047-7473. Available: <https://doi.org/10.1002/smr.1636>. Citation on page 23.

GOODENOUGH, J. B.; GERHART, S. L. Toward a theory of test data selection. **IEEE Transactions on Software Engineering**, SE-1, n. 2, p. 156–173, June 1975. ISSN 0098-5589. Citation on page 34.

GROCE, A.; PELED, D.; YANNAKAKIS, M. Adaptive model checking. In: **Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems**. London, UK, UK: Springer-Verlag, 2002. (TACAS '02), p. 357–370. ISBN 3-540-43419-4. Available: <http://dl.acm.org/citation.cfm?id=646486.694482>. Citations on pages 25, 27, 39, 41, 42, 55, and 70.

GROZ, R.; SIMAO, A.; PETRENKO, A.; ORIAT, C. 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**. Cham: Springer International Publishing, 2015. p. 161–177. ISBN 978-3-319-25945-1. Available: <http://dx.doi.org/10.1007/978-3-319-25945-1_10>. Citation on page 25.

GRULER, A.; LEUCKER, M.; SCHEIDEMANN, K. Modeling and model checking software product lines. In: ____. **Formal Methods for Open Object-Based Distributed Systems: 10th IFIP WG 6.1 International Conference, FMOODS 2008, Oslo, Norway, June 4-6, 2008 Proceedings**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. p. 113–131. ISBN 978-3-540-68863-1. Available: <https://doi.org/10.1007/978-3-540-68863-1_8>. Citation on page 26.

GURBUZ, H. G.; TEKINERDOGAN, B. Model-based testing for software safety: a systematic mapping study. **Software Quality Journal**, v. 26, n. 4, p. 1327–1372, 2018. ISSN 1573-1367. Available: <https://doi.org/10.1007/s11219-017-9386-2>. Citations on pages 24 and 55.

HAGERER, A.; HUNGAR, H.; NIESE, O.; STEFFEN, B. Model generation by moderated regular extrapolation. In: ____. **Fundamental Approaches to Software Engineering: 5th International Conference, FASE 2002 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002 Grenoble, France, April 8–12, 2002 Proceedings**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002. p. 80–95. ISBN 978-3-540-45923-1. Available: <http://dx.doi.org/10.1007/3-540-45923-5_6>. Citation on page 25.

HAREL, D. Statecharts: A visual formalism for complex systems. **Sci. Comput. Program.**, Elsevier North-Holland, Inc., Amsterdam, The Netherlands, The Netherlands, v. 8, n. 3, p. 231–274, Jun. 1987. ISSN 0167-6423. Citations on pages 104 and 105.

HASLINGER, E. N.; LOPEZ-HERREJON, R. E.; EGYED, A. Reverse engineering feature models from programs' feature sets. In: **2011 18th Working Conference on Reverse Engineering**. [S.l.]: IEEE, 2011. p. 308–312. ISSN 2375-5369. Citations on pages 97 and 104.

HEMMATI, H.; ARCURI, A.; BRIAND, L. Achieving scalable model-based testing through test case diversity. **ACM Trans. Softw. Eng. Methodol.**, Association for Computing Machinery, New York, NY, USA, v. 22, n. 1, Mar. 2013. ISSN 1049-331X. Available: <https://doi.org/10.1145/2430536.2430540>. Citation on page 24.

HENARD, C.; PAPADAKIS, M.; PERROUIN, G.; KLEIN, J.; HEYMANS, P.; LE TRAON, Y. Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test configurations for software product lines. **IEEE Transactions on Software Engineering**, IEEE, v. 40, n. 7, p. 650–670, 2014. Citations on pages 51 and 96.

HESS, M. R.; KROMREY, J. D. Robust confidence intervals for effect sizes: A comparative study of cohen's d and cliff's delta under non-normality and heterogeneous variances. In: **Annual meeting - American Educational Research Association**. [S.l.: s.n.], 2004. Citations on pages 62 and 82.

HIGUERA, C. de la. **Grammatical Inference: Learning Automata and Grammars**. New York, NY, USA: Cambridge University Press, 2010. ISBN 0521763169, 9780521763165. Citation on page 47.

HOWAR, F.; STEFFEN, B.; MERTEN, M. From zulu to rers. In: ____. **Leveraging Applications of Formal Methods, Verification, and Validation: 4th International Symposium on Leveraging Applications, ISoLA 2010, Heraklion, Crete, Greece, October 18-21, 2010, Proceedings, Part I**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 687–704. ISBN 978-3-642-16558-0. Available: <http://dx.doi.org/10.1007/978-3-642-16558-0_55>. Citation on page 40.

