SPL\textsuperscript{LIFT} — Statically Analyzing Software Product Lines in Minutes Instead of Years

Eric Bodden\textsuperscript{1}  Társis Tolêdo\textsuperscript{3}  Márcio Ribeiro\textsuperscript{3,4}  
Claus Brabrand\textsuperscript{2}  Paulo Borba\textsuperscript{3}  Mira Mezini\textsuperscript{1}

\textsuperscript{1}EC SPRIDE, Technische Universität Darmstadt, Darmstadt, Germany  
\textsuperscript{2}IT University of Copenhagen, Copenhagen, Denmark  
\textsuperscript{3}Federal University of Pernambuco, Recife, Brazil  
\textsuperscript{4}Federal University of Alagoas, Maceió, Brazil

boden@acm.org, \{twt, phmb\}@cin.ufpe.br, marcio@ic.ufal.br  
brabrand@itu.dk, mira.mezini@cased.de

Abstract: A software product line (SPL) encodes a potentially large variety of software products as variants of some common code base. Up until now, re-using traditional static analyses for SPLs was virtually intractable, as it required programmers to generate and analyze all products individually. In this work, however, we show how an important class of existing inter-procedural static analyses can be transparently lifted to SPLs. Without requiring programmers to change a single line of code, our approach SPL\textsuperscript{LIFT} automatically converts any analysis formulated for traditional programs within the popular IF\textsuperscript{DS} framework for inter-procedural, finite, distributive, subset problems to an SPL-aware analysis formulated in the IDE framework, a well-known extension to IF\textsuperscript{DS}. Using a full implementation based on Heros, Soot, CIDE and JavaBDD, we show that with SPL\textsuperscript{LIFT} one can reuse IF\textsuperscript{DS}-based analyses without changing a single line of code. Through experiments using three static analyses applied to four Java-based product lines, we were able to show that our approach produces correct results and outperforms the traditional approach by several orders of magnitude.

A Software Product Line (SPL) describes a set of software products as variations of a common code base. Variations, so-called features, are typically expressed through compiler directives such as the well-known \#ifdef from the C pre-processor or other means of conditional compilation.

Static program analyses are a powerful tool to find bugs in program code [GPT\textsuperscript{+11}, FYD\textsuperscript{+08}] or to conduct static optimizations [SHR\textsuperscript{+00}], and it is therefore highly desirable to apply static analyses also to software product lines. With existing approaches, though, it is often prohibitively expensive to reuse existing static analyses. The problem is that traditional static analyses cannot be directly applied to software product lines. Instead they have to be applied to pre-processed programs. But for an SPL with \(n\) optional features, there are \(2^n\) possible products, which therefore demands thousands of analysis runs even for small product lines. This exponential blowup is particularly annoying because many of those analysis runs will have large overlaps for different feature combinations. It therefore seems quite beneficial to share analysis information wherever possible.
In this work we introduce \textsc{SPL}^{LIFT}, a simple but very effective approach to re-using existing static program analyses without an exponential blowup. \textsc{SPL}^{LIFT} allows programmers to transparently lift an important class of existing static analyses to software product lines. Our approach is fully inter-procedural. It works for any analysis formulated for traditional programs within Reps, Horwitz and Sagiv’s popular \textsc{IFDS} [RHS95] framework for inter-procedural, finite, distributive, subset problems. \textsc{SPL}^{LIFT} automatically converts any such analysis to a feature-sensitive analysis that operates on the entire product line in one single pass. The converted analysis is formulated in the \textsc{IDE} framework [SRH96] for inter-procedural distributed environment problems, an extension to \textsc{IFDS}. In cases in which the original analysis reports that a data-flow fact $d$ may hold at a given statement $s$, the resulting converted analysis reports a feature constraint under which $d$ may hold at $s$.

At \url{http://bodden.de/spllift/} we make available our full implementation as open source, along with all data and scripts to reproduce our empirical results. To summarize, our approach presents the following original contributions:

- a mechanism for automatically and transparently converting any \textsc{IFDS}-based static program analysis to an \textsc{IDE}-based analysis over software product lines,
- a full open-source implementation for Java, and
- a set of experiments showing that our approach yields correct results and outperforms the traditional approach by several orders of magnitude.

Out work on \textsc{SPL}^{LIFT} was first published at PLDI 2013 [BTR+13].

References


