6 Comparison

In this chapter we describe several program slicing tools. Some of them are available freely. All of them are written for the programming language C. None of them can handle object-oriented programs. Most of them have difficulty with dynamic binding, aliases, structured data types, and side-effects of function calls. Most of them handle access to and definition of heap-allocated data extremely conservative.

In the literature, some graph representations have been proposed for slicing object-oriented programs ([Kri94] and [LaH96]). Chen et al. [ChWC97] addressed the problems of slicing object-oriented programs which make use of abstraction, encapsulation, inheritance, and polymorphism. Static slicing of Java programs has been described in [KoMG97]. Several people work on program slicing of object-oriented programs, but to the best of our knowledge, no program slicers for object-oriented programs are yet available.

6.1 Chopshop

Chopshop [Chop] is a reverse engineering tool to help programmers understand unfamiliar C code. Its data flow analysis technique is a modular generalization of static program slicing. The user can select several sources and sinks of information, and Chopshop shows how data flows from the sources to the sinks. Chopshop accepts full ANSI C and generates program slices in textual and pictorial form. The tool does alias analysis and handles all language features of ANSI C. It provides data flow abstractions for procedures and allows the user to zoom-in and zoom-out by applying different abstraction mechanisms to the pictures.

We could not evaluate the Chopshop ourselves since it is not freely available. The description is based on the technical papers and on the Web pages of Chopshop.

6.2 Ghinsu

Ghinsu is an environment [Ghin] that integrates a number of tools that aid the programmer in a number of software engineering activities, primarily maintenance. Ghinsu supports static and dynamic forward and backward slicing as well as dicing of programs written in a large subset of C (including pointers) or in Pascal. It is possible to list all uses of a definition and to list all definitions reaching a use. Dead code (i.e. code that is unreachable) can be highlighted.

We could not evaluate the Ghinsu environment ourselves since it is not freely available. The description in this thesis is based on the technical papers and on the Web pages of Ghinsu.

6.3 Spyder

Spyder [Spyd] is a debugging tool that has been prototyped by Hiralal Agrawal [Agr91] at Purdue University. It can compute interprocedural backward slices that are either static or dynamic. It implements slicing via graph reachability over dependence graphs. For dynamic slicing, instrumented code is compiled to an executable program that is executed while recording the input and output. This information is then used by the debugger to execute the program and generate a dynamic dependence graph that is finally traversed. The tool handles almost all of ANSI C, except functions with return values in general and function pointers in particular which is a big limitation. Arrays are treated as scalars, the indices are ignored.

An alpha release of Spyder is available for SunOS 4.0 and X11 Release 5 via the Web pages of Spyder [Spyd]. However, the description in this thesis is based on the technical papers and on the Web pages of Spyder.

6.4 Unravel

Unravel [Unra] is a prototype program slicing tool that can be used to statically evaluate ANSI C source code. It has been developed at the National Institute of Standards and Technology as part of a research project [Ly+95]. There are a few limitations that decrease the precision of the computed slices:

- Unions are not handled.
- Forks are not handled.
- Pointers to functions are not handled.
- Library functions are handled as follows: If a value is passed, it is assumed to be referenced, if an address is passed, then the referenced object is assumed to be changed. Nothing is assumed about global variables.

The tool is divided into three parts: a source code analysis component, a link component that links flow information from separate source files together and an interactive slicing component.

A prototype implementation is available for SunOS 4.1 and X11 Release 5 via the Web pages of Unravel. However, the description in this thesis is based on the technical papers and on the Web pages of Unravel.

6.5 VALSOFT

VALSOFT (Validating software controlled measuring systems by Program Slicing and Constraint Solving) [VALS] analyzes the data flow of a program and calculates exact conditions under which a suspect data flow can take place. The tool handles ANSI C programs. It can derive forward and backward slices, as well as chops. The nodes of the dependence graphs can consist of parts of statements in order to account for the side-effects of function calls. There are a few limitations that decrease the precision of the computed slices:

- Only flow-insensitive alias analysis is performed for pointers.
- Only structured uses of switch statements are supported.
- Gotos are not supported.
- Unions are handled in the same way as structs.

The system dependence graph can be written to a file for further use. The program slicer can also be used as a slicing server for clients such as a graph visualizer.

We could not evaluate VALSOFT ourselves since it is not freely available. The description in this thesis is based on the technical papers and on the Web pages of VALSOFT.

6.6 Wisconsin Program-Slicing Project

The Wisconsin Program-Slicing Tool [WIPS] supports operations on C programs, including backward slicing, forward slicing, and chopping. Furthermore, it can show control and/or flow dependences between program components and the set of global variables that might be modified by a procedure. There are a few limitations that decrease the precision of the computed slices:

- Functions that are not found in the program or library functions are handled as follows: Under the optimistic assumptions that these functions do not use and modify global variables and heap-allocated data, all possible summary edges are inserted between parameter nodes. If a library function uses or modifies global data and heap-allocated data, these effects are not handled properly.
- A call via a procedure variable is considered to be a possible call of every function whose address is taken somewhere in the program (i.e., it is assigned somewhere to a procedure variable).
- Every pointer dereference is considered to be a possible access of every piece of heap-allocated storage, as well as a possible access of every variable to which the address-of operator is applied somewhere in the program.
- Fields of a structured variable are not distinguished from the variable itself.
- Arrays are treated as a whole, access to array elements with constant indices are not handled specially.

The used algorithm can be outlined as follows: In a first pass, the call graph is built gathering information about global variables and dynamically bound calls. A second pass builds a control flow graph for each procedure. The program's procedure dependence graphs are then constructed from their corresponding control flow graphs. The system dependence graph is constructed by linking together all of the program's procedure dependence graphs with call-graph information. Additional information that summarizes transitive dependences due to procedure calls is finally added to the system dependence graph which can be written to a file for later use.

The Wisconsin Program-Slicing Project can be licensed from the University of Wisconsin under a distribution fee of US$500 for non-profit research purposes. We did not license it. The description in this thesis is based on the technical papers and on the Web pages. The Wisconsin Program-Slicing Project is being integrated into the tools of GrammaTech [Gram], a company working on innovative software development tools.