HUISTRA, D.; MEIJER, J.; VAN DE POL, J. Adaptive learning for learn-based regression testing. In: HOWAR, F.; BARNAT, J. (Ed.). **Formal Methods for Industrial Critical Systems**. Switzerland: Springer, 2018. (Lecture Notes in Computer Science), p. 162–177. ISBN 978-3-030-00243-5. Citations on pages 25, 26, 27, 28, 34, 35, 41, 42, 43, 55, 57, 58, 60, 61, 67, 70, and 105.

HUNGAR, H.; NIESE, O.; STEFFEN, B. Domain-specific optimization in automata learning. In: ____. **Computer Aided Verification: 15th International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003. Proceedings**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. p. 315–327. ISBN 978-3-540-45069-6. Available: <https://doi.org/10.1007/978-3-540-45069-6_31>. Citations on pages 25 and 102.

IEEE. Iso/iec/ieee international standard for software engineering - software life cycle processes - maintenance. **ISO/IEC 14764:2006 (E) IEEE Std 14764-2006 Revision of IEEE Std 1219-1998)**, p. 1–58, Sep. 2006. ISSN null. Citation on page 23.

____. Systems and software engineering – vocabulary. **ISO/IEC/IEEE 24765:2010(E)**, p. 1–418, Dec 2010. Citations on pages 23, 33, and 34.

____. Ieee standard for system and software verification and validation. **IEEE Std 1012-2012 (Revision of IEEE Std 1012-2004)**, p. 1–223, May 2012. Citations on pages 24 and 33.

IRFAN, M. N.; ORIAT, C.; GROZ, R. Angluin style finite state machine inference with non-optimal counterexamples. In: **Proceedings of the First International wWorkshop on Model Inference In Testing**. New York, NY, USA: ACM, 2010. (MIIT '10), p. 11–19. ISBN 978-1-4503-0147-3. Available: <http://doi.acm.org/10.1145/1868044.1868046>. Citation on page 63.

____. Chapter 3 - model inference and testing. In: MEMON, A. (Ed.). Elsevier, 2013, (Advances in Computers, v. 89). p. 89 – 139. Available: <http://www.sciencedirect.com/science/article/pii/B9780124080942000035>. Citations on pages 24, 25, 27, 38, 41, 55, 97, and 102.

ISBERNER, M.; HOWAR, F.; STEFFEN, B. Learning register automata: from languages to program structures. **Machine Learning**, v. 96, n. 1, p. 65–98, Jul 2014. ISSN 1573-0565. Available: <https://doi.org/10.1007/s10994-013-5419-7>. Citations on pages 33 and 38.

____. The TTT algorithm: A redundancy-free approach to active automata learning. In: BONAKDARPOUR, B.; SMOLKA, S. A. (Ed.). **Runtime Verification**. [S.l.]: Springer International Publishing, 2014. p. 307–322. ISBN 978-3-319-11164-3. Citation on page 107.

____. The open-source learnlib. In: ____. **Computer Aided Verification: 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I**. Cham: Springer International Publishing, 2015. p. 487–495. ISBN 978-3-319-21690-4. Available: <https://doi.org/10.1007/978-3-319-21690-4_32>. Citations on pages 35, 40, 56, 61, and 69.

JOHANSEN, M. F.; HAUGEN, Ø.; FLEUREY, F. Properties of realistic feature models make combinatorial testing of product lines feasible. In: WHITTLE, J.; CLARK, T.; KÜHNE, T. (Ed.). **Model Driven Engineering Languages and Systems**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. p. 638–652. ISBN 978-3-642-24485-8. Citations on pages 79, 83, and 102.

JORGENSEN, P. **Software Testing: A Craftsman's Approach, Fourth Edition**. Taylor & Francis, 2013. (An Auerbach book). ISBN 9781466560680. Available: <https://books.google.com.br/books?id=6WlmAQAAQBAJ>. Citation on page 34.

KAMPENES, V. B.; DYBå, T.; HANNAY, J. E.; SJøBERG, D. I. A systematic review of effect size in software engineering experiments. **Information and Software Technology**, v. 49, n. 11, p. 1073 – 1086, 2007. ISSN 0950-5849. Citations on pages 62 and 82.

KANG, K.; COHEN, S.; HESS, J.; NOVAK, W.; PETERSON, A. **Feature-Oriented Domain Analysis (FODA) Feasibility Study**. Pittsburgh, PA, 1990. Available: <http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=11231>. Citations on pages 26, 48, 51, and 102.

KUHN, D. R.; KACKER, R. N.; LEI, Y. **Introduction to Combinatorial Testing**. 1st. ed. [S.l.]: Chapman & Hall/CRC, 2013. ISBN 1466552298, 9781466552296. Citation on page 51.

