Symbolic Characterisation of Commonalities in Testing Software Product Lines

Sebastian Kunze
Centre for Research on Embedded Systems
Halmstad University
Halmstad, Sweden
sebastian.kunze@hh.se

***Abstract*—Validation and verification of Software Product Lines is particularly challenging due to the complex structure and interaction of commonalities and variabilities among products. There are several approaches to specify the structure of such commonalities and variabilities, such as the delta-oriented approach. Building upon such a structure, we propose an approach to avoid redundant analysis in Software Product Lines by extending them to semantic behavioural changes. To this end, we propose to use Differential Symbolic Execution, an automated technique for proving functional behavioural equivalence based on satisfiability modulo theories. Our proposal aims at identifying the behavioural commonalities of one software product relative to another and exploits them in order to establish an efficient model-based testing trajectory.**

I. Introduction

Software Product Line Engineering (SPLE) has proven to be beneficial to the development of embedded and safety-critical systems and has been applied successfully by several companies and institutions. The products of a *Software Product Line* (SPL) share certain commonalities and differentiate in terms of *variability points*, user-visible aspects or characteristics of products. SPLE promises distinct benefits, i.e., tailored products, reduced development costs, improved quality, and shorter time to market, but also gives rise to an complex structure and interaction among such products. This makes their validation and verification extremely challenging.

Analysing the correctness of safety-critical SPLs is of significant importance. For this particular reason, several analysis approaches on reducing the analysis effort have been developed and proposed recently; we refer to [1] for a detailed overview. Some earlier work propose (test) models that structure the commonalities and variabilities of an SPL and thereby structure the model-based testing process. We build upon such approaches and extend them in order to compare the behaviour of structural commonalities among different products.

To this end, we propose to exploit *Differential Symbolic Execution* (DSE) [2], which is capable of identifying shared and featured behaviour of two products and allows to describe their behavioural differences. Applied to SPLE, it can be used for characterising the behaviour of one product relative to another. Within our approach, the differential information is exploited to point out the test cases of a given test suite related to the shared behaviour for an analysed product accordingly. By focusing on the identified behavioural differences exclusively,

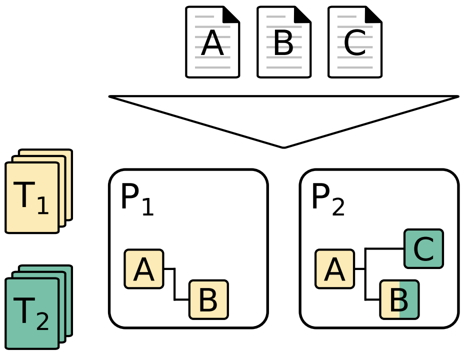

Fig. 1. A Software Product Line including the products P_1 and P_2. It illustrates possible feature combinations of the features A and B, describing common behaviour, and the elective feature C, describing optional behaviour. The behaviour of product P_1 is analysed through the test suite T_1 (coloured in yellow). By exploiting the behavioural differences, the analysis effort for product P_2 can be minimised to the test cases within test suite T_2 that are related to the syntactic and semantic changes (coloured in green), while reusing the test results of the unchanged part (coloured in yellow).

test case execution of optional behaviour is performed only once and reduces analysis effort ultimately.

The remainder of this paper is organised as follows: Section II outlines the problem and section III describes our approach briefly. An overview of related work is given in section IV. Finally, section V presents a future roadmap.

II. Problem Description

Typically, a sample of products is identified as a representative set of the whole SPL for analysis [1]. (Whether such a sample is indeed representative of the SPL behaviour is a related problem to our proposal, but it is beyond the scope of the present paper.) By analysing a particular product in this sample, parts of the common code and a set of the variability points are verified. Subsequently, analysing another product in this sample, some parts of the common code and a subset of the already verified variability points may be verified again. Our goal is to avoid those redundant analyses by identifying common parts whose behaviour has remained intact and focusing on the modified and added behaviour.

To illustrate this problem, consider the SPL shown in Fig. 1 including the sampled products P_1 and P_2. The SPL consists of the features A and B, describing the common behaviour, and the elective feature C, describing optional behaviour. Additionally, the behaviour of the feature B in product P_2 is modified due to the introduction of the feature C. Syntactic behavioural changes, i.e., the introduction of C, can simply be identified by using the structural information [3]; in our case using the delta-oriented approach. Complex feature interactions that introduce semantic behavioural changes, i.e., the modified behaviour in B, can easily be overlooked though. Our goal is to detect such changes and link them to the test-cases to be re-executed.

Consider the test suites T_1 and T_2 for the respective sampled products P_1 and P_2, which cover the products' control flow graphs. When exploiting the syntactic model, the test suite T_1 traverses the paths for the features A and B solely, whereas the paths for the feature C is traversed by the test suite T_2 exclusively. Although incorporating the syntactic models of the delta-oriented approach reduces redundant analysis effort, the syntactic models are inherently too imprecise to cover the semantic behavioural changes within feature B. We aim at remedying this imprecision by using a semantic analysis technique specified below.

