Anomaly Detection and Explanation in Context-Aware Software Product Lines

Jacopo Mauro
University of Oslo
Norway

Michael Nieke
Technische Universität Braunschweig
Germany

Christoph Seidl
Technische Universität Braunschweig
Germany

Ingrid Chieh Yu
University of Oslo
Norway

ABSTRACT

A software product line (SPL) uses a variability model, such as a feature model (FM), to describe the configuration options for a set of closely related software systems. Context-aware SPLs also consider possible environment conditions for their configuration options. Errors in modeling the FM and its context may lead to anomalies, such as dead features or a void feature model, which reduce if not negate the usefulness of the SPL. Detecting these anomalies is usually done by using Boolean satisfiability (SAT) that however are not expressive enough to detect anomalies when context is considered. In this paper, we describe `HyVarRec`: a tool that relies on Satisfiability Modulo Theory (SMT) to detect and explain anomalies for context-aware SPLs.

CCS CONCEPTS

• **Software and its engineering → Software product lines**; **Software verification and validation**;

KEYWORDS

Context-Aware Software Engineering, Anomaly Detection

ACM Reference format:
Jacopo Mauro, Michael Nieke, Christoph Seidl, and Ingrid Chieh Yu. 2017. Anomaly Detection and Explanation in Context-Aware Software Product Lines. In *Proceedings of SPLC '17, Sevilla, Spain, September 25-29, 2017,* 4 pages.
DOI: 10.1145/3109729.3109752

1 INTRODUCTION

A *software product line (SPL)* [17] uses a variability model, such as a *feature model (FM)* [10], to describe the configuration options for a set of closely related software systems, sometimes also using attributes with finite domains for individual features [4]. *Cross-tree constraints* may express further configuration rules, e.g., in the form of Boolean formulas [1]. A *context-aware SPL* [12] incorporates dependence on certain *context values* into the configuration options of an FM. This can be captured using *validity formulas (VFs)* [12], which specify a propositional formula that defines the condition under which a feature may be selected and, thus, constitute a further way of influencing configuration options. While users can explicitly (de)select features and set attribute values when creating a configuration, concrete context values are instead determined externally so that FM users have no direct influence on them.

Errors in the creation of FMs, their cross-tree constraints or VFs may lead to anomalies, such as *Void Feature Model* when the root feature of the FM cannot be selected or *Dead Feature* when a feature cannot be selected in any possible configuration [11]. As these anomalies severely reduce, if not negate, the usefulness of an SPL, procedures to determine and explain them are essential. Different tools have been developed to be able to detect anomalies in an FM. In the general case, assuming that the cross-tree constraints are powerful enough, the anomaly detection problem is an NP-complete problem that is usually tackled by encoding the FM and its cross-tree constraints into a Boolean satisfiability (SAT) or Constraint Satisfaction Problems (CSP), relying on respective solvers for determining a solution [3]. However, the introduction of context to FMs significantly increases the complexity of anomaly detection as developers need to guarantee that there is no anomaly for all configurations *and* all possible combinations of context values. Unfortunately, the need of a universal quantification over context does not allow the simple use of SAT and CSP solvers to detect anomalies. In this paper, we address this problem by utilizing *satisfiability modulo theories (SMT)* [5] instead of SAT. We describe how to encode a context-aware FM and its cross-tree constraints as an SMT instance (§3) and we introduce `HyVarRec` (§4): a tool that allows to detect and explain anomalies in context-aware FMs with attributes. A preliminary validation of `HyVarRec` is presented (§5) before giving some concluding remarks.

2 RUNNING EXAMPLE

To demonstrate the concepts of this paper, we introduce a running example based on a real-world scenario from our partners in the automotive industry. Figure 1 shows an excerpt from the FM describing the configuration options of a car. The car supports different emergency call systems – one for Europe (`eCall Europe`) and one for Russia (`ERA/GLONASS Russia`). GPS is used as positioning service in Europe and, therefore, the feature `eCall Europe` requires the `GPS` feature. Accordingly, GLONASS is used as positioning service in Russia and, therefore, the feature `ERA/GLONASS`

SPLC '17, Sevilla, Spain

DOI: 10.1145/3109729.3109752

Russia requires the feature GLONASS. In the FM, these dependencies are expressed as cross-tree constraints. The feature Adaptive Cruise Control provides an assistance system to automatically hold a set speed. It has an attribute maxSpeed, which defines the maximum speed settable for cruise control. Moreover, the value of maxSpeed is influenced by the road condition. To model the impact of the environment on the software system of the car, the SPL is aware of two contexts: Road, which represents the state of the road, and Location, which represents the current position of the car. The