KUHN, D. R.; WALLACE, D. R.; GALLO, A. M. Software fault interactions and implications for software testing. **IEEE Transactions on Software Engineering**, v. 30, n. 6, p. 418–421, June 2004. Citation on page 51.

KUNZE, S.; MOSTOWSKI, W.; MOUSAVI, M. R.; VARSHOSAZ, M. Generation of failure models through automata learning. In: **2016 Workshop on Automotive Systems/Software Architectures (WASA)**. [S.l.: s.n.], 2016. p. 22–25. Citations on pages 25 and 102.

LAGUNA, M. A.; CRESPO, Y. A systematic mapping study on software product line evolution: From legacy system reengineering to product line refactoring. **Science of Computer Programming**, v. 78, n. 8, p. 1010–1034, 2013. ISSN 0167-6423. Citation on page 105.

LearnLib. **LearnLib: a framework for automata learning**. 2017. <https://learnlib.de/>. [Online; accessed 17-Out-2017]. Citations on pages 29 and 70.

____. **LearnLib 0.13 - Javadoc**. 2018. <http://learnlib.github.io/learnlib/maven-site/0.13.0/apidocs/>. [Online; accessed 06-Aug-2018]. Citation on page 63.

LEHMAN, M. On understanding laws, evolution, and conservation in the large-program life cycle. **Journal of Systems and Software**, v. 1, p. 213 – 221, 1979. ISSN 0164-1212. Available: <http://www.sciencedirect.com/science/article/pii/0164121279900220>. Citation on page 23.

LI, K.; GROZ, R.; SHAHBAZ, M. Integration testing of components guided by incremental state machine learning. In: **Testing: Academic Industrial Conference - Practice And Research Techniques (TAIC PART'06)**. [S.l.: s.n.], 2006. p. 59–70. Citation on page 39.

LOBATO, F. M. F.; DAMASCENO, C. D. N.; LEITE, D. S.; SANTOS Ândrea Kelly Ribeiro-dos; DARNET, S.; FRANCêS, C. R.; VIJAYKUMAR, N. L.; SANTANA Ádamo Lima de. Data analysis of multiplex sequencing at SOLiD platform: A probabilistic approach to characterization and reliability increase. **American Journal of Molecular Biology (AJMB)**, Scientific Research Publishing, v. 08, n. 01, p. 26–38, 2018. [Online]. Available: <https://doi.org/10.4236/ajmb.2018.81003>. Citation on page 101.

LOPEZ-HERREJON, R. E.; FERRER, J.; CHICANO, F.; EGYED, A.; ALBA, E. Comparative analysis of classical multi-objective evolutionary algorithms and seeding strategies for pairwise testing of software product lines. In: **2014 IEEE Congress on Evolutionary Computation (CEC)**. [S.l.: s.n.], 2014. p. 387–396. ISSN 1941-0026. Citations on pages 51 and 102.

MACHADO, P.; VINCENZI, A.; MALDONADO, J. C. Software testing: An overview. In: ____. **Testing Techniques in Software Engineering: Second Pernambuco Summer School on Software Engineering, PSSE 2007, Recife, Brazil, December 3-7, 2007, Revised Lectures**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 1–17. ISBN 978-3-642-14335-9. Available: <https://doi.org/10.1007/978-3-642-14335-9_1>. Citation on page 35.

MARGARIA, T.; RAFFELT, H.; STEFFEN, B. Knowledge-based relevance filtering for efficient system-level test-based model generation. **Innovations in Systems and Software Engineering**, v. 1, n. 2, p. 147–156, Sep 2005. ISSN 1614-5054. Available: <https://doi.org/10.1007/s11334-005-0016-y>. Citation on page 41.

MARIANI, L.; PEZZÈ, M.; ZUDDAS, D. Chapter four - recent advances in automatic black-box testing. In: MEMON, A. (Ed.). Elsevier, 2015, (Advances in Computers, v. 99). p. 157 – 193. Available: <http://www.sciencedirect.com/science/article/pii/S0065245815000315>. Citations on pages 24, 25, 27, 33, 38, 43, and 55.

MARINESCU, R.; SECELEANU, C.; GUEN, H. L.; PETTERSSON, P. Chapter three - a research overview of tool-supported model-based testing of requirements-based designs. In: HURSON, A. R. (Ed.). Elsevier, 2015, (Advances in Computers, v. 98). p. 89 – 140. Available: <http://www.sciencedirect.com/science/article/pii/S0065245815000297>. Citations on pages 23, 24, 33, and 38.

MARQUES, M.; SIMMONDS, J.; ROSSEL, P. O.; BASTARRICA, M. C. Software product line evolution: A systematic literature review. **Information and Software Technology**, v. 105, p. 190–208, 2019. ISSN 0950-5849. Citations on pages 29, 75, 98, and 105.

