Acknowledgements

I want to thank my advisor Prof. H. Mössenböck for a liberal supervision of this project and for his ongoing encouragement and patience. The Oberon System was an excellent working tool and an appropriate base for the work presented in this thesis. Markus Hof and David Parsons proof-read earlier versions of this thesis and provided valuable comments and improvements. Last but not least, I wish to thank my colleagues at the department, my friends in Linz, my parents Grete and Alois, and my sisters Kathi and Ulli for their steady encouragement and help.

Contents

Abstract viii
Kurzfassung ix

1 Introduction 1
1.1 Motivation 1
1.2 Goals 2
1.3 Outline 2

2 Background Information 5
2.1 Oberon-2 5
2.2 Control Flow 7
2.2.1 Control Flow Graphs 7
2.2.2 Dominator and Post-dominator Trees 8
2.2.3 Control Dependences 10
2.3 Data Flow 12
2.3.1 Data Dependences 12
2.3.2 Computation of Used and Defined Variables 14
2.3.3 Computation of Reaching Definitions 16
2.4 Program Slicing 21
2.4.1 Variants of Program Slicing 22
2.4.2 Applications 26

3 Current Slicing Algorithms 31
3.1 Slicing as a Data Flow Problem 31
3.2 Slicing as a Graph-Reachability Problem 34
3.2.1 Program Dependence Graph 35
3.2.2 System Dependence Graph 35
3.2.3 Computation of Summary Edges 42
3.2.4 Enhancing Slicing Accuracy 45

4 Implementation 47
4.1 Overview 47
4.2 Algorithm 48
4.3 Data Structures 49
4.4 Computation of Control Flow Information 54
4.5 Computation of Data Flow Information 66
4.5.1 Computation of Used and Defined Variables 67
4.5.2 Computation of Reaching Definitions 78

4.6 Slicing 99
4.6.1 Intraprocedural Slicing 100
4.6.2 Interprocedural Slicing 101
4.6.3 Intermodular Slicing 102
4.7 Support of Object-Oriented Features 105
4.8 Modularization 106
4.8.1 Module Repository 107
4.8.2 Module Slicer 109

5 User Interface 113
5.1 Visual Elements 113
5.1.1 Bidirectional Links Between the Caller and the Callee 113
5.1.2 Data Dependences 115
5.1.3 Parameters 116
5.1.4 Aliases 119
5.1.5 Dynamic Types 120
5.2 User Feedback 124
5.3 Module SlicerFE 125
5.4 Model-View-Controller Concept 127

6 Comparison 129
6.1 Chopshop 129
6.2 Ghinsu 129
6.3 Spyder 130
6.4 Unravel 130
6.5 VALSOFT 131
6.6 Wisconsin Program-Slicing Project 131

7 Conclusions 133

8 Future Work 137
8.1 Integration into the Programming Environment 137
8.2 Other Variants of Slicing 138
8.3 Software Metrics 138

Appendix: Additional Module Definitions 141

Bibliography 151

Curriculum Vitae 157

Abstract

Program slicing is a program analysis technique that reduces programs to those statements that are relevant for a particular computation. A slice provides the answer to the question "What program statements potentially affect the value of variable *v* at statement *s*?" Mark Weiser introduced program slicing because he made the observation that programmers have some abstractions about the program in mind during debugging. When debugging a program one follows the 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 or because they define a variable that is used by *s*. Program slicing computes these dependences automatically and thus assists the programmer in a lot of error prone tasks, such as debugging, program integration, software maintenance, testing, and software quality assurance.

Object-oriented programming languages have attracted more and more attention during the last years since they allow one to write programs that are more flexible, reusable and maintainable. However, the concepts of inheritance, dynamic binding and polymorphism represent new challenges for static program analysis.

