8 Future Work

Since many variants of program slicing have been proposed for different applications, our work could be extended into many different directions. In the following we will only briefly discuss how the Oberon Slicing Tool could be better integrated into the programming environment. Furthermore we discuss that other variants of program slicing could be implemented based on our graph representation and how the derived information could be used for software metrics.

8.1 Integration into the Programming Environment

At the time of writing this thesis, the Oberon Slicing Tool visualizes the slices in its own window. The source code is reconstructed from the abstract syntax tree and the symbol table. There are several advantages of this approach:

- The source code is presented in a canonical form. Each line contains at most one statement.
- Additional information can be inserted right away during the reconstruction of the source code.

However, it also has some disadvantages:

- The layout of the original source code is lost.
- The front-end of the compiler skips all comments, so they are lost and cannot be displayed.
- The front-end of the compiler performs some simple optimizations such as constant folding, transformation of IF statements with constant conditions, replacement of integer multiplication by a power of two by arithmetic shifts, etc. These optimizations cannot be undone, the results are presented to the user. This may give insights, but may also confuse.
- The reconstruction of the source code is difficult, the module implementing the reconstruction and the user interface is very big (approximately 3000 lines).

Another problem is that the slicing window cannot be used to edit the program since edit operations would probably invalidate the computed information. A simple approach would be to remove all visual elements at the first insert or delete operation performed on the text. A more sophisticated approach would be to partially invalidate and remove the visual elements and to recompute the information for the invalidated parts in the background. Currently slicing information is computed for one module at a time. It might be advantageous to keep the information more or less up-to-date during editing. This could be

done by performing control and data flow analysis on a per procedure basis. Additional conservative assumptions would be necessary, but while editing a procedure, intraprocedural information might be sufficient. During longer phases without edit operations, the analysis could be performed for the whole module. Therefore, the error handling and recovery capabilities of the Oberon compiler would have to be enhanced since the compiler would have to derive approximate syntax trees for incomplete or erroneous programs.

The Oberon Slicing Tool could also be integrated with the compiler which could benefit from the slicing information. Information about reaching definitions and aliases could be used to generate faster code.

The Oberon Slicing Tool could also be integrated with the Oberon interpreter [Obi98] and the debugger. Since the interpreter also uses the abstract syntax tree and the symbol table as its internal data structures, integration might be possible with reasonable effort. Interpretation of Boolean expressions might help to determine feasible paths. During debugging, information about the reaching definitions could be very useful. Additionally run-time information could be used to perform more precise data flow analysis. Gupta et al. [GuSH97] introduced *hybrid slicing*. They integrate dynamic information from a specific execution into a static analysis. They use breakpoint information and the dynamic call graph to more accurately estimate potential paths taken by the program. Their ideas could possibly be integrated into the Oberon Slicing Tool.

8.2 Other Variants of Slicing

The Oberon Slicing Tool can only compute static backward slices. This is not a big limitation for the envisaged fields of application. However, the graph representation of the program used for backward slicing has also been used by many researchers to implement other variants of slicing: Horwitz et al. [HoRB90] adapted the algorithms of backward slicing to forward slicing, Agrawal and Horgan [AgH90] presented algorithms for dynamic slicing based on dependence graphs.

8.3 Software Metrics

Since the Oberon Slicing Tool computes precise control flow and data flow information, it could be used to derive structural metrics based on

- the number of call destinations at dynamically bound calls
- the number of ordinary and additional parameters
- the number of aliases per definition
- the number of possible dynamic types per polymorphic variable
- the length of recursion chains due to static and dynamic binding
- the number of reaching definitions per usage

- the number of side-effects of procedures and functions

The resulting metrics could be tested against object-oriented and procedural programs, investigating the differences.