Institut für Systemsoftware

O.Univ.-Prof. Dr. Hanspeter Mössenböck

JOHANNES KEPLER
UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

Automatic Code Generation Using Dynamic Programming Techniques

Diploma Thesis for Igor Böhm
Student ID: 0155477
Email: igor@bytelabs.org

Advances in compiler construction have shown that many steps can be automated during the process of compiler design and implementation. While there is a plethora of tools offering automation for the early phases of a compiler (e.g. scanner and parser generators), the later phases (e.g. syntax tree generation and manipulation, code generation) lack such variety of good tool support. The scope of this diploma thesis covers the design and implementation of a tool which produces a code generator for a target machine, given a specification of an abstract syntax tree (AST) of a program as well as a specification of the target machine itself. A dynamic programming approach as described in [1, 2, 3, 4, 5] should be utilized. The basic concept is as follows:

- Initially the addressing modes of operands the target machine has to offer must be determined. Examples of addressing modes are Constant, Register, Register relative, Indexed, Absolute, etc.
- Afterwards the operators of the AST must be mapped to target machine instructions. The following properties must be specified for each instruction:
  - the AST-Operator which is implemented by this instruction
  - the address modes of the operands as well as the result of the instruction
  - the costs of the instruction (e.g. memory in bytes or processor cycles used)
  - the instruction pattern to be emitted iff this instruction is selected
- Finally it must be specified how operands can be converted from one addressing mode into another one. Such operations also induce costs depending on the cost model (e.g. code size, processor cycles used, etc.).

During the first pass over the AST, dynamic programming is the key ingredient enabling us to tile the AST with instruction patterns in such a way that addressing modi fit together, and the overall costs are being minimized as well. Finally, the second pass over the AST emits the optimal instruction patterns selected earlier.

The following key points are essential ingredients of this diploma thesis:

- Design a specification language which enables one to specify all matching instructions including their addressing modes and costs for each operator of the AST. It should also be possible to specify addressing mode conversions and their costs. There should also be the possibility to annotate each Instruction with semantic actions, which are to be emitted when the instruction pattern is selected.
- Implement a generator which takes such a specification as its input, and produces a code generator as its output. The produced code generator should find the optimal tiling of instructions for the AST and emit the appropriate code for it.
- Using the designed specification language as well the generator, specify code generators for at least two target machines and create those code generators using your tool.
- Define a common format for syntax trees your generator supports. Write a front end for a simple language which emits such syntax trees and plug the front end together with your code generator back end.

Try to design the specification language with simplicity and easy readability in mind. It is essential to pay attention to good programming style as well as good documentation. After all it should be easy for other persons to maintain and improve your tool later on. The resulting tool will carry a simple and permissive open source license.

Regular meetings with the supervisor regarding the progress of the project are mandatory.

Supervisor: o.Univ.-Prof. Dr. Hanspeter Mössenböck
Diploma thesis commenced: January 2006

Literature

1. Fraser, Ch.W., Hanson D.R., Proebsting T.A.: Engineering a Simple and Efficient Code Generator Generator. Letters on Programming Languages and Systems , Vol 1(3), 213-226, 1992
2. Emmelmann, H., Schröer, F.-W., Landwehr, R.: BEG - A Generator for Efficient Back Ends. Proc. SIGPLAN'89 Conference on Programming Language Design and Implementation, SIGPLAN Notices 24, 7 (July 1989), 227-237
3. Horspool, N.R.: Automating the Selection of Code Templates. SOFTWARE - Practice and Experience, Vol 15(5), 503-514 (May 1985)
4. Proebsting, T.A.: Simple and Efficient BURS Table Generation. Proc. SIGPLAN'92 Conference on Programming Language Design and Implementation, SIGPLAN Notices (1992), 331-340
5. Proebsting, T.A.: BURS Automata Generation. ACM Transactions on Programming Languages and Systems (TOPLAS), Vol 17(3), 461-486 (May 1995)