LIFTS: Learning Featured Transition Systems

Sophie Fortz
sophie.fortz@unamur.be
University of Namur
Namur, Belgium

ABSTRACT

This PhD project aims to automatically learn transition systems capturing the behaviour of a whole family of software-based systems. Reasoning at the family level yields important economies of scale and quality improvements for a broad range of systems such as software product lines, adaptive and configurable systems. Yet, to fully benefit from the above advantages, a model of the system family's behaviour is necessary. Such a model is often prohibitively expensive to create manually due to the number of variants. For large long-lived systems with outdated specifications or for systems that continuously adapt, the modelling cost is even higher. Therefore, this PhD proposes to automate the learning of such models from existing artefacts. To advance research at a fundamental level, our learning target are Featured Transition Systems (FTS), an abstract formalism that can be used to provide a pivot semantics to a range of variability-aware state-based modelling languages. The main research questions addressed by this PhD project are: (1) Can we learn variability-aware models efficiently? (2) Can we learn FTS in a black-box fashion? (*i.e.,* with access to execution logs but not to source code); (3) Can we learn FTS in a white/grey-box testing fashion? (*i.e.,* with access to source code); and (4) How do the proposed techniques scale in practice?

CCS CONCEPTS

• **Software and its engineering → Software reverse engineering; Software product lines**.

KEYWORDS

Featured Transition Systems, Software Product Lines, Variability Mining, Active Automata Learning, Model Learning

ACM Reference Format:
Sophie Fortz. 2021. LIFTS: Learning Featured Transition Systems. In *25th ACM International Systems and Software Product Line Conference - Volume B (SPLC '21), September 6–11, 2021, Leicester, United Kingdom.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3461002.3473066

1 INTRODUCTION AND MOTIVATION

Variability-Intensive Systems (VIS) form a vast and heterogeneous class of systems that encompasses *software product lines* (SPL) [49], configurable systems, adaptive systems, *etc.* All these systems have the ability to be customised to specific needs, through the (de)activation of different options or *features*, a phenomenon known as *variability*. Addressing variability proactively during software engineering (SE) activities means shifting from reasoning on a single system to reasoning on a *family* of systems (*i.e.,* a set of *variants*) to yield important economies of scale and quality improvements [49]. Conversely, variability can also be a curse, especially for Quality Assurance (QA), *i.e.,* verification and testing of such systems, due to the combinatorial explosion of the number of system/software variants. Verifying or testing each variant is therefore impossible in the vast majority of practical cases.

About a decade ago, *Featured Transition Systems* (FTS) were introduced as a formalism to represent, and reason on, the behaviour of VIS [17]. Instead of representing each variant by a (classical) *transition systems* (TS), an FTS bears annotations that relate transitions to features through *feature expressions (FE)*. By their large expressiveness to encode variability [7, 66], FE allow FTS to reason *at the family level* by modelling all the variants of a system in a single behavioural model. FTS have been shown to significantly improve the possibilities and execution time of automated QA activities such as model-checking and model-based testing [14–16, 24]. They can also be useful to guide design exploration activities [43].

Yet, as most model-based approaches, FTS modelling requires both strong human expertise and significant effort that would be unaffordable in many cases, in particular for large legacy systems with outdated specifications and/or systems that evolve continuously. The overall objective of this research is **to automatically learn FTS to ease the burden of modelling them and support continuous QA activities**. LIFTS addresses current **automation** and **scalability** issues. For this purpose, we will leverage Machine Learning (ML) techniques to develop efficient and general behavioural inference of FTS.

2 RESEARCH QUESTIONS

LIFTS investigates four main research questions:

RQ1 ***How can we learn variability-aware models efficiently?*** Primarily, we explore theoretically how variability can affect learning. The challenge here is to find tractable alternatives to the naive approach (*i.e.,* merging each individual TS) whose worst-case complexity is $O((2^n) * cost_L + k^{2^n})$ where k is the number of TS states, n the number of VIS features and $cost_L$ is the complexity of a specific algorithm to learn a single variant. The general strategy to address RQ1 will be to take advantage of shared behaviour amongst variants *during the learning phase*.

RQ2 ***Can we learn FTS in a black-box fashion?*** In this scenario, we assume that we do not have access to the source code of the system but that we can interact with it at runtime and/or

SPLC '21, September 6–11, 2021, Leicester, United Kingdom

ACM ISBN 978-1-4503-8470-4/21/09…$15.00
https://doi.org/10.1145/3461002.3473066

have access to execution traces. This happens frequently *e.g.,* when source code is closed, unavailable or some parts of the system's functionality are realised by non-software components as it is the case in cyber-physical systems.