III. Proposed Approach

A key characteristic of an efficient SPL analysis approach is to avoid redundant analysis steps for inherited commonalities [4]. Our approach comprises applying DSE to SPLE, in analysing behavioural differences by comparing one sampled product relative to another, representing the differential semantic information. This information is then used to transform the underlying test model of a product accordingly in order to save analysis effort for testing. Additionally, this information can be exploited to modify a given set of test cases and to execute change-related test cases exclusively.

By applying DSE to the SPL illustrated in Fig. 1, the test cases for syntactic and semantic behavioural changes within the sampled product P_2 (coloured in green) can be derived from the given test suites T_2, while reducing redundant analysis steps at the same time. Therefore, the differential information of product P_2 relative to product P_1 is created by comparing them to each other. Those changes, denoted as $\Delta_{P_2 \rightarrow P_1}$, describe the relative changes and are exploited to select the test cases covering changed behaviour within the given test suite T_2, further referred to as $T_{\Delta P_2}$.

When using the test suites T_1 and $T_{\Delta P_2}$ to analyse the SPL, the paths for the features A and B are covered by the test suite T_1 exclusively. The paths for the syntactic behavioural changes of the elective feature C as well as the paths for the semantic behavioural changes of the shared behaviour C are covered by the test suite $T_{\Delta P_2}$ then.

In summary, applying DSE to SPLE brings about three main advantages. Differential information can (a) be applied to select a number of test cases from a given test suite related to semantic behavioural changes, (b) be incorporated to optimise test case execution, and (c) be embodied to guide the overall analysis of an SPL. Consequently, redundant analysis steps are avoided and the analysis effort is reduced.

IV. Related Work

Classen et al. [5] developed a model checking algorithm based on *Featured Transition Systems* (FTS) and applied the principles of symbolic execution in [4] to address the state explosion problem. In contrast to our approach, they express the behaviour of all products of an SPL in one single model.

Lochau et al. [3] applied the principles of *Delta-Oriented Programming* to state machines to transform one product's state machine into another accordingly. This approach has been further refined by Lity et al. [6] to abstract parts of the program that do not influence the current point of interest. Compared to their approach, we focus on behavioural changes at the code level rather than abstract state machine models.

Lachmann et al. [7] presented an integration testing approach for SPLs, which focuses on structural changes between products. Unlike our approach, they do not focus on generating the test cases for each product, but select and prioritise them based on the SPL's specification instead.

V. Future Roadmap

Our first milestone is to formalise our approach based on the syntax and the formal semantics of DeltaJava, a delta-oriented framework for SPLs. We would like to implement our approach by translating our formal semantics into the DSE tool input and adapt the tool to take the SPL structure into account. Subsequently, we plan to apply the approach to case studies and analyse the data. Our goal is to verify the following theses:

- Coverage: the test cases generated for the behavioural differences cover semantic behavioural changes including the added, deleted, and modified behaviour of commonalities.
- Efficiency: the accumulated effort in applying structural analysis and in performing DSE is by far less than the saved effort by avoiding re-testing.

References

[1] T. Thüm, S. Apel, C. Kästner, I. Schaefer, and G. Saake, "A classification and survey of analysis strategies for software product lines," *ACM Computing Surveys (CSUR)*, vol. 47, no. 1, p. 6, 2014.

[2] S. Person, M. B. Dwyer, S. Elbaum, and C. S. Păsăreanu, "Differential symbolic execution," in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*. ACM, 2008, pp. 226–237.

[3] M. Lochau, I. Schaefer, J. Kamischke, and S. Lity, "Incremental model-based testing of delta-oriented software product lines," in *Tests and Proofs*. Springer, 2012, pp. 67–82.

[4] A. Classen, M. Cordy, P. Heymans, A. Legay, and P.-Y. Schobbens, "Formal semantics, modular specification, and symbolic verification of product-line behaviour," *Science of Computer Programming*, vol. 80, pp. 416–439, 2014.

[5] A. Classen, M. Cordy, P.-Y. Schobbens, P. Heymans, A. Legay, and J.-F. Raskin, "Featured transition systems: Foundations for verifying variability-intensive systems and their application to ltl model checking," *Software Engineering, IEEE Transactions on*, vol. 39, no. 8, pp. 1069–1089, 2013.

[6] S. Lity, H. Baller, and I. Schaefer, "Towards incremental model slicing for delta-oriented software product lines," in *Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on*. IEEE, 2015, pp. 530–534.

[7] R. Lachmann, S. Lity, S. Lischke, S. Beddig, S. Schulze, and I. Schaefer, "Delta-oriented test case prioritization for integration testing of software product lines," in *Proceedings of the 19th International Conference on Software Product Line*. ACM, 2015, pp. 81–90.