1 Introduction

1.1 Motivation

Program slicing [Wei84] is a program analysis and reverse engineering technique that reduces a program to those statements that are relevant for a particular computation. Informally, a slice provides the answer to the question "What program statements potentially affect the value of variable *v* at statement *s*?"

Program slicing was introduced by Mark Weiser because he made the observation that programmers have some abstractions about the program in mind during debugging. The process of debugging consists of following dependences from the erroneous statement *s* back to the influencing parts of the program. These statements may influence *s* either because they decide whether *s* is executed at all (*control dependence*) or because they define a variable that is used by *s* (*data dependence*). A program slicer can be used to automatically compute and visualize the slice of the program with regard to the statement *s* and the variables used or defined at *s*. It allows the programmer to focus his attention on the statements that are part of the slice and that might therefore contribute to the fault. Additionally, the programmer sees any statements that are not part of the slice although he knows that they should be.

Program slicing can be used to assist the programmer in a lot of tedious and error prone tasks, such as debugging, program integration, software maintenance, testing, and software quality assurance. Several variants of program slicing have been proposed for these purposes, including static slicing, dynamic slicing, backward slicing, forward slicing, chopping, interface slicing, etc.

A survey of existing program slicing tools shows that most of them are written for the programming language C, some for COBOL and FORTRAN. Most of these program slicers have problems with dynamic binding which is a cornerstone of object-oriented programming. Neither do they address the concepts of inheritance and polymorphism. Another problem is the performance of the program slicers. Since program slicing is an interactive method intended to assist the programmer, the results should ideally be presented immediately. It is unsatisfactory to wait for several minutes before the slice can be viewed.

1.2 Goals

The purpose of this work is to investigate whether static program slicing can be efficiently implemented for an object-oriented language such as Oberon-2. Although Oberon-2 is a small language, it is powerful and we may encounter many difficulties. We used the following goals as guidelines for the implementation of the Oberon Slicing Tool:

- The program slicer should be able to analyze the entire language which includes user-declared data types, structured types (records and arrays), global variables, functions with side-effects, nested procedures, type extension, dynamic binding via type-bound procedures (also called methods) and procedure variables (also called function pointers), recursion, and modules.
- The object-oriented features of Oberon-2 such as inheritance, dynamic binding and polymorphism should be fully supported. Object-oriented programs make heavy use of dynamic binding and pointers which are both difficult to handle by static analysis. Necessary conservative assumptions shall be restricted by feedback from the user.
- The internal data structures of the program slicer should closely model the semantics of the program.
- The computation of the slices should be fast, but the resulting slices should still be as precise as possible.
- The program slicer should support slicing of modular systems. Information that has already been computed for a module should be reused when slicing dependent modules.
- The program slicer should be an interactive tool. It should visualize all the information that has been computed during slicing and that could be useful for the programmer in order to understand the program.
- As the main fields of application of the program slicer we envisage assistance to the programmer, debugging, code understanding, maintenance, program testing and software metrics.

1.3 Outline

Chapter 2 gives an overview of the programming language Oberon-2, some background information about the computation of control flow and data flow as well as an overview of program slicing and a survey of the variants and applications of program slicing.

Chapter 3 describes current slicing algorithms together with their data structures, ranging from the original approach where slicing is seen as a data flow problem to the state-of-the-art where slicing is seen as a graph-reachability problem.

Chapter 4 describes the implementation of the Oberon Slicing Tool, its data structures and the algorithms used for the computation of control flow and data flow information as well as

for slicing itself.

Chapter 5 describes the user interface of the Oberon Slicing Tool, its visual elements and how user feedback is used to bridge the gap between static and dynamic slicing.

Chapter 6 compares the Oberon Slicing Tool with existing program slicers.

Chapter 7 gives a summary of the contributions of this thesis.

Chapter 8 outlines some areas for future work.