RQ3 ***Can we learn FTS in a white/grey-box testing fashion?*** In this second scenario, we assume the learner has access to source code, which will improve its precision since all possible behaviour can (theoretically) be analysed. We will also consider the grey-box scenario where we learn from both observed behaviour and source code.

RQ4 ***What is the scalability of the proposed techniques in practice?*** LIFTS' techniques will eventually have to deal with large industrial VIS, where they are the most needed. This question therefore addresses the scalability of the theoretical results obtained from RQ1-RQ3 from an empirical perspective.

Hypothesis. The LIFTS project concentrates on the behavioural aspects and assumes that the Feature Model (see Section 3) already exists or has been learned in some way. Indeed, several techniques to learn a FM exist (*e.g.,* they can be learned from variant catalogues (product tables) using data mining [1, 2] or evolutionary algorithms [46]). Other approaches include learning the FM via static analysis of variant configurators [56] or via natural language analysis [45].

3 METHODOLOGY AND APPROACH

3.1 State of the art

Feature Modelling. *Feature Models* [41, 42] are tree-like diagrams representing common and variable aspects of a variability-aware system, where features (nodes) are decomposed hierarchically using Boolean operators and cross-tree constraints (edges). Over time, more sophisticated FM dialects were proposed [27], equipped with formal semantics [13, 18, 47, 53], automated analyses [6] and comprehensive tool support [50]. A FM declares the features of a VIS at a very abstract level and constrains how they can be combined. As such, it does not aim to model the behaviour of a VIS, only its *structural variability.*

Featured Transition Systems. Complementary to FM, FTS model the behaviour of a VIS. An FTS uses FE that are logical formulae referring to its structural variability. FE describe which variants can execute the behaviour encoded by the transitions of the FTS.

Learning Behaviour. Reconstructing a behavioural model from an existing software system is an active line of research which can be divided into two categories: *black-box* and *white-box* approaches, both of which are addressed in this PhD project.

Black-box approaches were particularly influenced by the seminal L^* algorithm from Dana Angluin [4]. This approach is based on a learning component that *actively* learns a model by testing: it generates candidate input sequences of action to a teaching component, which checks whether they are part of the behaviour of the system. Based on that, the learning component incrementally learns a behavioural model of the system that can then be assessed for equivalence.

Angluin's algorithm is a powerful theoretical framework that has given birth to numerous optimised versions and extensions (*e.g.,* with probabilities [3]), some of which were integrated in learning libraries [51]. There are also *passive* approaches where the learning component uses existing observations (execution traces) [26, 37, 44, 48, 58, 59, 63, 67]. In this case, the model is incomplete and can only contain the *observed behaviour* of the system [62]. Process discovery algorithms studied by the process mining community [65] also fall into this category.

White-box approaches rely on program analysis (*e.g.,* Shoham *et al.* to mine Internet API specifications [57], or Fraser *et al.* to infer object usage and thereby generate more meaningful tests [31]).

Black-box and white-box approaches are complementary and can be orchestrated in a grey-box fashion. For example, Howar *et al.* use a mix of static, dynamic and *concolic analysis* (a mix of symbolic and concrete execution) to learn safe interfaces for critical embedded systems [39]. Recently, they suggested a grey-box scenario where predicates or guards are exploited to guide black-box learning [40].

In contrast to FM learning, learning behavioural models of VIS (*i.e.,* at the family level) is still in its infancy. Buijs *et al.* [9] use genetic algorithms to combine process models mined from event logs of (a few) different variants, while Greenyer *et al.* [32, 33] check and synthesise controllers for VIS from scenarios (message sequence charts). The contexts and assumptions of these contributions are quite remote from ours.

We found very few contributions matching our goal. First, based on their approach to keep VIS models up-to-date [20], Damasceno *et al.* recently proposed to learn featured finite state-machine models *from individual models* [19]. Other approaches such as [25, 52] aim to learn other kind of family behavioural model by merging models of single system. However, at this stage, these works are limited to few variants, due to the high cost of generating and merging individual models. Devroey *et al.* [23] learned usage models (Markov Chains) from logs in order to perform statistical prioritisation of FTS-based tests. While the FTS was partly based on the learned behaviour, *significant human effort and expertise* was necessary to complete the FTS with FE. Additionally, **none of the aforementioned research sought to automatically construct FTS** as we aim in this PhD project. FTS being a fundamental formalism that can serve as a semantics for other VIS modelling languages such as UML State Diagrams (*e.g.,* via flattening [22]), our results are intended to *be more generic and therefore have a more profound impact on behavioural inference and automation.*

3.2 Work Packages

The LIFTS project will contain four main work packages (WP) described below.

