# On Automated N-way Program Merging for Facilitating Family-based Analyses of Variant-rich Software

Dennis Reuling[1] Udo Kelter[1] Johannes Bürdek[2] Malte Lochau[2]

**Abstract:** In this work, we report about research results initially published in ACM Transactions on Software Engineering and Methodology (TOSEM), volume 28 Issue 3, 2019 [Re19]. Nowadays software comes in many different, yet similar variants, often derived from common code via clone-and-own. Family-based-analysis strategies show promising potentials for improving efficiency of quality assurance for variant-rich programs, as compared to variant-by-variant approaches. Unfortunately, these strategies require one superimposed program representation containing all program variants in a syntactically well-formed, semantically sound, and variant-preserving manner, which is hard to obtain manually in practice. In this talk, we present our methodology SiMPOSE for generating superimpositions of program variants to facilitate family-based analyses of variant-rich software. We utilize a novel N-way model-merging methodology for control-flow automaton (CFA) representations of C programs, an abstraction used by many recent software-analysis tools. To cope with the complexity of N-way merging, we use similarity-propagation to reduce the number of N-way matches and enable incremental merging of arbitrary subsets of variants. We apply our SiMPOSE tool to realistic C programs and investigate applicability and efficiency/effectiveness trade-offs of family-based program analyses. Our results reveal efficiency improvements by a factor of up to 2.6 for unit-test generation and 2.4 for model-checking under stable effectiveness, as compared to variant-by-variant.

**Keywords:** Program Merging, Model Matching, Variability Encoding, Quality Assurance

## 1 Summary

Software-product-line engineering is a comprehensive methodology to cope with the inherent complexity of variant-rich software, by making explicit common and variable parts in a family of program variants. Product-line engineering facilitates systematic *reuse* of code fragments among variants to increase productivity and quality of software development, as compared to variant-by-variant or clone-and-own approaches. Novel techniques for also lifting quality-assurance techniques (e.g., unit testing and model-checking) to variant-rich software pursue so-called *family-based* analysis strategies [Th14]. The goal is to analyze all program variants in a single run, instead of considering every variant separately one after the other. However, these techniques often require a (virtual) integration of all program variants into one *superimposed* representation satisfying several requirements: it has to be a)

[1] University of Siegen, Software Engineering Group, Siegen, Germany, dreuling@informatik.uni-siegen.de, kelter@informatik.uni-siegen.de
[2] TU Darmstadt, Real-time Systems Lab, Darmstadt, Germany, malte.lochau@es.tu-darmstadt.de, johannes.buerdek@es.tu-darmstadt.de

doi:10.18420/SE2020_14

syntactically well-formed, b) semantically sound (e.g., its functionality corresponds to the union of functionality of all variants), c) variant-preserving (e.g., meta-data for tracing back to program variants) and d) sufficiently succinct (e.g., all parts shared among variants are identified and integrated to maximize reuse). Most recent approaches either apply $N$-way merging to superimpose $N$ design models instead of programs, or perform purely syntactic matching on locally restricted program fragments. The latter yields inherently imprecise (i.e., non-succinct or even unsound) merges as well as merge conflicts that are not automatically resolvable. Conversely, succinct $N$-way merges are, in general, not efficiently computable due to the combinatorial explosion of the number of possible matches [RC13]. Finally, existing approaches are either not variant-preserving, or utilize *compile-time variability* like #ifdef directives which is often incompatible with family-based analyses tools.

Our novel methodology SIMPOSE allows for automatically superimpose $N$ program variants for enabling family-based analysis. SIMPOSE comprises a novel $N$-way model merging algorithm for control-flow automata (CFA) representations of programs. To cope with the complexity of $N$-way comparison, matching and merging of path-based program models like CFA, our approach uses principles of *similarity propagation* for a controllable trade-off between precision and computational effort. To meet requirements a)–d), we semantically embed variability-information as conditional CFA patterns into the merged CFA. SIMPOSE further enables *compositional* merging such that $N$-way program merging can be decomposed into incremental merging steps. Our SIMPOSE tool integrates our novel technique with a tool for family-based program analysis using the C model-checker CPACHECKER. Our experiments show that SIMPOSE outperforms state-of-the-art algorithms GNU Diffutils (DIFF) in terms of precision and N-way model merging (NWM) [RC13] in terms of scalability thus constituting, on average, the best efficiency/effectiveness trade-off between efficiency and effectiveness. The results further reveal efficiency improvements by a factor of up to 2.6 for unit-test generation and 2.4 for model-checking under stable effectiveness, as compared to variant-by-variant approaches.

***Acknowledgments.*** This work was partially supported by the DFG (German Research Foundation) within the CoMoVa project (grant nr 330452222). This work was funded by the Hessian LOEWE initiative within the Software Factory 4.0 project.

## Bibliography

[RC13] Rubin, J.; Chechik, M.: N-way Model Merging. In: Proceedings of the 2013 Joint Meeting on Foundations of Software Engineering. 2013.

[Re19] Reuling, Dennis; Kelter, Udo; Bürdek, Johannes; Lochau, Malte: Automated N-way Program Merging for Facilitating Family-based Analyses of Variant-rich Software. ACM Trans. Softw. Eng. Methodol., 28(3):13:1–13:59, July 2019.

[Th14] Thüm, T.; Apel, S.; Kästner, C.; Schaefer, I.and Saake, G.: A Classification and Survey of Analysis Strategies for Software Product Lines. ACM Comput. Surv., 2014.