Compiler Construction 2016/2017
Intro

Peter Thiemann

October 21, 2016

Outline

Compilers

What is a compiler?

- a program that reads an executable program in one language and translates it into an executable program in another language
- we expect the program produced by the compiler to exhibit the same behavior as the original

What is an interpreter?

A program that reads an executable program and its input; produces the results of running that program.

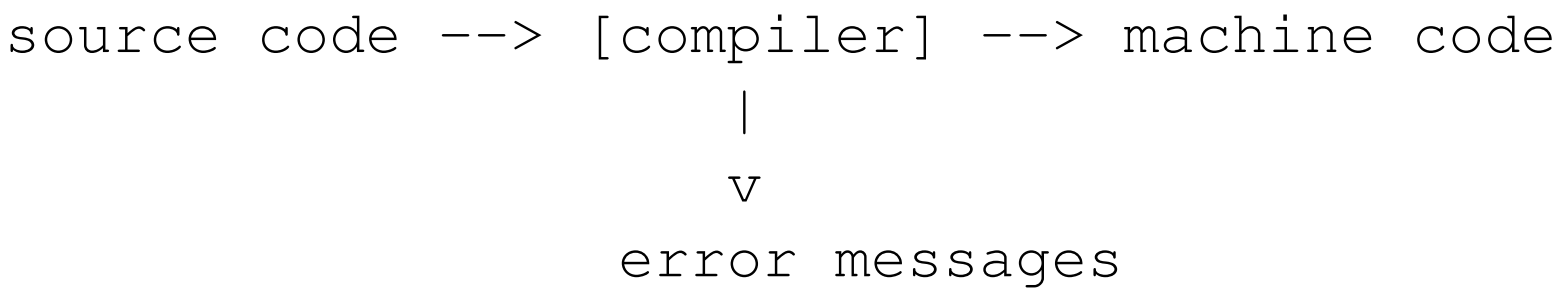
- This course deals mainly with compilers
- Many of the same issues arise in interpreters

Experience

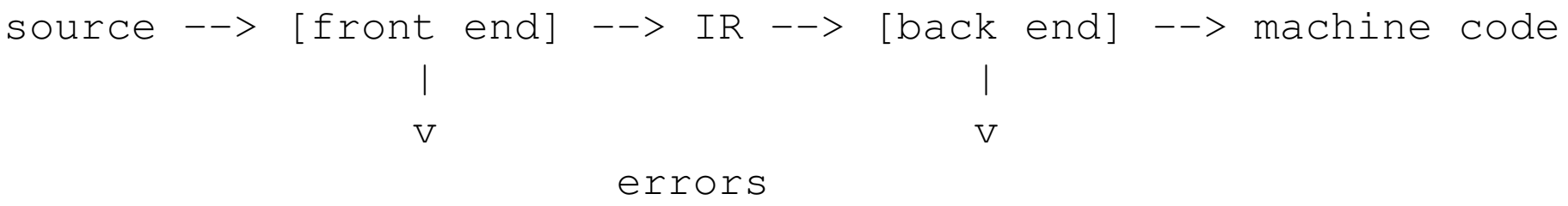
What qualities are important in a compiler?

1. Correct code
2. Output runs fast
3. Compiler runs fast
4. Compile time proportional to program size
5. Support for separate compilation
6. Good diagnostics for syntax errors
7. Works well with the debugger
8. Good diagnostics for flow anomalies
9. Cross language calls
10. Consistent, predictable optimization

Abstract View



Traditional Two-Pass Compiler



Implications

- intermediate representation (IR)
- front end maps legal code into IR
- back end maps IR onto target machine
- simplify retargeting
- allows multiple front ends
- multiple passes $\Rightarrow$ better code

Front End

```
source --> [scanner] --> tokens --> [parser] --> IR
```

Responsibilities

- recognize legal procedure
- report errors
- produce IR
- preliminary storage map
- shape the code for the back end

Scanner

```
source --> [scanner] --> tokens
```

Scanner:

- partitions input into lexemes — the basic unit of syntax
- maps lexemes into tokens
- `x = x + 1;` becomes

  `<id, x> <sym,=> <id, x> <sym,+> <num, 1> <sym,;>`
- typical tokens: number, id, +, -, *, /, do, end
- eliminates white space (tabs, blanks, comments)
- a key issue is speed

Front end/Parser

```
tokens --> [parser] --> IR
```

Parser

- recognize context-free syntax
- guide context-sensitive analysis
- construct IR(s)
- produce meaningful error messages
- attempt error correction

Parser generators

Front End/Context-Free Syntax

Context-free syntax is specified with a context-free grammar, often in Backus-Naur form (BNF).

```
<sheep noise> ::= baa
                | baa <sheep noise>
```

The noises sheep make under normal circumstances

- `<sheep noise>` variable, nonterminal symbol
- `::=` and `|` metasymbols
- everything else: terminal symbols that appear in the input
- convention: first variable is the goal or start variable

Syntax/Expressions

Context free syntax can be put to better use

```
1 <goal> ::= <expr>
2 <expr> ::= <expr> <op> <term>
3          | <term>
4 <term> ::= number
5          | id
6 <op> ::= +
7        | -
```

Simple expressions with addition and subtraction over tokens id and number

Derive an expression

Starting from the goal variable, repeatedly replace a variable by its right-hand side until no variables are left.
Ex: `x + 2 - y`

Front End/IR

The result of parsing can be represented by a derivation tree.

Front End/AST

A derivation tree contains information that is useless for compiling. Hence, use abstract syntax trees (AST) as IR.

Back End

```
IR --> [instruction selection]
   --> [register allocation] --> machine code
```

Responsibilities

- translate IR into target machine code
- choose instructions for each IR operation
- decide what to keep in registers at each point
- ensure conformance with system interfaces

Back End/Instruction Selection

```
IR --> [instruction selection] --> IR'
```

Instruction selection

- produce compact, fast code
- use available addressing modes
- pattern matching problem
  - ad hoc techniques
  - tree pattern matching
  - string pattern matching
  - dynamic programming

Back End/Register Allocation

```
IR' --> [register allocation] --> machine code
```

Register Allocation

- have value in a register when used
- limited resources
- changes instruction choices
- can move loads and stores
- optimal allocation is difficult

Further Passes

```
IR --> [transform] --> IR
```

Code Improvement

- analyzes and changes IR
- goal is to reduce runtime, space usage, energy usage, ...
- must preserve values
- sometimes several passes, in certain order, run repeatedly