WP1 *Formalise the Variability-aware Learning Problem.* In WP1, we conduct a state-of-the-art exploration and a systematic comparison of applicable learning approaches from different communities including SE, ML and process mining. Then, we want to formulate the learning problem for VIS to define abstractions that will guide variability-aware learning algorithms. In contrast to current approaches which reuse existing single-system learning algorithms as-is (*e.g.,* by naively merging TS), we aim to make variability a first-class concept which is leveraged to gain efficiency. Our vision is to come up with a variability-aware foundational algorithm (analogue to

Angluin's L^* for single systems) that can later be extended and tailored in multiple ways for various purposes.

WP2 *Develop Black-box Model Learning Techniques.* In WP2, we will build a black-box FTS learner. We will consider passive black-box approaches, which only rely on existing logs, but also active ones, like Angluin-style learning [4, 8], which assume direct interactions with the system to learn.
Some passive approaches to consider are the process discovery techniques [65], designed to deal with large amounts of data, but providing less guarantees since they cannot deal with negative examples. In other words, we can ensure that a behaviour is allowed by the system, but we cannot ensure that a behaviour is not allowed. The process discovery algorithms will have to be adapted to deal with FE on the transitions. We will also transpose other methods to VIS, including those based on ML [60] *e.g.,* Long-Short Term Memory (LSTM) [38] and Gated Recurrent Unit (GRU) [11] that can deal with temporal sequences like execution traces.

WP3 *Develop White/Grey-box Model Learning Techniques.* Source code analysis allows to retrieve information that is difficult to obtain via black-box queries [40] (*e.g.,* learning properties to purvey to Angluin's oracle), and is of interest to learn *extended FTS* (*e.g.,* FTS extended with hierarchy, concurrency or quantitative properties). *Concolic execution* seems relevant to infer FE that are part of FTS. We will also consider a mix of scenarios, leading to a *grey-box learner.*

WP4 *Perform Empirical Validation.* To evaluate the applicability of our learning techniques, the prototypes will be applied to learn FTS from a range of existing codebases and datasets, both from open-source communities (*e.g.,* [35]) and industrial partners[1] such as IBA, Haulogy and SkalUP. This diversification of cases allows for a better generalisation of our empirical results. However, because white-box learning depends on the programming language, we will probably only consider Java, as it is the most used language in our datasets and for which several robust static analysis frameworks exist [36, 64].

Methodology and risk management. For better risk management (notably on scalability), we adopt an iterative methodology where theoretical investigations are systematically confronted with empirical evaluations. Risks related to case collection are mitigated by a two-pronged strategy. First, we employ generated examples and open-source cases to ensure that learning strategies are worthwhile. Second, we will rely on industrial code to conduct realistic assessments of our contributions.

4 PRELIMINARY RESULTS

So far, we focused on *WP1* and *WP2*, by investigating state of the art through a mapping study, experimenting with Recurrent Neural Networks (RNN) [30], and considering a first adaptation of Angluin's L^* algorithm.

[1] https://iba-worldwide.com, http://www.haulogy.net, https://skalup.com/

4.1 Variability-Aware Behavioural Modelling: A Cross-Domain Mapping Study (WP1)

This study focuses on models describing the behaviour of an entire family of systems, *i.e.,* taking variability into account. SPL are a well fitted example since they naturally imply variability (*e.g.,* FTS, Featured Finite State Machines [34], etc). Process lines also seem relevant for the same reason, but the usual techniques do not always imply straightforward variability. There are three common ways of representing process/product families: with a collection of models; with a reference model (*i.e.,* a model representing the most common behaviour and which should be adapted, depending of the needs); and finally a configurable process/product line. We focus on the last category, since it is the only one explicitly supporting variability by means of graph annotations for example. This cross-domain mapping study aims to build bridges between the different communities, in order to have a better overview of the existing techniques to model variability-aware behaviour. So far, about 5,000 papers were evaluated and 475 were accepted for a second round of selection. We will submit this work to a journal later this year.

4.2 VaryMinions: Leveraging RNNs to Identify Variants in Event Logs (WP2)

Motivation. Business processes capture the activities of any profit or non-profit, public or private organisation, coordinating humans and software to collectively deliver value. As organisations evolve, new needs appear, requiring a variability mechanism and leading to the emergence of *process variants.* We consider process executions stored in event logs, where an *event trace* (or trace) is an ordered sequence of events. To debug an anomalous process execution or to explore process refactoring opportunities, it is necessary to identify which variant(s) may have produced a given trace. Existing *variant analysis* [61] techniques do not answer this question but rather cover the inverse operation *i.e.,* focusing on the differences between identified variants. In this paper, we train RNNs [54] with different hyperparameters (loss and activation functions among others) to predict the candidate variant(s) that could produce a given event trace. Figure 1 describes the workflow of the approach. Our results have been accepted to the MaLTeSQuE21 Workshop [30].