MASSOL, V.; HUSTED, T. **JUnit in Action**. Greenwich, CT, USA: Manning Publications Co., 2003. ISBN 1930110995. Citation on page 35.

MEINKE, K.; SINDHU, M. A. Incremental learning-based testing for reactive systems. In: ____. **Tests and Proofs: 5th International Conference, TAP 2011, Zurich, Switzerland, June 30 – July 1, 2011. Proceedings**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. p. 134–151. ISBN 978-3-642-21768-5. Available: <http://dx.doi.org/10.1007/978-3-642-21768-5_11>. Citations on pages 24 and 38.

____. Incremental learning-based testing for reactive systems. In: GOGOLLA, M.; WOLFF, B. (Ed.). **Tests and Proofs**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. p. 134–151. ISBN 978-3-642-21768-5. Citation on page 40.

MEINKE, K.; WALKINSHAW, N. Model-based testing and model inference. In: ____. **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**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. p. 440–443. ISBN 978-3-642-34026-0. Available: <http://dx.doi.org/10.1007/978-3-642-34026-0_32>. Citations on pages 25 and 34.

MLYNARSKI, M.; GÜLDALI, B.; WEIßLEDER, S.; ENGELS, G. Model-based testing: Achievements and future challenges. In: HURSON, A.; MEMON, A. (Ed.). Elsevier, 2012, (Advances in Computers, v. 86). p. 1 – 39. Available: <http://www.sciencedirect.com/science/article/pii/B9780123965356000016>. Citations on pages 24 and 38.

MYERS, G. J.; SANDLER, C.; BADGETT, T. **The Art of Software Testing**. 3rd. ed. John Wiley & Sons, Inc., 2012. i–xi p. ISBN 1118031962, 9781118031964. Available: <http://dx.doi.org/10.1002/9781119202486>. Citation on page 33.

NAUR, P.; RANDELL, B. (Ed.). **Proceedings, NATO Conference on Software Engineering**. Garmisch, Germany: [s.n.], 1968. Citation on page 23.

NEJATI, S.; SABETZADEH, M.; CHECHIK, M.; EASTERBROOK, S.; ZAVE, P. Matching and merging of variant feature specifications. **IEEE Transactions on Software Engineering**, v. 38, n. 6, p. 1355–1375, Nov 2012. ISSN 0098-5589. Citations on pages 103 and 104.

NEUBAUER, J.; STEFFEN, B.; BAUER, O.; WINDMüLLER, S.; MERTEN, M.; MARGARIA, T.; HOWAR, F. Automated continuous quality assurance. In: **2012 First International Workshop on Formal Methods in Software Engineering: Rigorous and Agile Approaches (FormSERA)**. [s.n.], 2012. p. 37–43. Available: <http://doi.org/10.1109/FormSERA.2012.6229787>. Citations on pages 25 and 105.

NIESE, O. **An integrated approach to testing complex systems**. Phd Thesis (PhD Thesis) — University of Dortmund, 2003. Available: <http://d-nb.info/969717474>. Citation on page 39.

OpenSSL Foundation, Inc. **OpenSSL - Cryptography and SSL/TLS Toolkit**. 2018. <https://www.openssl.org/>. [Online; accessed 21-Ago-2018]. Citations on pages 56 and 62.

____. **OpenSSL Releases at Github**. 2018. <https://github.com/openssl/openssl/releases>. [Online; accessed 26-Ago-2018]. Citation on page 64.

ORACLE. **Java SE Development Kit 8 Downloads**. 2014. <https://www.oracle.com/java/technologies/javase-jdk8-downloads.html>. [Online; accessed 14-Mar-20]. Citations on pages 63 and 84.

OSTER, S. **Feature Model-based Software Product Line Testing**. Phd Thesis (PhD Thesis) — Technische Universität, 2012. Available: <https://d-nb.info/1106113845/34>. Citations on pages 26 and 28.

OSTER, S.; WÜBBEKE, A.; ENGELS, G.; SCHÜRR, A. A survey of model-based software product lines testing. In: **Model-Based Testing for Embedded Systems**. [S.l.]: CRC Press, 2011. p. 338–381. Citation on page 26.

PELED, D.; VARDI, M. Y.; YANNAKAKIS, M. Black box checking. In: ____. **Formal Methods for Protocol Engineering and Distributed Systems (FORTE XII)**. Boston, MA: Springer US, 1999. p. 225–240. ISBN 978-0-387-35578-8. Available: <http://dx.doi.org/10.1007/978-0-387-35578-8_13>. Citations on pages 25 and 102.

