Strategies for Analyzing Configurable Systems

Alexander von Rhein, Sven Apel
University of Passau
{rhein, apel}@fim.uni-passau.de

Abstract: The advent of variability management and generator technology enables users to derive individual system variants from a given configurable system just based on a selection of desired configuration options. To cope with the possibly huge configuration space, researchers have been developing analysis techniques that follow different strategies to incorporate (static) variability. We discuss different strategies (variability-aware analysis and sampling) and evaluate them in different settings (model checking, type checking, and liveness analysis). A key finding is that variability-aware analysis outperforms most sampling approaches with respect to analysis time while being able to make definite statements about all variants of a configurable system.

Generator-based approaches have proved successful for the implementation of configurable software systems [CE00, AK09]. For example, the Linux kernel can be configured by means of about 10 000 compile-time configuration options, giving rise to possibly billions of variants that can be generated and compiled on demand. While advances in variability management and generator technology facilitate the development of configurable software systems with myriads of variants, this high degree of variability is not without cost. How could we analyze all possible variants for defects (type errors, division-by-zero errors, ...)? Classic analyses consider all (or only some) individual variants and do not scale to the possibly exponential number of variants that can be generated from a configurable system [TAK+14].

In the face of this huge degree of variability, it is imperative to tailor existing analysis approaches to the specifics of configurable systems. A brute-force strategy to analyze an entire configurable system, called *variant-based strategy*, is to create and analyze each product individually. This is the baseline of our experiments. A practical alternative, the *sample-based* strategy, focuses on a subset of possible feature combinations (i.e., products) to reduce the analysis load and to identify defects faster. While sampling reduces the analysis effort significantly, the information obtained is necessarily incomplete. A third alternative strategy, called *family-based strategy*, is to analyze the design and implementation artifacts of a configurable system as a whole (i.e., product family) in a single pass (e.g., by creating a simulator that simulates the behavior of all individual variants), incorporating configuration knowledge [LTP09, CHS+10, ASW+11].

We conducted a series of experiments based on 9 case studies to compare the sample-based strategy and the family-based strategy (with the variant-based strategy as a base line). Specifically, we used a model-checking analysis and conducted experiments to compare the verification performance and the ability to identify defects of all three strategies [AvW+13]. We concentrated on undesired *feature interactions*, a specific class of defects that is especially challenging for verification, because they emerge between mul-

tiple features and not within individual features [CKMRM03]. In a second series of experiments, we evaluated different family-based analyses for C on real world systems (LINUX, BUSYBOX, and OPENSSL) [LvK+13]. In particular, we implemented and evaluated family-based type checking and liveness analysis (a standard data-flow analysis) based on the variability-aware parsing framework TYPECHEF [KGR+11].

Overall, we found that family-based analyses scale well—and even outperform some of the sampling analyses—while still providing complete information on all system variants. Our experiments also revealed that the success of a sample-based strategy depends on how many variants of the system contain defects and how many variants are chosen in the sample set; The success of the family-based strategy depends on the similarity between variants, represented by the potential for handling analysis facts for multiple variants at the same time. While sampling can reduce the verification time significantly, this does not necessarily increase the effectiveness of verification, because many defective system variants may be missed. Finally, for large systems, such as the LINUX kernel, we observed that sampling does not scale as already calculating a meaningful sample set is a great obstacle because of the large number of variants.

References

[AK09] S. Apel and C. Kästner. An Overview of Feature-Oriented Software Development. *J. Object Technology*, 8(5):49–84, 2009.

[ASW+11] S. Apel, H. Speidel, P. Wendler, A. von Rhein, and D. Beyer. Detection of Feature Interactions using Feature-Aware Verification. In *Proc. ASE*, pages 372–375. IEEE, 2011.

[AvW+13] S. Apel, A. von Rhein, P. Wendler, A. Größlinger, and D. Beyer. Strategies for Product-Line Verification: Case Studies and Experiments. In *Proc. ICSE*, pages 482–491. IEEE, 2013.

[CE00] K. Czarnecki and U. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, 2000.

[CHS+10] A. Classen, P. Heymans, P.-Y. Schobbens, A. Legay, and J.-F. Raskin. Model Checking Lots of Systems: Efficient Verification of Temporal Properties in Software Product Lines. In *Proc. ICSE*, pages 335–344. ACM, 2010.

[CKMRM03] M. Calder, M. Kolberg, E. Magill, and S. Reiff-Marganiec. Feature Interaction: A Critical Review and Considered Forecast. *Computer Networks*, 41(1):115–141, 2003.

[KGR+11] C. Kästner, P. Giarrusso, T. Rendel, S. Erdweg, K. Ostermann, and T. Berger. Variability-aware Parsing in the Presence of Lexical Macros and Conditional Compilation. In *Proc. OOPSLA*, pages 805–824. ACM, 2011.

[LTP09] K. Lauenroth, S. Toehning, and K. Pohl. Model Checking of Domain Artifacts in Product Line Engineering. In *Proc. ASE*, pages 269–280. IEEE, 2009.

[LvK+13] J. Liebig, A. von Rhein, C. Kästner, S. Apel, J. Dörre, and C. Lengauer. Scalable Analysis of Variable Software. In *Proc. ESEC/FSE*, pages 81–91. ACM, 2013.

[TAK+14] T. Thüm, S. Apel, C. Kästner, I. Schaefer, and G. Saake. A Classification and Survey of Analysis Strategies for Software Product Lines. *Computing Surveys*, 47(1):6:1–6:45, 2014.