Compilation

Translating high level languages to machine code

November 2, 2019

Welcome to compilation course

- Hi, my name is Oren
  - Email: here
  - GitHub user: here
  - Stack Overflow user: here
  - LinkedIn user: here
- Why I love compilers?
- Why should *you* study compilers?
  - A fascinating software product, get to know its internals
  - Become a better software programmer
  - Understand different programming paradigms
  - Learn new programming languages faster
- Teaching style
  - Feel very free to particiapte in class
  - You can't ask a bad question in compilation

What to expect from the course

- Build an entire compiler for an object oriented language
- Use industrial lexer and parser generators
- Sharpen your programming and debugging skills
- Improve your ability to work as a part of a team
- Familiarize yourself with *.nix systems
- Estimated time and effort for the course

Exercise	Points	Time (days)
1	10	1/4
2	15	1/2
3	25	2
4	25	3
Project	25	3

Resources

- There are several relevant text books
  - modern complier implementation in Java / Appel
  - compilers: principles, techniques and tools / Aho et al.
  - modern complier design / Grune
- Two open source modern compilers
  - Good old gcc
  - Its modern brother llvm/clang
- Propraietery compiler, but free and easy to use
  - Microsoft visual studio
- Online resources
  - Forums: stack overflow
  - RTFMs: JFlex, CUP, Graphviz, etc.
  - Mailing lists: cfe-dev, gcc-lists, gdb-lists

Overview

- Compilers have *front-ends* and *backends*
- *Front-end* handles the *source* language, performing
  - *Lexical Analysis*
    - Suppose you translate a book from Spanish to Hebrew
    - How would you start?
    - Scan the words one by one, and make sure they are all legal
    - See how Google Docs does it too here
  - *Syntax Analysis*
    - Continue thinking about the Spanish to Hebrew analogy
    - What to check next?
    - Each sentence must have valid structure: subject, verb etc.
    - Too hard for Google Docs to detect syntax errors here
  - *Semantic Analysis*
    - Still with the Spanish to Hebrew analogy
    - Check each sentence has valid meaning
  - *Intermediate representation* (IR)

Overview (cont)

- *Backend* handles the steps from IR to *destination* language
  - *Intermediate Representation* (IR)
    - Instead of direct src → dst translation, do src → IR → dst
    - IR contains everything needed for translation to machine code
    - IR is effective for handling *multiple src and dst languages*
    - Different compilers have different IRs
    - IR features ideally *independent of src language*
    - IR features ideally *independent of dst language*
    - Designing a good IR is more art than science
  - Example: `int i=1+2+3` → `int temp=1+2; int i=temp+3`
  - *Optimizations*
    - Example: `int i = 4+3` → `int i = 7`
  - *Static checks*
    - Example: `int i = 8/0` → compilation error
  - *Machine code generation*

Summary

- Simplified example int i=1+2+3
- Lexical analyis passed
  - Sentence contains the words: int,i,=,1,+,2,3
- Syntax analyis passed
  - Sentence contains the sturcture: type var = initvalue
- Semantic analyis passed
  - 1+2+3 is assigned to an integer-typed variable
- IR (should I write passed here too?)
  - int i=1+2+3 → int temp=1+2; int i=temp+3
- Machine Code (MIPS shown here)
  - li $t5,1 , li $t3,2 , add $t4,$t5,$t3 , li $t5,3 , add $t6,$t5,$t4