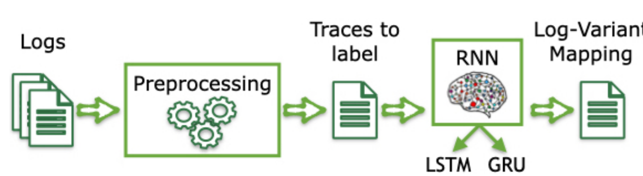

Figure 1: VaryMinions workflow

Results. we made the following contributions in this topic:

- a first experiment of the usage of Long Short Term Memory (LSTMs) [38] and Gated Recurrent Units (GRUs) [11], two RNN architectures, on two datasets (municipality management and travel expenses) showing that we can identify the variant(s) that could produce an event trace with a high accuracy (> 87%) and that there is no clear dominance of one network architecture;

- a characterisation of the learning difficulty based on behaviour sharing amongst event traces;
- an implementation of our approach exploiting the Tensorflow [21] and Keras [12] frameworks, our replication package and full results are available online [29].

Our evaluation addresses the following research questions:

- *How accurately can we identify process variants based on their traces?*

Answer: We were able to train RNNs providing an accuracy above 87% for both datasets. The following pairs of loss and activation functions stand out: MSE with tanh, MSE with sigmoid and binary cross-entropy combined with the sigmoid.

- *What is the performance of LSTMs versus that of GRUs for process traces classification?*

Answer: In the top combinations of both DS1 and DS2, performance of the LSTM and GRU varies significantly (*e.g.,* from 79% to 88% for GRU) and are mixed, with no absolute winner. Therefore, we cannot conclude on the prevalence of GRUs over LSTMs for our datasets.

Discussion. Our evaluation is limited to the identification of five variants and we need to determine if these promising results hold for a larger number of variants (*i.e.,* hundred or thousands in usual VIS). Our future plans includes: *i)* considering identifying features rather than complete variants, *ii)* the design of dedicated loss functions, and *iii)* the exploration of different neural architectures.

4.3 FTSLearnLib: Variability-aware Angluin-style Learning of FTS (WP1 & WP2)

In this study, we want to tackle the problems discussed in previous sections by offering a variability-aware model inference approach for FTS based on the seminal L^* algorithm and its extensions [5, 10]. In particular, we encode fragments of variability-aware behaviour as symbolic execution trees and take advantage of feature valuations to guide Angluin-style learning. This encoding does not require a merging step after some variant models have been learnt and maps FE directly in the resulting model.

Since we do not have a specification of the system, we cannot use *equivalence queries* to choose the right learned model among all the candidates. However, the *teacher* (Angluin's concept of oracle) can be provided with properties of the system to learn. These properties could take the form of negative examples as in the RPNI algorithm [26, 48]. These properties can be either given by the user, statically found by exploring feature interactions, or deduced from unit test executions or by learning metamorphic relations [55].

In short, we aim to provide:

(1) An Angluin-style algorithm definition treating variability as first-class citizen via specific encodings of FE;
(2) An implementation of this algorithm as an extension of the RALib automata learning library [10];
(3) Experimental results on several FTS demonstrating the feasibility of the proposed approach.

5 WORK PLAN

The first months of this PhD were partly dedicated to concluding the master thesis on concolic testing. We published "An SMT-Based Concolic Testing Tool for Logic Programs" [28] to the FLOPS 2020 conference. This work was a good introduction to source code analysis (used in WP3).

In addition to the work on WP1 and WP2 (Section 4), we participated to some scientific events, even if the COVID19 outbreak prevented physical attendance in 2020 and 2021: the Grascomp Doctoral Day, the 18th & 19th Belgium-Netherlands Software Evolution Workshop, the kick-off meeting of the "Software Velocity" (GDR-GPL, France) working group and both annual workshops of the EOS project (Belgium) on Verifying Learning Artificial Intelligence Systems. These events fostered new collaborations with experts, especially with Prof J.-F. Raskin (ULB, Belgium) with whom we are collaborating on the project FTSLearnLib.

In January 2020, the BigDat20 winter school was the opportunity to discover new interesting sub-fields of ML and data science in general. In particular, the introduction to process mining given by Prof W.M.P. Van der Aalst gave new perspectives to explore in our research project (notably on process mining) and incited the mapping study described previously.

There are also ongoing collaborations with other PhD students in our research group. For example, we aim to generate a behavioural interaction model in a variability-aware environment from multiples sources (code, component descriptors, etc.). A part of this work is focused on static code analysis techniques to infer relationships between different components (*i.e.,* features) of the system (WP3). The results were submitted to the 4th Context-aware, Autonomous and Smart Architecture Workshop[2].