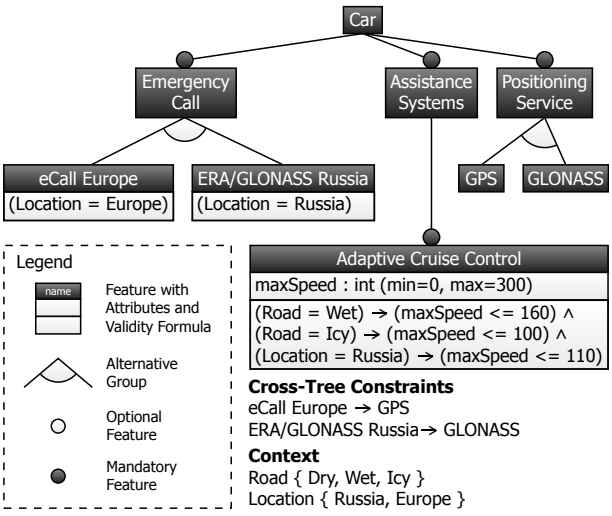


Figure 1: Example of context-aware FM for a car.

configuration logic of the car depends on the current context values. For example, the formula Location = Europe associated to the feature eCall Europe states that this feature may only be selected if the car is located in Europe. These formulas are called *validity formulas* (VFs) [12]. This is a simple FM and it is easy to check that every context allows a possible configuration. However, when dealing with large and complex FMs, it is easy to inadvertently introduce anomalies. For example, considering the previous FM, if the context Location is extended with a new possible value China, no valid configuration can be computed for this context. Similarly, updating the Positioning Service feature by adding the BeiDou satellite system for China will result in dead features if Emergency Call is not modified accordingly.

3 ANOMALY DETECTION: FROM SAT TO SMT

The anomalies of an SPL can be reduced to the Void Feature Model anomaly, i.e., when the root feature of the FM cannot be selected thus forbidding the existence of any possible configuration [3, 11]. For this reason, in this work, we focus solely on detection and explanations of the Void Feature Model anomaly.

Ensuring that the Void Feature Model does not happen for all the possible combinations of context values requires the check of a universally quantified formula. From a theoretical point of view, this implies an increase of the complexity of the check (i.e., the problem becomes Σ_2 complete [16] instead of NP-complete). From the practical point of view, instead, it means that it is not possible to directly use the SAT solvers as they support only existentially quantified formulas but not universally quantified formulas. Hence, to analyze the FM, we have to rely on a more powerful logic: Satisfiability Modulo Theories (SMT) [5]. SMTs are a generalization of Boolean SAT formulas in which variables are replaced by predicates from a variety of underlying theories. Some SMT solvers natively support the quantified fragments of their logics.

Given a context-aware FM F, it is possible to obtain an SMT formula φ_F that is satisfiable iff the FM is not void for all possible combinations of context values. To obtain this, following what is usually done to encode FMs as SAT problem [2], for every feature f, it is possible to introduce a Boolean variable that is evaluated to true iff f is selected. In an analogous way, as context and attributes are assumed to have finite domains, they are encoded into an integer variable x with two constraint, i.e., $x \geq x_{min}$ and $x \leq x_{max}$, which limit its domain in the range $[x_{min}, x_{max}]$. As an example, the context Location of the running example is encoded with an integer variable that can take two values: 0 for Russia, 1 for Europe.

The SMT formula φ_F is a conjunction of constraints derived from the structure of the FM, the cross-tree constraints and the validity formulas. The structure of the FM can be captured by implications. For instance, the fact that if a feature f_1 is selected implies that its parent feature f_0 needs to be selected can be encoded with the implication $f_1 \rightarrow f_0$ and a mandatory child feature f_1 of a feature f_0 can be encoded with $f_0 \rightarrow f_1$. Cross-tree constraints may be used directly as constraints of the SMT formula while a validity formula φ associated to a feature f can be encoded as $f \rightarrow \varphi$.

The variables introduced for features and attributes are quantified existentially, while the variables for context values are quantified universally. The SMT formula can therefore be defined as:

$$\varphi_F = \forall c_1, \ldots c_n \,.\, \exists f_1 \ldots, f_m \,.\, \exists a_1, \ldots, a_h \,.$$
$$\varphi_{dom} \wedge \varphi_{F,struct} \wedge \varphi_{F,ctc} \wedge \varphi_{F,vf} \wedge f_{root}$$

In this encoding, $c_1, \ldots c_n$ are the variables associated with context values, $f_1 \ldots, f_m$ the variables associated with features and $a_1, \ldots, a_h$ the variables associated with the attributes. In addition, φ_{dom} are the inequalities constraints to bound the domain of context and attributes, $\varphi_{F,struct}$ are the constraints derived form the structure of F, $\varphi_{F,ctc}$ are the cross-tree constraints, $\varphi_{F,vf}$ are the constraints related to the validity formulas, and f_{root} is the variable associated with the root feature of F.