PELLEGRINO, G.; LIN, Q.; HAMMERSCHMIDT, C.; VERWER, S. Learning behavioral fingerprints from netflows using timed automata. In: **2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)**. [S.l.: s.n.], 2017. p. 308–316. Citation on page 108.

PERROUIN, G.; SEN, S.; KLEIN, J.; BAUDRY, B.; LE TRAON, Y. Automated and scalable t-wise test case generation strategies for software product lines. In: **2010 Third International Conference on Software Testing, Verification and Validation**. [S.l.: s.n.], 2010. p. 459–468. Citations on pages 28, 30, 50, 51, 79, 102, and 108.

PETKE, J.; YOO, S.; COHEN, M. B.; HARMAN, M. Efficiency and early fault detection with lower and higher strength combinatorial interaction testing. In: **Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering**. New York, NY, USA: ACM, 2013. (ESEC/FSE 2013), p. 26–36. ISBN 978-1-4503-2237-9. Available: <http://doi.acm.org/10.1145/2491411.2491436>. Citations on pages 80 and 95.

POHL, K.; BÖCKLE, G.; VAN DER LINDEN, F. J. **Software Product Line Engineering: Foundations, Principles and Techniques**. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005. ISBN 3540243720. Citations on pages 26, 28, 29, 48, and 104.

PRESTON-WERNER, T. **Semantic Versioning 2.0.0**. 2013. <https://semver.org/spec/v2.0.0.html>. [Online; accessed 21-Feb-2020]. Citation on page 105.

PRETSCHNER, A.; PHILIPPS, J. 10 methodological issues in model-based testing. In: ____. **Model-Based Testing of Reactive Systems: Advanced Lectures**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. p. 281–291. ISBN 978-3-540-32037-1. Available: <https://doi.org/10.1007/11498490_13>. Citations on pages 24 and 38.

RAFFELT, H.; MERTEN, M.; STEFFEN, B.; MARGARIA, T. Dynamic testing via automata learning. **International Journal on Software Tools for Technology Transfer**, v. 11, n. 4, p. 307, 2009. ISSN 1433-2787. Available: <http://dx.doi.org/10.1007/s10009-009-0120-7>. Citations on pages 25 and 102.

RAFFELT, H.; STEFFEN, B. 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**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. p. 377–380. ISBN 978-3-540-33094-3. Available: <https://doi.org/10.1007/11693017_28>. Citations on pages 28, 63, 85, and 96.

RIVEST, R.; SCHAPIRE, R. Inference of finite automata using homing sequences. **Information and Computation**, v. 103, n. 2, p. 299 – 347, 1993. ISSN 0890-5401. Available: <http://www.sciencedirect.com/science/article/pii/S0890540183710217>. Citation on page 41.

RSTUDIO. **RStudio: Open source and enterprise-ready professional software for data science**. 2019. <https://www.rstudio.com/>. [Online; accessed 19-May-19]. Citation on page 85.

RUNESON, P.; ENGSTRöM, E. Chapter 7 - regression testing in software product line engineering. In: HURSON, A.; MEMON, A. (Ed.). [S.l.]: Elsevier, 2012, (Advances in Computers, v. 86). p. 223 – 263. Citations on pages 98 and 105.

RYSSEL, U.; PLOENNIGS, J.; KABITZSCH, K. Extraction of feature models from formal contexts. In: **Proceedings of the 15th International Software Product Line Conference, Volume 2**. New York, NY, USA: ACM, 2011. (SPLC '11), p. 1–8. ISBN 978-1-4503-0789-5. Citations on pages 97 and 104.

SABOURI, H.; KHOSRAVI, R. Delta modeling and model checking of product families. In: ____. **Fundamentals of Software Engineering: 5th International Conference, FSEN 2013, Tehran, Iran, April 24-26, 2013, Revised Selected Papers**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 51–65. ISBN 978-3-642-40213-5. Available: <http://dx.doi.org/10.1007/978-3-642-40213-5_4>. Citations on pages 73, 75, 103, and 105.

SAID, W.; QUANTE, J.; KOSCHKE, R. Do extracted state machine models help to understand embedded software? In: **Proceedings of the 27th International Conference on Program Comprehension**. IEEE Press, 2019. (ICPC '19), p. 191–196. Available: <https://doi.org/10.1109/ICPC.2019.00038>. Citation on page 24.

SAMIH, H.; GUEN, H. L.; BOGUSCH, R.; ACHER, M.; BAUDRY, B. Deriving usage model variants for model-based testing: An industrial case study. In: **2014 19th International Conference on Engineering of Complex Computer Systems**. [S.l.: s.n.], 2014. p. 77–80. Citations on pages 30, 74, 75, 86, 87, and 96.