The result of this thesis is the Oberon Slicing Tool, a fully operational program slicing tool for the programming language Oberon-2. It integrates state-of-the-art algorithms and applies them to a strongly-typed object-oriented programming language. It extends them to support intermodular slicing of object-oriented programs. Control and data flow analysis considers inheritance, dynamic binding and polymorphism, as well as side-effects of functions, short-circuit evaluation of Boolean expressions and aliases due to reference parameters and pointers. The algorithm for alias analysis is fast but effective by taking into account information about the type of variables and the place of their declaration. The result of static program analysis is visualized with active text elements: hypertext links connect the call sites with the possible call destinations, parameter information elements indicate the direction of data flow at calls. Since static program analysis must make conservative assumptions about actual program executions, the sets of possible aliases and call destinations due to dynamic binding are more general then necessary. We visualize these sets and allow the programmer to restrict them via user interaction. These restrictions are then used to compute more precise control and data flow information. In this way, the programmer can limit the effects of aliases and dynamic binding and bring in his knowledge about the program into the analysis.

Kurzfassung

Program Slicing ist eine Programmanalysetechnik, die Programme auf jene Anweisungen reduziert, die für eine bestimmte Berechnung relevant sind. Ein Slice ist die Antwort auf die Frage: "Welche Anweisungen im Programm können den Wert der Variable *v* bei der Anweisung *s* beeinflussen?" Mark Weiser erfand Program Slicing, weil er beobachtete, dass sich Programmierer während der Fehlersuche Gedanken über die Beziehungen zwischen Programmteilen machen. Bei der Fehlersuche verfolgt man solche Beziehungen von der fehlerhaften Anweisung *s* zurück zu den Programmteilen, die sich auf *s* auswirken. Diese Anweisungen können *s* beeinflussen, indem sie entscheiden, ob *s* ausgeführt wird, oder indem sie einer Variablen einen Wert zuweisen, der von *s* verwendet wird. Program Slicing berechnet diese Beziehungen automatisch und unterstützt dadurch den Programmierer bei vielen fehleranfälligen Tätigkeiten, wie Fehlersuche, Integration von Programmversionen, Software-Wartung, Testen und Software-Qualitätssicherung.

Objektorientierte Programmiersprachen haben sich in den letzten Jahren immer mehr durchgesetzt, da sie es erlauben, Programme zu schreiben, die flexibler, besser wiederverwendbar und besser wartbar sind. Die Konzepte der Vererbung, der dynamischen Bindung und des Polymorphismus stellen allerdings für die Programmanalyse neue Herausforderungen dar.

Das Ergebnis dieser Doktorarbeit ist das Oberon Slicing Tool, ein voll funktionsfähiges Werkzeug für das Slicen von Oberon-2 Programmen. Es kombiniert Algorithmen, die dem Stand der Technik entsprechen, und wendet sie auf eine objektorientierte Programmiersprache mit strenger Typprüfung an. Es erweitert sie, um intermodulares Slicen von objektorientierten Programmen zu unterstützen. Die Kontroll- und Datenflussanalyse berücksichtigt Vererbung, dynamische Bindung und Polymorphismus, sowie Nebeneffekte von Funktionen, Kurzschlussauswertung von Booleschen Ausdrücken und Aliase aufgrund von Referenzparametern und Zeigern. Der Algorithmus für die Analyse von Aliasen ist schnell, aber trotzdem effektiv, indem er Information über den Typ von Variablen und den Ort ihrer Deklaration in Betracht zieht. Die Ergebnisse der statischen Programmanalyse werden durch aktive Textelemente dargestellt: Hypertext-Verknüpfungen verbinden Prozeduraufrufe mit den möglichen Aufrufzielen, Parameter-Informations-Elemente zeigen die Richtung des Datenflusses bei Aufrufen an. Da bei statischer Programmanalyse konservative Annahmen über die tatsächlichen Programmabläufe gemacht werden müssen, sind die Mengen der möglichen Aliase und Aufrufziele aufgrund von dynamischer Bindung allgemeiner, als sie sein müssten. Wir stellen diese Mengen dar und erlauben dem Programmierer, die Mengen durch Benutzerinteraktion einzuschränken. Die Restriktionen werden anschließend verwendet, um genauere Kontroll- und Datenflussinformation zu berechnen. Auf diese Weise kann der Programmierer die Auswirkungen von Aliasen und dynamischer Bindung einschränken und sein Wissen in die Analyse einbringen.