Compilers

Nai-Wei Lin
Department of Computer Science and
Information Engineering
National Chung Cheng University
Objectives

♠ Introduce principles and techniques for compiler construction
♠ Introduce principles and techniques for compiler-compiler construction
♠ Use a compiler-compiler to exercise the compiler construction for a small language
Introduction
Programming Languages

♠ Human use *nature languages* to communicate with each other
  – *Chinese, English, French*

♠ Human use *programming languages* to communicate with computers
  – *Fortran, Pascal, C*
Computer Organization

- Applications
- Compiler
- Operating System
- Hardware Machine
Compiler

Source Code

Compiler

Target Code

Error Messages
Components of a Compiler

♦ Analysis
  – Lexical Analysis
  – Syntax Analysis
  – Semantic Analysis

♦ Synthesis
  – Intermediate Code Generation
  – Code Optimization
  – Code Generation
Lexical Analysis

Someone breaks the ice

final := initial + rate * 60

Someone breaks the ice

id₁ := id₂ + id₃ * 60
Syntax Analysis

Someone breaks the ice

\[ \text{sentence} \]
\[ \downarrow \]
\[ \text{subject} \quad \text{verb} \quad \text{object} \]
\[ \text{Someone breaks the ice} \]

\[ id_1 := id_2 + id_3 * 60 \]

\[ \text{id}_1 \]
\[ \downarrow \]
\[ := + \]
\[ \text{id}_2 \quad \text{id}_3 \]
\[ * 60 \]
Semantic Analysis

Someone plays the piano  
(meaningful)

The piano plays someone  
(meaningless)
Intermediate Code Generation

Someone breaks the ice

有人打破冰

\[ \text{id}_1 := \text{id}_2 + \text{id}_3 \ast \text{i2r} (60) \]

\[ \text{temp1} := \text{i2r} (60) \]
\[ \text{temp2} := \text{id}_3 \ast \text{temp1} \]
\[ \text{temp3} := \text{id}_2 + \text{temp2} \]
\[ \text{id}_1 := \text{temp3} \]
Code Optimization

有人打破冰

```plaintext
temp1 := i2r ( 60 )
temp2 := id3 * temp1
temp3 := id2 + temp2
id1 := temp3
```

有人打破沉默

```plaintext
temp1 := id3 * 60.0
id1 := id2 + temp1
```
Code Generation

有人打破沉默

temp1 := id₃ * 60.0
id₁ := id₂ + temp1

movf id₃, r2
mulf #60.0, r2
movf id₂, r1
addf r2, r1
movf r1, id₁
Structure of a Compiler

♠ Front End
  – *Lexical Analysis*
  – *Syntax Analysis*
  – *Semantic Analysis*
  – *Intermediate Code Generation*

♠ Back End
  – *Code Optimization*
  – *Code Generation*
Reuse of Components

Fortran  Pascal  C

 Intermediate Code

MIPS  SPARC  Pentium
Applications

♠ Interpreters
♠ WWW Browsers
♠ Word Processors
♠ Computer-Aided Design
♠ Computer-Aided Manufacture
Compiler-Compiler

Compiler

Compiler Description

Error Messages

Compiler-compiler

Compiler
Formal Language Theory

- **language**
  - define grammar
  - accept automaton

- grammar
  - generate automaton
    - automatically
Grammars

♠ The sentences in a language may be defined by a set of rules called a grammar

\[ L: \{00, 01, 10, 11\} \]

\[ G: \text{the set of binary strings of length 2} \]
Automata

◆ An *acceptor* of a language is an automaton which determines if an input string is a sentence in the language

◆ A *transducer* of a language is an automaton which determines if an input string is a sentence in the language and may produce strings as output if it is in the language
Transducers

language $L_1$

define grammar $G_1$

automaton

accept

transduce

generate

language $L_2$

define grammar $G_2$
Metalanguage

♦ **Metalanguage**: a language used to define another language
Automatic Compiler Generation

♠ We will use different metalanguages to define the various components of a programming language so that these components can be generated automatically.
Definition of Programming Languages

♦ *Lexical syntax*: regular expressions
♦ *Syntax*: context free grammars
♦ *Semantics*: attribute grammars
♦ *Intermediate code generation*: attribute grammars
♦ *Code generation*: tree grammars
Languages

♦ Alphabet - any finite set of symbols
  \{0, 1\}: binary alphabet

♦ String - a finite sequence of symbols from the alphabet
  1011: a string of length 4
  \varepsilon: the empty string

♦ Language - any set of strings on the alphabet
  \{00, 01, 10, 11\}: the set of strings of length 2
  \emptyset: the empty set
Terms for Parts of a String

♠ **string**: banana

♠ (proper) **prefix**: ε, b, ba, ban, ..., banana

♠ (proper) **suffix**: ε, a, na, ana, ..., banana

♠ (proper) **substring**: ε, b, a, n, ba, an, na, ..., banana

♠ **subsequence**: ε, b, a, n, ba, bn, an, aa, na, nn, ..., banana

♠ **sentence**: a string in the language
Operations on Strings

♦ *concatenation*:
  
  \[ x = \text{dog} \]
  \[ y = \text{house} \]
  \[ xy = \text{doghouse} \]

♦ *exponentiation*:
  
  \[ s^0 = \varepsilon \]
  \[ s^1 = s \]
  \[ s^2 = ss \]
Operations on Languages

♠ **Union of** L and M, L ∪ M
   \[ L \cup M = \{ s \mid s \in L \text{ or } s \in M \} \]

♠ **Concatenation of** L and M, LM
   \[ LM = \{ st \mid s \in L \text{ and } t \in M \} \]

♠ **Kleene closure of** L, L*
   \[ L^* = \bigcup_{i=0}^{\infty} L^i \]

♠ **Positive closure of** L, L+
   \[ L^+ = \bigcup_{i=1}^{\infty} L^i \]
Implementation of Programming Languages

♦ *Regular expressions*:  
  finite automata, lexical analyzer

♦ *Context free grammars*:  
  pushdown automata, parser

♦ *Attribute grammars*:  
  attribute evaluators, type checker and intermediate code generator

♦ *Tree grammars*:  
  finite tree automata, code generator
Automatic Program Generators

- Problem
  - Define
  - Solve
- Specification Language
  - Generate automatically
- Program
Content

- Lexical analysis
- Syntax analysis
- Semantic analysis
- Intermediate code generation
- Code generation
共勉

子曰：
「學而時習之，不亦說乎？」

-- 論語