SCHAEFER, I.; BETTINI, L.; BONO, V.; DAMIANI, F.; TANZARELLA, N. Delta-oriented programming of software product lines. In: ____. **Software Product Lines: Going Beyond: 14th International Conference, SPLC 2010, Jeju Island, South Korea, September 13-17, 2010. Proceedings**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 77–91. ISBN 978-3-642-15579-6. Available: <http://dx.doi.org/10.1007/978-3-642-15579-6_6>. Citation on page 103.

SCHAEFER, I.; RABISER, R.; CLARKE, D.; BETTINI, L.; BENAVIDES, D.; BOTTERWECK, G.; PATHAK, A.; TRUJILLO, S.; VILLELA, K. Software diversity: state of the art and perspectives. **International Journal on Software Tools for Technology Transfer**, v. 14, n. 5, p. 477–495, 2012. ISSN 1433-2787. Available: <http://dx.doi.org/10.1007/s10009-012-0253-y>. Citations on pages 26, 28, 34, 73, 75, and 79.

SCHUTS, M.; HOOMAN, J.; VAANDRAGER, F. Refactoring of legacy software using model learning and equivalence checking: An industrial experience report. In: ____. **Integrated Formal Methods: 12th International Conference, IFM 2016, Reykjavik, Iceland, June 1-5, 2016, Proceedings**. Cham: Springer International Publishing, 2016. p. 311–325. ISBN 978-3-319-33693-0. Available: <http://dx.doi.org/10.1007/978-3-319-33693-0_20>. Citation on page 25.

SERY, O.; FEDYUKOVICH, G.; SHARYGINA, N. Incremental upgrade checking. In: ____. **Validation of Evolving Software**. [S.l.]: Springer International Publishing, 2015. p. 55–72. ISBN 978-3-319-10623-6. Citations on pages 43 and 105.

SHAFIQUE, M.; LABICHE, Y. A systematic review of state-based test tools. **International Journal on Software Tools for Technology Transfer**, v. 17, n. 1, p. 59–76, 2015. ISSN 1433-2787. Available: <https://doi.org/10.1007/s10009-013-0291-0>. Citations on pages 24 and 38.

SHAHBAZ, M.; GROZ, R. Inferring mealy machines. In: ____. **FM 2009: Formal Methods: Second World Congress, Eindhoven, The Netherlands, November 2-6, 2009. Proceedings**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. p. 207–222. ISBN 978-3-642-05089-3. Available: <http://dx.doi.org/10.1007/978-3-642-05089-3_14>. Citations on pages 24, 38, 39, 40, 41, 47, 60, 62, 76, and 107.

____. Analysis and testing of black-box component-based systems by inferring partial models. **Softw. Test. Verif. Reliab.**, John Wiley and Sons Ltd., Chichester, UK, v. 24, n. 4, p. 253–288,

Jun. 2014. ISSN 0960-0833. Available: <http://dx.doi.org/10.1002/stvr.1491>. Citation on page 25.

SHAHBAZ, M.; PARREAUX, B.; KLAY, F. Model inference approach for detecting feature interactions in integrated systems. In: **Feature Interactions in Software and Communication Systems IX, International Conferenceserence on Feature Interactions in Software and Communication Systems, ICFI 2007, 3-5 September 2007, Grenoble, France**. [S.l.: s.n.], 2007. p. 161–171. Citation on page 25.

SMEENK, W.; MOERMAN, J.; VAANDRAGER, F.; JANSEN, D. N. Applying automata learning to embedded control software. In: "BUTLER, M.; CONCHON, S.; ZAÏDI, F. (Ed.). **Formal Methods and Software Engineering**. Cham: Springer International Publishing, 2015. p. 67–83. ISBN 978-3-319-25423-4. Citations on pages 25 and 69.

SOKOLOVA, M.; LAPALME, G. A systematic analysis of performance measures for classification tasks. **Information Processing & Management**, v. 45, n. 4, p. 427 – 437, 2009. ISSN 0306-4573. Available: <http://www.sciencedirect.com/science/article/pii/S0306457309000259>. Citation on page 47.

SPINELLIS, D. Version control systems. **IEEE Software**, v. 22, n. 5, p. 108–109, Sep. 2005. ISSN 1937-4194. Citation on page 105.

STEFFENS, M.; OSTER, S.; LOCHAU, M.; FOGDAL, T. Industrial evaluation of pairwise spl testing with moso-polite. In: **Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems**. New York, NY, USA: ACM, 2012. (VaMoS '12), p. 55–62. ISBN 978-1-4503-1058-1. Available: <http://doi.acm.org/10.1145/2110147.2110154>. Citations on pages 80 and 95.