4 THE TOOL HYVARREC

HyVarRec is developed in Python, open source and available online.[1] It was originally conceived as a utility to configure FMs [12] by relying on constraint programming [18]. However, to support anomaly detection, it was rewritten to utilize the SMT solver Z3 [6]. HyVarRec can be deployed by using the Docker container technology and used as a microservice by POST http requests.[2] However, to facilitate its use, it is possible to use it via DarwinSPL,[3] a tool suite to develop evolving context-aware SPLs using the paradigm of validity formulas [14]. For example, Figure 2 depicts how the FM described in §2 can be defined in DarwinSPL. HyVarRec can be invoked by

[1] https://github.com/HyVar/hyvar-rec
[2] The input for running the tool is defined by the JSON schemas available at https://github.com/HyVar/hyvar-rec/blob/master/spec.
[3] https://github.com/HyVar/DarwinSPL

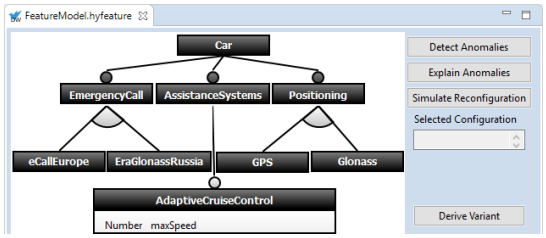


Figure 2: Integration of HyVarRec in DarwinSPL

pressing the `Detect Anomalies` and `Explain Anomalies` buttons in the upper right part of the figure.

4.1 Anomaly Detection

The JSON file received as input by HyVarRec is first used to translate the FM F into an SMT formula φ_F (§3) for the Z3 SMT solver. To check if φ_F is valid, HyVarRec negates it and searches for a solution of its negation. A solution of $\neg\varphi_F$ represents a set of context values for which the FM does not allow a valid configuration. If a solution is found, HyVarRec detects the problematic contexts and returns them to the user for further investigation on the causes of the anomaly.[4] For example, Figure 3 shows the output of HyVarRec visualized in DarwinSPL giving as input the FM defined in Section 2 with the additional location China.

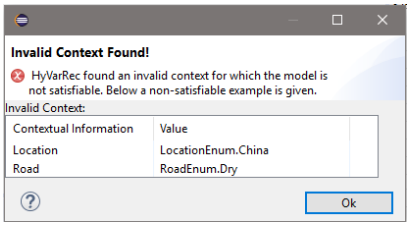


Figure 3: Visualized anomaly detection of HyVarRec

If no solution to the formula $\neg\varphi_F$ exists, HyVarRec notifies the user that the FM given in input has no anomalies.

4.2 Anomaly Explanation

When there is an anomaly and the FM has a significant size with even hundreds of features, manually determining the reason of the anomaly becomes a daunting task because the combination of possible reasons to check may be overwhelming. HyVarRec can assist in this procedure. After having detected the anomaly for a given context C using HyVarRec in the anomaly detection mode, users may run HyVarRec in explanation mode specifying the context C for which they want the explanation. In this modality, HyVarRec tries to find a solution for the formula φ_F after setting the contextual variables according to C, thus transforming the universally quantified formula into an existentially quantified one. At this point, HyVarRec finds a solution to the formula since we assumed that the FM F for context C does not permit a valid configuration. HyVarRec then extracts from the trace of the execution of the SMT solver the unsat core, i.e., the set of constraints that makes the formula unsatisfiable. This set of constraints is given in return to the user as explanation for the anomaly.

To assist users in detecting the source of the anomaly, HyVarRec provides no logically implied constraints but only those originally defined in the formula φ_F.

5 PRELIMINARY VALIDATION

In this section, we present the preliminary tests performed to validate HyVarRec. To the best of our knowledge, due to the novelty of these approaches, there are neither established benchmarks nor big industrial instances of context-aware FMs. Due to this reason, following common practice [3], to have at least a preliminary validation of HyVarRec, we benchmarked it against random generated context-aware FMs. The FMs were generated using the AFMwC tool,[5] i.e., an extension of the BeTTy FM generator [19], which also creates contexts and validity formulas. For the first benchmark, we generated 100 FMs each with 100 features (`Bench100`), while for the second, we generated 100 FMs each with 500 features (`Bench500`). We used the generator with default parameters for fixing the number of cross-tree constraints and validity formulas. For this purpose, up to 10 contexts were generated randomly, each context having possibly up to 10 values. HyVarRec was run on every single instance with a time cap of 1 hour on an Intel i7-5600U CPU 4 core machine having 8GB RAM and Ubuntu 16.04 operating system. For every instance in the benchmark, we first check if the FM was valid and, if not, we run HyVarRec in explanation mode to find out why.

