7 Conclusions

To use Oberon-2 as a source and target language helped us on the one hand (because we already had a parser, a compiler and other tools for that language), but made it difficult on the other hand (e.g., due to reference parameters, dynamic binding, and modules). The Oberon System was an excellent working tool and an appropriate base for the work presented in this thesis.

We have implemented the Oberon Slicing Tool, a fully operational program slicing tool for Oberon-2, with the following features:

- It computes static slices of Oberon-2 programs. There are no restrictions on the programs, i.e. they may use structured types (records and arrays), global variables of any type, objects on the heap; functions may have arbitrary side-effects; procedures may be nested; there may be recursion and dynamic binding due to type-bound procedures and procedure variables; and the program can consist of several modules.
- It uses a fine-grained program representation: the abstract syntax tree and the symbol table constructed by the front end of the Oberon-2 compiler, enriched with slicing information. The targets of the control and data dependences are the nodes of the abstract syntax tree. The entities that are considered for inclusion into the slice are nodes of the abstract syntax tree, rather than whole statements. This improves the precision of the slices significantly.
- Data flow information is computed precisely, taking into account side-effects of functions and short-circuit evaluation of Boolean expressions. Definition of array elements and record fields are handled as precisely as possible.
- It computes intraprocedural, interprocedural and intermodular slices. It uses a repository to store the computed slicing information which can be re-used later when importing already sliced modules. Thereby, precise knowledge about the parameter usage of imported functions is available.
- It handles procedural and object-oriented programs. The key concepts of object-oriented programming, such as inheritance, polymorphism and dynamic binding, as well as abstraction, information hiding, and encapsulation of data and code are handled in a natural way. Inheritance, polymorphism and dynamic binding are considered during control flow and data flow analysis, as well as during alias analysis.
- It restricts the sets of possible aliases at definitions by exploiting the fact that Oberon-2 is strongly typed and by exploiting knowledge about the place of the declaration of the possibly aliasing variables. Additionally the sets of possible aliases can be restricted by user feedback. The restricted sets of possible aliases are then used to compute more precise slicing information.

- It handles dynamic binding due to type-bound procedures and procedure variables by computing all call destinations for all call sites. The user may restrict the set of possible call destinations by user feedback. The restricted sets of possible call destinations are then used to compute more precise slicing information.
- We narrowed the gap between static and dynamic analysis by starting from conservative assumptions, having the user restrict the effects of aliases and dynamic binding and recompute more precise slicing information. The cycle of computing slicing information, slicing the program, and restricting the effects of aliases and dynamic binding may continue several times, in order to tailor the slices to the use case that the programmer has in mind. The user-feedback can be recorded and be played back later to customize the slice without user interaction.
- The computation of slicing information is very fast: Slicing information is computed within a few seconds, slices are computed without perceptible delays.
- The front end of the Oberon Slicing Tool uses active text elements to visualize the computed control flow and data flow information: bidirectional hypertext links connect the call sites and the called procedures, parameter information elements indicate how the parameters are used at call sites, the sets of possible aliases and possible call destinations are represented by popup elements that process user-feedback and forward it to the slicing algorithm. The Oberon Slicing Tool implements the Model-View-Controller concept, i.e. it is possible to have multiple views of the same slice that are kept consistent.

Among these features, the following constitute contributions to the state-of-the-art:

- Combination of state-of-the-art algorithms (slicing as a graph-reachability problem [HoRB90], computation of summary edges [LivC94, LivJ95]) for a strongly-typed programming language.
- Natural and efficient support for object-oriented features.
- Uniform handling of dynamic binding due to method calls and due to calls of procedure variables.
- Precise computation of control flow and data flow information based on the abstract syntax tree, taking into account side-effects of functions and short-circuit evaluation of Boolean expressions. Definition of array elements and record fields are handled as precisely as possible.
- Fast and efficient alias analysis taking into account type information and information about the place of the declaration.
- Intermodular slicing with a repository to store the computed slicing information for later reuse.
- Active text elements for the visualization of control and data flow information.

- User feedback via active text elements in order to restrict the sets of possible aliases at definitions and the sets of possible call destinations at polymorphic call sites, thus bridging the gap between static analysis and dynamic analysis.

The Oberon Slicing Tool is freely available under the conditions of the Oberon Slicing Tool License via the URL http://www.ssw.uni-linz.ac.at/Staff/CS/Slicing.html