STEVENSON, A.; CORDY, J. R. A survey of grammatical inference in software engineering. **Sci. Comput. Program.**, Elsevier North-Holland, Inc., Amsterdam, The Netherlands, The Netherlands, v. 96, n. P4, p. 444–459, Dec. 2014. ISSN 0167-6423. Citations on pages 97 and 102.

TER BEEK, M. H.; DE VINK, E. P.; WILLEMSE, T. A. C. Family-based model checking with mcrl2. In: ____. **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**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2017. p. 387–405. ISBN 978-3-662-54494-5. Available: <http://dx.doi.org/10.1007/978-3-662-54494-5_23>. Citations on pages 73, 75, 103, and 105.

THÜM, T.; APEL, S.; KÄSTNER, C.; SCHAEFER, I.; SAAKE, G. A classification and survey of analysis strategies for software product lines. **ACM Comput. Surv.**, ACM, New York, NY, USA, v. 47, n. 1, p. 6:1–6:45, Jun. 2014. ISSN 0360-0300. Available: <http://doi.acm.org/10.1145/2580950>. Citations on pages 26, 49, 50, 52, 73, 79, 80, 81, 102, and 103.

THÜM, T.; KÄSTNER, C.; BENDUHN, F.; MEINICKE, J.; SAAKE, G.; LEICH, T. Featureide: An extensible framework for feature-oriented software development. **Science of Computer Programming**, v. 79, n. Supplement C, p. 70 – 85, 2014. ISSN 0167-6423. Experimental Software and Toolkits (EST 4): A special issue of the Workshop on Academic Software Development Tools and Techniques (WASDeTT-3 2010). Available: <http://www.sciencedirect.com/science/article/pii/S0167642312001128>. Citations on pages 29, 82, 83, 85, 96, and 102.

TORCHIANO, M. **effsize: Efficient Effect Size Computation (v. 0.7.1)**. [S.l.], 2017. <https://cran.r-project.org/web/packages/effsize/effsize.pdf> [Online; accessed 20-November-2017]. Citations on pages 62 and 82.

TRETMANS, J. Model-based testing and some steps towards test-based modelling. In: ____. **Formal Methods for Eternal Networked Software Systems: 11th International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM 2011, Bertinoro, Italy, June 13-18, 2011. Advanced Lectures**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. p. 297–326. ISBN 978-3-642-21455-4. Available: <https://doi.org/10.1007/978-3-642-21455-4_9>. Citation on page 38.

UTTING, M.; PRETSCHNER, A.; LEGEARD, B. A taxonomy of model-based testing approaches. **Software Testing, Verification and Reliability**, John Wiley & Sons, Ltd, v. 22, n. 5, p. 297–312, 2012. ISSN 1099-1689. Citations on pages 24, 26, 35, 50, and 73.

VAANDRAGER, F. Model learning. **Commun. ACM**, ACM, New York, NY, USA, v. 60, n. 2, p. 86–95, Jan. 2017. ISSN 0001-0782. Available: <http://doi.acm.org/10.1145/2967606>. Citations on pages 24, 25, 27, 38, 39, 55, 80, 103, and 105.

VALE, T.; ALMEIDA, E. S. de; ALVES, V.; KULESZA, U.; NIU, N.; LIMA, R. de. Software product lines traceability: A systematic mapping study. **Information and Software Technology**, v. 84, p. 1–18, 2017. ISSN 0950-5849. Citations on pages 29, 75, and 98.

VAN DER LINDEN, F. J.; SCHMID, K.; ROMMES, E. **Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering**. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007. ISBN 3540714367. Citations on pages 26 and 48.

VARGHA, A.; DELANEY, H. D. A critique and improvement of the CL common language effect size statistics of McGraw and Wong. **Journal of Educational and Behavioral Statistics**, v. 25, n. 2, p. 101–132, 2000. Citations on pages 62 and 82.

VARSHOSAZ, M.; AL-HAJJAJI, M.; THüM, T.; RUNGE, T.; MOUSAVI, M. R.; SCHAEFER, I. A classification of product sampling for software product lines. In: **Proceedings of the 22nd International Systems and Software Product Line Conference - Volume 1**. New York, NY, USA: Association for Computing Machinery, 2018. (SPLC '18), p. 1–13. ISBN 9781450364645. Available: <https://doi.org/10.1145/3233027.3233035>. Citations on pages 30, 50, 51, 74, 79, 80, 94, 102, and 106.

VASILEVSKII, M. P. Failure diagnosis of automata. **Cybernetics**, v. 9, n. 4, p. 653–665, 1973. ISSN 1573-8337. Available: <http://dx.doi.org/10.1007/BF01068590>. Citations on pages 24, 37, 38, 40, and 103.