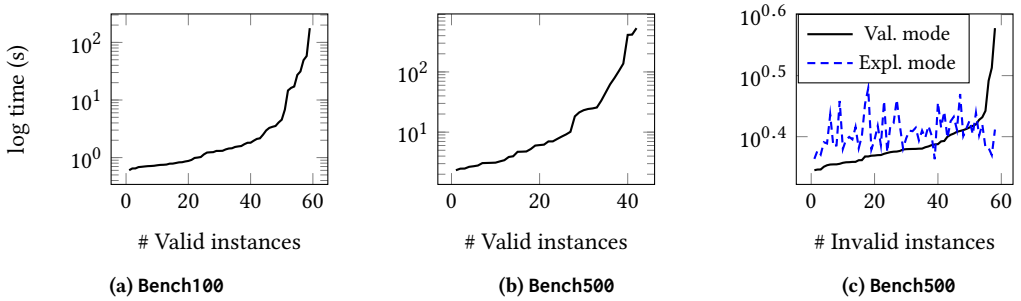
HyVarRec proved that 41 instances of `Bench100` are invalid while for `Bench500` 58 instances were proven invalid. As far as `Bench100` is concerned, we noticed that the times taken by HyVarRec to find that instances were invalid was less than a second. In particular, in the worst case, HyVarRec took 0.79 seconds to prove the invalidity, and 0.98 seconds to provide an explanation. When the instance was valid instead, HyVarRec took longer for few instances: 8 instances out of 59 required more than 10 seconds, 1 required even 177 seconds. This was to be expected as proving validity of an FM requires to explore all possible combination of context values, while proving invalidity only requires finding one example. The times taken to prove the validity of the 59 instances is shown in Figure 4a. The x axis presents the instances ordered according to their solving time, shortest first (y axis presents times using a logarithm scale).

Similar results were obtained considering `Bench500`. Due to the bigger size of the FM, the times for detecting the invalidity of the model and to explain why were relatively bigger: from 2.21 to 3.78 seconds to prove invalidity, from 2.31 to 3.02 seconds to explain the anomaly. Figure 4c shows these times for the instances sorted by the times it took to prove their invalidity. For valid instances, as before, HyVarRec took longer. In this case, to prove the validity, 11 out of 42 instances required a time between 10 and 100 seconds while 5 required instead a time between 100 and 534 seconds. These times are shown in Figure 4b.

Due to the fact that usually it requires few seconds to detect if there is an anomaly, we believe that HyVarRec can be used to handle FM with up to hundreds of features. Clearly, due to the

[4] In case some combinations of context values are not realistic (e.g., a wet road condition in a desertic area) HyVarRec also allows to exclude the check of these unrealistic scenarios.

[5] https://github.com/magnurh/AFMwC-thesisProject



(a) `Bench100` (b) `Bench500` (c) `Bench500`

Figure 4: `HyVarRec` solving times

complexity of the tests, developers need to be aware that checking the validity may require even minutes and not only seconds.

6 RELATED WORK & CONCLUSION

In this work we presented `HyVarRec`, a tool based on an SMT solver that allows to detect and explains anomalies in context-aware SPL. `HyVarRec` can be used as a standalone program or via a graphical interface. Preliminary results conducted on random generated instances show the feasibility of the approach.

Several technologies and methodologies to analyze FMs exist. We invite interested readers to [3] for a comprehensive survey on analyses techniques for SPLs and their tool support. In particular, as derived from the survey, we would like to point out that most of the tools missed two requirements: (i) to be able to deal with extended FMs, including feature attributes, and (ii) to provide explanations for defects and anomalies. As we are using an SMT solver, as in [20], we are able do deal with numerical values and, thus, are able to also support feature attributes. Moreover, as discussed in § 4.2, we are able to generate explanations for void FMs, similar to [11].

We are not aware of other tools to conduct analysis of context-aware SPLs. One of the closest approaches to ours is the UbiFex notation to model context-aware SPLs [7]. Thanks to the UbiFex-Simulator, it is possible to determine if the FM is void given a certain context. However, the authors claim that it is hard to reason for each possible context regarding the FM being void. With `HyVarRec`, we are instead able to do both: we can reconfigure the SPL given a configuration and a concrete context [15] and we are able to check for each possible context if we can generate a valid configuration.