2021-2023. The third year will focus on the evaluation of these strategies and the development of white-box ones. A detailed schedule is presented hereafter:

Sep. : Confirmation Exam and Open-Source Case Collection
Oct. - Nov. : FTSLearnLib (evaluation & publication writing)
Dec. - Jan. : Mapping Study (analysis & publication writing)
Feb. - Mar. : VaryMinions (Feature-based representation)
Apr. - July : White/Grey-box scenario (preliminary study, prototype development, evaluation & publication writing) (*WP 3*)
Aug. : Industrial Case Collection

The first part of the last year will be dedicated to an industrial validation of the prototypes (*WP 4*). The second part will be fully devoted to writing and defending the thesis.

ACKNOWLEDGEMENTS

Sophie Fortz is supported by the FNRS via a FRIA grant.

[2] ***Short paper (Submitted):*** Lima dos Santos, E, Fortz, S, Perrouin, G, & Schobbens, P-Y, 2021, 'Behavioral Maps: Towards Identifying Runtime Issues for Dynamic Software Product Lines', 4th Context-aware, Autonomous and Smart Architecture Workshop, 13-17 September 2021

REFERENCES

[1] Mathieu Acher, Benoit Baudry, Patrick Heymans, Anthony Cleve, and Jean-Luc Hainaut. 2013. Support for reverse engineering and maintaining feature models. In *VaMoS*, Stefania Gnesi, Philippe Collet, and Klaus Schmid (Eds.). ACM, 20.

[2] Mathieu Acher, Anthony Cleve, Gilles Perrouin, Patrick Heymans, Philippe Collet, and Charles Vanbeneden Lahire, Philippe. 2012. On Extracting Feature Models From Product Descriptions. In *VaMoS*. ACM, Leipzig, Germany.

[3] Hasan Ibne Akram, Colin La Higuera, and Claudia Eckert. 2012. Actively learning probabilistic subsequential transducers. In *International Conference on Grammatical Inference*. 19–33.

[4] Dana Angluin. 1987. Learning regular sets from queries and counterexamples. *Information and computation* 75, 2 (1987), 87–106.

[5] Dana Angluin. 1987. Learning regular sets from queries and counterexamples. *Information and Computation* 75, 2 (1987), 87–106. https://doi.org/10.1016/0890-5401(87)90052-6

[6] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. 2010. Automated Analysis of Feature Models 20 years later: A Literature Review. *Information Systems* 35, 6 (2010), 615 – 636.

[7] Fabian Benduhn, Thomas Thüm, Malte Lochau, Thomas Leich, and Gunter Saake. 2015. 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* (Hildesheim, Germany) *(VaMoS '15)*. Association for Computing Machinery, New York, NY, USA, 80–87. https://doi.org/10.1145/2701319.2701332

[8] Benedikt Bollig, Peter Habermehl, Carsten Kern, and Martin Leucker. 2009. Angluin-style learning of NFA. In *Twenty-First International Joint Conference on Artificial Intelligence*.

[9] J. C. A. M. Buijs, B. F. van Dongen, and W. M. P. van der Aalst. 2013. Mining Configurable Process Models from Collections of Event Logs. In *Business Process Management*, Florian Daniel, Jianmin Wang, and Barbara Weber (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 33–48.

[10] Sofia Cassel, Falk Howar, Bengt Jonsson, and Bernhard Steffen. 2014. Learning extended finite state machines. In *International Conference on Software Engineering and Formal Methods*. Springer, 250–264.

[11] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).

[12] François Chollet et al. 2015. Keras. https://keras.io.

[13] Andreas Classen, Quentin Boucher, and Patrick Heymans. 2010. A Text-based Approach to Feature Modelling: Syntax and Semantics of TVL. *Science of Computer Programming, Special Issue on Software Evolution* (2010). https://doi.org/10.1016/j.scico.2010.10.005

[14] Andreas Classen, Maxime Cordy, Pierre-Yves Schobbens, Patrick Heymans, Axel Legay, and Jean-Francois Raskin. 2013. Featured Transition Systems: Foundations for Verifying Variability-Intensive Systems and Their Application to LTL Model Checking. *IEEE Transactions on Software Engineering* 39, 8 (Aug. 2013), 1069–1089. https://doi.org/10.1109/TSE.2012.86

[15] Andreas Classen, Patrick Heymans, Pierre-Yves Schobbens, Axel Legay, and Jean-François Raskin. 2010. Model Checking Lots of Systems: Efficient Verification of Temporal Properties in Software Product Lines. In *ICSE*. ACM, 335–344. http://www.sbs.co.za/ICSE2010/

[16] Maxime Cordy, Andreas Classen, Gilles Perrouin, Patrick Heymans, Pierre-Yves Schobbens, and Axel Legay. 2012. Simulation-based abstractions for software product-line model checking. In *ICSE*. ACM.

