Specialization Slicing

Min Aung[†], Susan Horwitz[†], Rich Joiner[†], and Thomas Reps[†,‡]

[†]University of Wisconsin-Madison [‡]GrammaTech, Inc.

{aung, horwitz, joiner, reps}@cs.wisc.edu

Abstract

In this paper, we investigate opportunities to be gained from broadening the definition of program slicing. A major inspiration for our work comes from the field of partial evaluation, in which a wide repertoire of techniques have been developed for specializing programs. While slicing can also be harnessed for specializing programs, the kind of specialization obtainable via slicing has heretofore been quite restricted, compared to the kind of specialization allowed in partial evaluation. In particular, most slicing algorithms are what the partial-evaluation community calls *monovariant*: each program element of the original program generates *at most one element* in the answer. In contrast, partial-evaluation algorithms can be *polyvariant*, i.e., one program element in the original program may correspond to more than one element in the specialized program.

The full paper appears in *ACM TOPLAS 36*(2), 2014.

Categories and Subject Descriptors D.3.3 [*Programming Languages*]: Language Constructs and Features—Control structures, procedures, functions, and subroutines, recursion; F.1.1 [*Computation by Abstract Devices*]: Models of Computation—Automata; F.3.2 [*Logics and Meanings of Programs*]: Semantics of Programming Languages—Partial evaluation, program analysis

Summary

This paper defines a new variant of program slicing, called *specialization slicing*, and presents an algorithm for creating an *optimal* specialization slice. The algorithm is polyvariant: for a given procedure p, the algorithm may create multiple specialized copies of p. In creating specialized procedures, the algorithm must decide for which patterns of formal parameters p should be specialized, and which program elements should be included in each specialized copy of p. Specialization slicing represents a new point in the "design space" of slicing problems. The algorithm still has the main characteristics of a slicing algorithm—that is, the elements of the output slice are all elements from the input program; no evaluation or simplification is performed. Our work adopts just one feature from the partial-evaluation literature—polyvariance—and studies how that extension changes the slicing problem.

In the full paper, we define specialization slicing, describe an elegant algorithm for solving the problem, and present results from studying specialization slicing from a number of angles.

PLDI '14, Jun 9–11 2014, Edinburgh, United Kingdom.
Copyright © 2014 ACM 978-1-4503-2784-8/14/06...$15.00.
http://dx.doi.org/10.1145/http://dx.doi.org/10.1145/2594291.2594345

- We formalize the problem of specialization slicing as a partitioning problem on the elements of the (possibly infinite) unrolled program (§3.1). We give definitions of *soundness*, *completeness*, and *minimality* for specialization slicing (§3.1).
- To represent finitely the infinite sets of objects that we need to manipulate to solve the partitioning problem, we make use of symbolic techniques originally developed in the model-checking community. Using this machinery, we give an algorithm in §4 that with just a few simple automata-theoretic operations identifies
  - the minimal set of specialized procedures that capture each of the different patterns of behavior for a given procedure, as well as
  - the minimal set of program elements required in each specialized procedure.
- We prove that our specialization-slicing algorithm is sound and complete, and returns a minimal specialization slice (§4.4 and Appendix A); consequently, the algorithm always creates an optimal output slice (§5.1).
- We characterize the time and space used by the algorithm (§5).
  - We present a family of examples for which the running time and space of the algorithm can be exponential in certain parameters of the input program (§5.3).
  - Our experience to date has been that neither such examples, nor the worst-case exponential behavior of operations like automaton determinization, arise in practice. Hence, we believe it is fair to say that, for the *observed cost*, both the running time and space of the algorithm are bounded by the sum of two terms: one is polynomial in the size of the input program; the other is linear in the size of the output slice.
- The specialization-slicing algorithm provides a new way to create *executable* slices—in particular, it creates polyvariant executable slices (§6).
- We describe several extensions of the basic algorithm:
  - We describe how to extend the algorithm to handle programs that (i) make calls to library procedures (§7.1), and (ii) make calls via pointers to procedures (§7.2).
  - We show that the algorithm possesses a kind of idempotence property (§7.3).
  - We show how to speed up one of the key steps of the algorithm (§7.4).
- We describe a method for removing unwanted program features (§8). The method uses specialization slicing in conjunction with forward slicing. While it was previously known how to solve the feature-removal problem for single-procedure programs, no algorithm was known for multi-procedure programs.
- In §9, we present the results of experiments using C programs to evaluate (i) our specialization-slicing algorithm (for polyvariant executable slicing), and (ii) an algorithm for monovariant executable slicing.

§10 discusses related work. §11 concludes. Proofs are given in Appendix A.