Other works which focuses on modeling context-aware SPLs exist. For instance, in [9] context-awareness is captured by providing a second FM while in [13] ontologies are used instead to model the context. Unfortunately, these works just present these models without discussing their analysis.

As future work we are planning to extend `HyVarRec` to be incremental in order to speed up the checking of a sequence of analysis (e.g., verify that all the features are not dead) and to use minimal unsat core extractor such as [8] to search for the minimal possible explanation of an anomaly. We are also interested in establishing a real industrial context-aware benchmark that can be used to evaluate the performances of tools able to analyses context-aware SPL.

ACKNOWLEDGEMENTS

This work was partially supported by European Commission within the project HyVar (grant agreement H2020-644298) and by the Federal Ministry of Education and Research of Germany within the project CrESt (funding number 01IS16043S).

REFERENCES

[1] D. Batory. 2005. Feature Models, Grammars, and Propositional Formulas. *Software Product Lines* (2005).
[2] Don S. Batory. 2005. Feature Models, Grammars, and Propositional Formulas. In *SPLC (LNCS)*, Vol. 3714. Springer.
[3] David Benavides, Sergio Segura, and Antonio Ruiz Cortés. 2010. Automated analysis of feature models 20 years later: A literature review. *Inf. Syst.* 35, 6 (2010).
[4] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. 2005. Formalizing Cardinality-Based Feature Models and their Specialization. *Software Process: Improvement and Practice* 10, 1 (2005).
[5] Leonardo De Moura and Nikolaj Bjørner. 2011. Satisfiability Modulo Theories: Introduction and Applications. *Commun. ACM* 54, 9 (2011).
[6] Leonardo Mendonça de Moura and Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *TACAS (LNCS)*, Vol. 4963. Springer.
[7] Paula Fernandes and Cláudia Maria Lima Werner. 2008. UbiFEX: Modeling Context-Aware Software Product Lines. In *SPLC (Workshops)*. Lero Int. Science Centre, University of Limerick, Ireland.
[8] Ofer Guthmann, Ofer Strichman, and Anna Trostanetski. 2016. Minimal unsatisfiable core extraction for SMT. In *FMCAD*. IEEE.
[9] Herman Hartmann and Tim Trew. 2008. Using Feature Diagrams with Context Variability to Model Multiple Product Lines for Software Supply Chains. In *SPLC*. IEEE Computer Society.
[10] K.C. Kang, S.G. Cohen, J.A. Hess, W.E. Novak, and A.S. Peterson. 1990. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Technical Report. DTIC.
[11] Matthias Kowal, Sofia Ananieva, and Thomas Thüm. 2016. Explaining anomalies in feature models. In *GPCE*. ACM.
[12] Jacopo Mauro, Michael Nieke, Christoph Seidl, and Ingrid Chieh Yu. 2016. Context Aware Reconfiguration in Software Product Lines. In *VAMOS*. ACM.
[13] Sinisa Neskovic and Rade Matic. 2015. Context modeling based on feature models expressed as views on ontologies via mappings. *Comput. Sci. Inf. Syst.* 12, 3 (2015).
[14] Michael Nieke, Gil Engel, and Christoph Seidl. 2017. DarwinSPL: An Integrated Tool Suite for Modeling Evolving Context-aware Software Product Lines. In *VAMOS*. ACM.
[15] Michael Nieke, Jacopo Mauro, Christoph Seidl, and Ingrid Chieh Yu. 2016. User Profiles for Context-Aware Reconfiguration in Software Product Lines. In *ISoLA (LNCS)*, Vol. 9953.
[16] Christos H. Papadimitriou. 2007. *Computational complexity*. Academic Internet Publ.
[17] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. 2005. *Software Product Line Engineering - Foundations, Principles and Techniques*. Springer New York.
[18] Francesca Rossi, Peter van Beek, and Toby Walsh. 2006. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc.
[19] Sergio Segura, José A. Galindo, David Benavides, José Antonio Parejo, and Antonio Ruiz Cortés. 2012. BeTTy: benchmarking and testing on the automated analysis of feature models. In *International Workshop on Variability Modelling of Software-Intensive Systems*. ACM.
[20] Maurice H. ter Beek, Axel Legay, Alberto Lluch-Lafuente, and Andrea Vandin. 2015. Statistical analysis of probabilistic models of software product lines with quantitative constraints. In *SPLC*. ACM.