[17] Maxime Cordy, Xavier Devroey, Axel Legay, Gilles Perrouin, Andreas Classen, Patrick Heymans, Pierre-Yves Schobbens, and Jean-François Raskin. 2019. *A Decade of Featured Transition Systems*. Springer International Publishing, Cham, 285–312. https://doi.org/10.1007/978-3-030-30985-5_18

[18] K. Czarnecki, S. Helsen, and U. Eisenecker. 2005. Formalizing Cardinality-based Feature Models and their Specialization. *Software Process Improvement and Practice* 10, 1 (2005), 7–29.

[19] Carlos Diego Nascimento Damasceno. 2019. Learning From Families: Inferring Behavioral Variability From Software Product Lines. In *International Conference on Integrated Formal Methods*. PhD Symposium.

[20] Carlos Diego N Damasceno, Mohammad Reza Mousavi, and Adenilso da Silva Simao. 2019. Learning to reuse: Adaptive model learning for evolving systems. In *International Conference on Integrated Formal Methods*. Springer, 138–156.

[21] TensorFlow Developers. 2021. *TensorFlow*. https://doi.org/10.5281/zenodo.4758419

[22] X. Devroey, M. Cordy, P. Schobbens, A. Legay, and P. Heymans. 2015. State machine flattening, a mapping study and tools assessment. In *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. 1–8. https://doi.org/10.1109/ICSTW.2015.7107408

[23] Xavier Devroey, Gilles Perrouin, Maxime Cordy, Hamza Samih, Axel Legay, Pierre-Yves Schobbens, and Patrick Heymans. 2015. Statistical prioritization for software product line testing: an experience report. *Software & Systems Modeling* (2015), 1–19. https://doi.org/10.1007/s10270-015-0479-8

[24] Xavier Devroey, Gilles Perrouin, Mike Papadakis, Axel Legay, Pierre-Yves Schobbens, and Patrick Heymans. 2016. Featured Model-based Mutation Analysis. In *Proceedings of the 38th International Conference on Software Engineering* (Austin, Texas) *(ICSE '16)*. ACM, New York, NY, USA, 655–666. https://doi.org/10.1145/2884781.2884821

[25] Deepak Dhungana, Paul Grünbacher, Rick Rabiser, and Thomas Neumayer. 2010. Structuring the modeling space and supporting evolution in software product line engineering. *Journal of Systems and Software* 83, 7 (2010), 1108–1122. https://doi.org/10.1016/j.jss.2010.02.018 SPLC 2008.

[26] Pierre Dupont. 1996. Incremental regular inference. In *International Colloquium on Grammatical Inference*. Springer, 222–237.

[27] Holger Eichelberger and Klaus Schmid. 2015. Mapping the design-space of textual variability modeling languages: a refined analysis. *International Journal on Software Tools for Technology Transfer* 17, 5 (2015), 559–584. https://doi.org/10.1007/s10009-014-0362-x

[28] Sophie Fortz, Fred Mesnard, Etienne Payet, Gilles Perrouin, Wim Vanhoof, and German Vidal. 2020. An SMT-Based Concolic Testing Tool for Logic Programs. In *International Symposium on Functional and Logic Programming*. Springer, 215–219.

[29] Sophie Fortz, Paul Temple, Xavier Devroey, Patrick Heymans, and Gilles Perrouin. 2021. VaryMinions. https://doi.org/10.5281/zenodo.5083334

[30] Sophie Fortz, Paul Temple, Xavier Devroey, Patrick Heymans, and Gilles Perrouin. 2021. VaryMinions: Leveraging RNNs to Identify Variants in Event Logs. In *Proceedings of the 5th International Workshop on Machine Learning Techniques for Software Quality Evolution (MaLTESQuE '21)*. ACM, New York, NY, USA. https://doi.org/10.1145/3472674.3473980 To appear.

[31] Gordon Fraser and Andreas Zeller. 2011. Exploiting Common Object Usage in Test Case Generation. In *2011 Fourth IEEE International Conference on Software Testing, Verification and Validation*. IEEE, 80–89. https://doi.org/10.1109/ICST.2011.53

[32] Joel Greenyer, Christian Brenner, Maxime Cordy, Patrick Heymans, and Erika Gressi. 2013. Incrementally Synthesizing Controllers from Scenario-based Product Line Specifications. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering* (Saint Petersburg, Russia) *(ESEC/FSE 2013)*. ACM, New York, NY, USA, 433–443. https://doi.org/10.1145/2491411.2491445

[33] Joel Greenyer, Amir Molzam Sharifloo, Maxime Cordy, and Patrick Heymans. 2012. Efficient consistency checking of scenario-based product-line specifications. In *2012 20th IEEE International Requirements Engineering Conference (RE)*. IEEE, 161–170.