VIBeS. **VIBeS - Variability Intensive system Behavioural teSting - Case study: Aero UC5**. 2016. <https://projects.info.unamur.be/vibes/case-study-aerouc5.html>. [Online; accessed 23-Mar-20]. Citation on page 88.

____. **VIBeS - Variability Intensive system Behavioural teSting - Case study: Card payement terminal**. 2016. <https://projects.info.unamur.be/vibes/case-study-cpterminal.html>. [Online; accessed 23-Mar-20]. Citation on page 88.

____. **VIBeS - Variability Intensive system Behavioural teSting - Case study: Minepump**. 2016. <https://projects.info.unamur.be/vibes/case-study-aerouc5.html>. [Online; accessed 23-Mar-20]. Citation on page 89.

VOLPATO, M.; TRETMANS, J. Approximate active learning of nondeterministic input output transition systems. **ECEASST**, v. 72, 2015. Available: <http://journal.ub.tu-berlin.de/eceasst/article/view/1008>. Citation on page 39.

VON RHEIN, A.; GREBHAHN, A.; APEL, S.; SIEGMUND, N.; BEYER, D.; BERGER, T. Presence-condition simplification in highly configurable systems. In: **Proceedings of the 37th International Conference on Software Engineering - Volume 1**. Piscataway, NJ, USA: IEEE, 2015. (ICSE '15), p. 178–188. ISBN 978-1-4799-1934-5. Citation on page 97.

WALKINSHAW, N. Chapter 1 - reverse-engineering software behavior. In: MEMON, A. (Ed.). **Advances in Computers**. Elsevier, 2013, (Advances in Computers, Supplement C). p. 1 – 58. Available: <http://www.sciencedirect.com/science/article/pii/B978012408089800001X>. Citations on pages 24, 26, 38, 55, 73, 75, and 106.

WALKINSHAW, N.; BOGDANOV, K. Automated comparison of state-based software models in terms of their language and structure. **ACM Transactions on Software Engineering and Methodology**, ACM, New York, NY, USA, v. 22, n. 2, p. 1–37, Mar. 2013. ISSN 1049-331X. Citations on pages 29, 44, 45, 46, 47, 48, 62, 74, 76, 82, 84, 85, 103, and 104.

WEYUKER, E. J. On testing non-testable programs. **The Computer Journal**, v. 25, n. 4, p. 465–470, 1982. Available: <+http://dx.doi.org/10.1093/comjnl/25.4.465>. Citation on page 35.

____. Assessing test data adequacy through program inference. **ACM Trans. Program. Lang. Syst.**, ACM, New York, NY, USA, v. 5, n. 4, p. 641–655, Oct. 1983. ISSN 0164-0925. Available: <http://doi.acm.org/10.1145/69575.357231>. Citations on pages 24, 33, and 38.

WIECZOREK, W. **Grammatical Inference: Algorithms, Routines and Applications**. Cham: Springer International Publishing, 2017. ISBN 978-3-319-46801-3. Available: <https://doi.org/10.1007/978-3-319-46801-3>. Citation on page 47.

WINDMÜLLER, S.; NEUBAUER, J.; STEFFEN, B.; HOWAR, F.; BAUER, O. Active continuous quality control. In: **Proceedings of the 16th International ACM Sigsoft Symposium on Component-Based Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2013. (CBSE '13), p. 111–120. ISBN 9781450321228. Available: <https://doi.org/10.1145/2465449.2465469>. Citations on pages 25, 27, 41, 42, 43, 55, and 70.

WOHLIN, C.; RUNESON, P.; HÖST, M.; OHLSSON, M. C.; REGNELL, B.; WESSLÉN, A. Systematic literature reviews. In: ____. **Experimentation in Software Engineering**. Berlin, Heidelberg: Springer, 2012. p. 45–54. ISBN 978-3-642-29044-2. Citation on page 62.

YANG, N.; ASLAM, K.; SCHIFFELERS, R.; LENSINK, L.; HENDRIKS, D.; CLEOPHAS, L.; SEREBRENIK, A. Improving model inference in industry by combining active and passive learning. In: **26th IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER 2019)**. United States: Institute of Electrical and Electronics Engineers (IEEE), 2019. p. 253–263. Citation on page 107.

YOO, S.; HARMAN, M. Regression testing minimization, selection and prioritization: A survey. **Softw. Test. Verif. Reliab.**, John Wiley and Sons Ltd., Chichester, UK, v. 22, n. 2, p. 67–120, Mar. 2012. ISSN 0960-0833. Available: <http://dx.doi.org/10.1002/stv.430>. Citations on pages 35 and 51.


ICMC
USP
SÃO CARLOS