Static Program Analysis

Anders Møller and Michael I. Schwartzbach

September 9, 2020

Department of Computer Science
Aarhus University, Denmark

Contents

Preface

Static program analysis is the art of reasoning about the behavior of computer programs without actually running them. This is useful not only in optimizing compilers for producing efficient code but also for automatic error detection and other tools that can help programmers. A static program analyzer is a program that reasons about the behavior of other programs. For anyone interested in programming, what can be more fun than writing programs that analyze programs?

As known from Turing and Rice, all nontrivial properties of the behavior of programs written in common programming languages are mathematically undecidable. This means that automated reasoning of software generally must involve approximation. It is also well known that testing, i.e. concretely running programs and inspecting the output, may reveal errors but generally cannot show their absence. In contrast, static program analysis can – with the right kind of approximations – check all possible executions of the programs and provide guarantees about their properties. One of the key challenges when developing such analyses is how to ensure high precision and efficiency to be practically useful. For example, nobody will use an analysis designed for bug finding if it reports many false positives or if it is too slow to fit into real-world software development processes.

These notes present principles and applications of static analysis of programs. We cover basic type analysis, lattice theory, control flow graphs, dataflow analysis, fixed-point algorithms, widening and narrowing, path sensitivity, relational analysis, interprocedural analysis, context sensitivity, control-flow analysis, several flavors of pointer analysis, and key concepts of semantics-based abstract interpretation. A tiny imperative programming language with pointers and first-class functions is subjected to numerous different static analyses illustrating the techniques that are presented.

We take a *constraint-based approach* to static analysis where suitable constraint systems conceptually divide the analysis task into a front-end that generates constraints from program code and a back-end that solves the constraints to produce the analysis results. This approach enables separating the analysis