[34] Vanderson Hafemann Fragal, Adenilso Simao, and Mohammad Reza Mousavi. 2017. Validated Test Models for Software Product Lines: Featured Finite State Machines. In *Formal Aspects of Component Software*, Olga Kouchnarenko and Ramtin Khosravi (Eds.). Springer International Publishing, Cham, 210–227. https://doi.org/10.1007/978-3-319-57666-4_13

[35] Axel Halin, Alexandre Nuttinck, Mathieu Acher, Xavier Devroey, Gilles Perrouin, and Patrick Heymans. 2017. Yo Variability! JHipster: A Playground for Web-apps Analyses. In *Proceedings of the Eleventh International Workshop on Variability Modelling of Software-intensive Systems* (Eindhoven, Netherlands) *(VAMOS '17)*. ACM, New York, NY, USA, 44–51. https://doi.org/10.1145/3023956.3023963

[36] Klaus Havelund and Thomas Pressburger. 2000. Model checking JAVA programs using JAVA PathFinder. *International Journal on Software Tools for Technology Transfer* 2, 4 (01 Mar 2000), 366–381. https://doi.org/10.1007/s100090050043

[37] Steffen Herbold, Patrick Harms, and Jens Grabowski. 2017. Combining usage-based and model-based testing for service-oriented architectures in the industrial practice. *International Journal on Software Tools for Technology Transfer* 19, 3 (jun 2017), 309–324. https://doi.org/10.1007/s10009-016-0437-y

[38] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[39] Falk Howar, Dimitra Giannakopoulou, and Zvonimir Rakamarić. 2013. Hybrid learning: interface generation through static, dynamic, and symbolic analysis. In *Proceedings of the 2013 International Symposium on Software Testing and Analysis*. ACM, 268–279.

[40] Falk Howar, Bengt Jonsson, and Frits Vaandrager. 2018. Combining black-box and white-box techniques for learning register automata. *Computing and Software Science. LNCS* 10000 (2018).

[41] K. Kang, SG Cohen, JA Hess, WE Novak, and A.S. Peterson. 1990. *Feature Oriented Domain Analysis (FODA)-Feasibility Study*. Technical Report. Technical report, Carnegie-Mellon University.

[42] Kyo C. Kang. 2010. FODA: Twenty Years of Perspective on Feature Modeling. In *VaMoS (ICB-Research Report, Vol. 37)*, David Benavides, Don S. Batory, and Paul Grünbacher (Eds.). Universität Duisburg-Essen, 9.

[43] Sami Lazreg, Maxime Cordy, Philippe Collet, Patrick Heymans, and Sébastien Mosser. 2019. Multifaceted Automated Analyses for Variability-intensive Embedded Systems. In *Proceedings of the 41st International Conference on Software Engineering* (Montreal, Quebec, Canada) *(ICSE '19)*. IEEE Press, Piscataway, NJ, USA, 854–865. https://doi.org/10.1109/ICSE.2019.00092

[44] Maikel Leemans, Wil M. P. van der Aalst, and Mark G. J. van den Brand. 2018. The Statechart Workbench: Enabling scalable software event log analysis using

process mining. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 502–506. https://doi.org/10.1109/SANER.2018.8330248

[45] Yang Li, Sandro Schulze, and Gunter Saake. 2017. Reverse Engineering Variability from Natural Language Documents: A Systematic Literature Review. In *Proceedings of the 21st International Systems and Software Product Line Conference - Volume A* (Sevilla, Spain) *(SPLC '17)*. ACM, New York, NY, USA, 133–142. https://doi.org/10.1145/3106195.3106207

[46] Roberto E. Lopez-Herrejon, Lukas Linsbauer, and Alexander Egyed. 2015. A systematic mapping study of search-based software engineering for software product lines. *Information and Software Technology* 61 (2015), 33 – 51. https://doi.org/10.1016/j.infsof.2015.01.008

[47] Raphaël Michel, Andreas Classen, Arnaud Hubaux, and Quentin Boucher. 2011. A Formal Semantics for Feature Cardinalities in Feature Diagrams. In *Proceedings of the Fifth International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'11), Namur, Belgium, January 27-29*. ACM Press, 83–90. http://info.fundp.ac.be/vamos2011/

[48] José Oncina and Pedro Garcia. 1992. Inferring regular languages in polynomial updated time. In *Pattern recognition and image analysis: selected papers from the IVth Spanish Symposium*. World Scientific, 49–61.

