

DAT2 Project Proposals

Language and Compilation, DAT2A, Spring 2010

Martin Toft

`mt@cs.aau.dk`

Department of Computer Science
Aalborg University

February 1, 2009

Two-Way Compilers

- Missing source code can be a huge problem
- Only assembly code or program binaries available
- Decompilation would be useful
- Generate higher-level code from lower-level code
- A two-way compiler
- The tasks:
 - 1 Define a simple, imperative language
 - 2 Develop a compiler for the language
 - 3 Develop a decompiler for the language

Implementing Functional Languages

- The functional programming paradigm has received increasing attention in recent years
- Not without reason:
 - Easier to reason about systematically
 - Easier to parallelize automatically
- Functions are treated like mathematical functions
- No side effects, e.g., if $f(n) = 2$ in one place, then $f(n)$ equals 2 everywhere
- Functions can be generated, used as input and returned
- The tasks:
 - 1 Design your own, simple, functional programming language or choose an existing language
 - 2 Implement a compiler, interpreter or hybrid solution for the language

Compiling and Refactoring an Imperative Language

- Learn foundations of compiler construction using a relatively low-level language
- Actually two proposals
- The shared, first part:
 - ① Design a language, e.g., base it on a subset of C or Pascal
 - ② Design and build a lexer
 - ③ Design and build a parser
 - ④ Design and build a semantic analyzer

Compiling and Refactoring an Imperative Language (cont.)

A Compiler for a Small Imperative Language

- 1 Design and build a code generator
- 2 Produce x86 assembly programs that can be assembled, linked with the system's C library, and run
- 3 Explore different code optimization techniques

A Refactoring Tool for a Small Imperative Language

- 1 Implement a refactoring technique or two
- 2 Involves necessary static analysis, making required changes, and transforming the intermediate representation back to the source code

Implicit Parallelism

- The future is multi-core!
- Writing threaded code is difficult and error prone
- The programmer does not want to learn a new language that allows parallelism
- He/she wants something that he/she is used to
- The tasks:
 - 1 Design the well-known language, *Lang1*
 - 2 Study which types of expressions in *Lang1* that can be executed concurrently
 - 3 Design *Lang2*, which is *Lang1* extended with a construction allowing parallelism
 - 4 Develop a compiler from *Lang1* to *Lang2*
 - 5 Develop an interpreter for *Lang2*

DAT2B: Concurrency and Operating Systems

- If “PSS projects” are allowed, feel free to talk to me about PSS project proposals
- Look in the semester archive for inspiration

Thank you for your attention

Questions?

These slides are available at

<http://martintoft.dk/slides/dat2proposals10.pdf>