[49] K. Pohl, G. Böckle, and F. Van Der Linden. 2005. *Software product line engineering: foundations, principles, and techniques*. Springer-Verlag New York Inc.

[50] PureSystems. 2011. Pure::Variants Website http://www.pure-systems.com/.

[51] Harald Raffelt and Bernhard Steffen. 2006. Learnlib: A library for automata learning and experimentation. In *International Conference on Fundamental Approaches to Software Engineering*. Springer, 377–380.

[52] Julia Rubin and Marsha Chechik. 2012. Combining related products into product lines. In *International Conference on Fundamental Approaches to Software Engineering*. Springer, 285–300.

[53] Pierre-Yves Schobbens, Patrick Heymans, Jean-Christophe Trigaux, and Yves Bontemps. 2007. Generic semantics of feature diagrams. *Computer Networks* 51, 2 (2007), 456–479.

[54] Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing* 45, 11 (1997), 2673–2681. https://doi.org/10.1109/78.650093

[55] Sergio Segura, Gordon Fraser, Ana B Sanchez, and Antonio Ruiz-Cortés. 2016. A survey on metamorphic testing. *IEEE Transactions on software engineering* 42, 9 (2016), 805–824.

[56] Steven She, Rafael Lotufo, Thorsten Berger, Andrzej Wąsowski, and Krzysztof Czarnecki. 2011. Reverse engineering feature models. In *Proceedings of the 33rd International Conference on Software Engineering*. ACM, 461–470.

[57] Sharon Shoham, Eran Yahav, Stephen J Fink, and Marco Pistoia. 2008. Static specification mining using automata-based abstractions. *IEEE Transactions on Software Engineering* 34, 5 (2008), 651–666.

[58] Sara Sprenkle, Lori Pollock, and Lucy Simko. 2011. A Study of Usage-Based Navigation Models and Generated Abstract Test Cases for Web Applications. In *Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth International Conference on*. IEEE, 230–239. https://doi.org/10.1109/ICST.2011.34

[59] Sara E Sprenkle, Lori L Pollock, and Lucy M Simko. 2013. Configuring effective navigation models and abstract test cases for web applications by analysing user behaviour. *Software Testing, Verification and Reliability* 23, 6 (2013), 439–464. https://doi.org/10.1002/stvr.1496

[60] Andrew Stevenson and James R Cordy. 2012. Grammatical inference in software engineering: an overview of the state of the art. In *International Conference on Software Language Engineering*. Springer, 204–223.

[61] Farbod Taymouri, Marcello La Rosa, Marlon Dumas, and Fabrizio Maria Maggi. 2021. Business process variant analysis: Survey and classification. *Knowledge-Based Systems* 211 (2021), 106557. https://doi.org/10.1016/j.knosys.2020.106557

[62] Paolo Tonella, Alessandro Marchetto, Cu Duy Nguyen, Yue Jia, Kiran Lakhotia, and Mark Harman. 2012. Finding the optimal balance between over and under approximation of models inferred from execution logs. In *Proceedings of the 5th IEEE International Conference on Software Testing, Verification and Validation, ICST 2012*. IEEE, 21–30. https://doi.org/10.1109/ICST.2012.82

[63] Paolo Tonella, Roberto Tiella, and Cu Duy Nguyen. 2014. Interpolated n-grams for model based testing. In *Proceedings of the 36th International Conference on Software Engineering - ICSE 2014*. ACM Press, 562–572. https://doi.org/10.1145/2568225.2568242

[64] Raja Vallée-Rai, Phong Co, Etienne Gagnon, Laurie Hendren, Patrick Lam, and Vijay Sundaresan. 1999. Soot - a Java Bytecode Optimization Framework. In *Proceedings of the 1999 Conference of the Centre for Advanced Studies on Collaborative Research* (Mississauga, Ontario, Canada) *(CASCON '99)*. IBM Press, 13–. http://dl.acm.org/citation.cfm?id=781995.782008

[65] Wil van der Aalst. 2016. *Process Mining: Data Science in Action* (2nd ed.). Springer Publishing Company, Incorporated.

[66] Mahsa Varshosaz, Lars Luthmann, Paul Mohr, Malte Lochau, and Mohammad Reza Mousavi. 2019. Modal transition system encoding of featured transition systems. *Journal of Logical and Algebraic Methods in Programming* 106 (2019), 1–28. https://doi.org/10.1016/j.jlamp.2019.03.003

[67] Sicco Verwer and Christian A Hammerschmidt. 2017. flexfringe: A Passive Automaton Learning Package. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, L O'Conner (Ed.). IEEE, 638–642. https://doi.org/10.1109/ICSME.2017.58