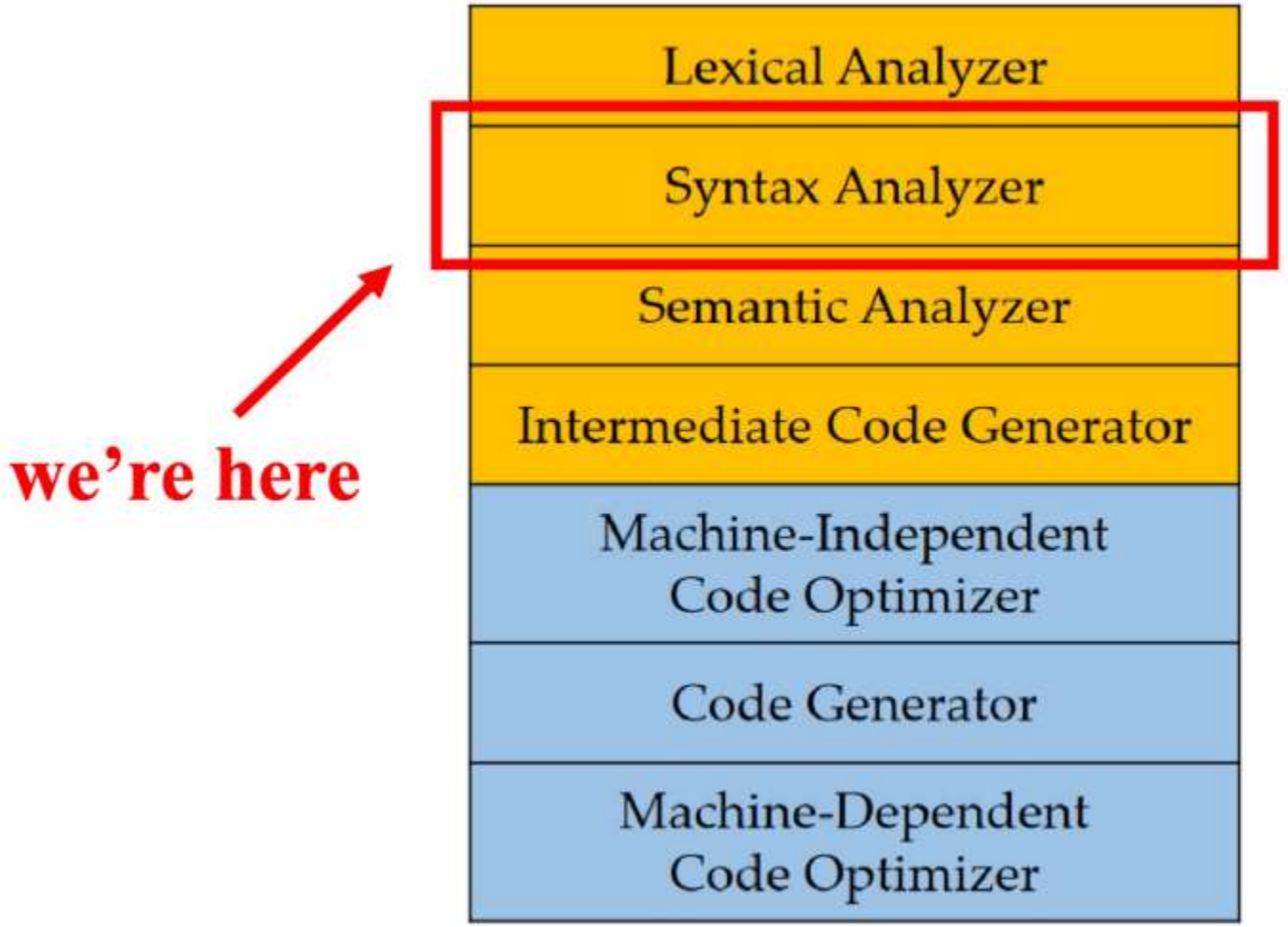
北京郵電大學
BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

Compiler - Lab 3

Outline

- Context-Free Grammar Introduction
- GNU Bison Introduction
- Interacting Bison with Flex
- Bison Exercise

Syntax Analysis



Context-Free Grammar Introduction

- Context-Free Grammar

  Context-free grammar (or CFG) defines the context-free languages, a strict superset of the regular language

Context-Free Grammar Introduction

- A context-free grammar (CFG) consists of four parts:
  - **Terminals (终结符号)**: Basic symbols from which strings are formed (token names)
  - **Nonterminals (非终结符号)**: Syntactic variables that denote sets of strings
    - Usually correspond to a language construct, such as stmt (statements)
  - One nonterminal is distinguished as the **start symbol (开始符号)**
    - The set of strings denoted by the start symbol is the language generated by the CFG
  - **Productions (产生式)**: Specify the manner in which the terminals and non-terminals can be combined to form strings
  - Format: head (left side) → body (right side)
  - The head is a nonterminal; the body consists of zero or more terminals/nonterminals

Context-Free Grammar Introduction

- Example – Arithmetic Expression

```
Exp → int | Exp Op Exp | (Exp)
 Op → + | - | * | /
```

Red for non-terminals
Blue for terminals

```
Exp ⇒ Exp Op Exp
    ⇒ Exp Op (Exp)
    ⇒ Exp Op (Exp Op Exp)
    ⇒ Exp * (Exp Op Exp)
    ⇒ int * (Exp Op Exp)
    ⇒ int * (int Op Exp)
    ⇒ int * (int Op int)
    ⇒ int * (int + int)
```

Context-Free Grammar Introduction

- Example – Chemistry

```
Form → Cmp | Cmp Ion
Cmp → Term | Term Num | Cmp Cmp
Term → Elem | (Cmp)
Elem → H|He|Li|Be|B|C|...
Ion → + | - | IonNum+ | IonNum-
IonNum → 2|3|4|...
Num → 1 | IonNum
```

Form ⇒ Cmp Ion
⇒ Cmp Cmp Ion
⇒ Cmp Term Num Ion
⇒ Term Term Num Ion
⇒ Elem Term Num Ion
⇒ Mn Term Num Ion
⇒ Mn Elem Num Ion
⇒ MnO Num Ion
⇒ MnO IonNum Ion
⇒ MnO_4 Ion
⇒ MnO_4^-

Context-Free Grammar Introduction

- Example – Chemistry

```
Form → Cmp | Cmp Ion
Cmp → Term | Term Num | Cmp Cmp
Term → Elem | (Cmp)
Elem → H|He|Li|Be|B|C|...
Ion → + | - | IonNum+ | IonNum-
IonNum → 2|3|4|...
Num → 1 | IonNum
```

Derived forms:

MnO_4^-

$C_{19}H_{14}O_5S$

$Cu_3(CO_3)_2(OH)_2$

S^{2-}

$H_2Na(CO_4)^{3+}$ ← valid syntax, wrong semantics

...

GNU Bison Introduction

- Bison的前身为基于Unix的Yacc。令人惊讶的是， Yacc的发布时间甚至比Lex还要早。 Yacc所采用的LR分析技术的理论基础早在20世纪50年代就已经由Knuth逐步建立了起来，而Yacc本身则是贝尔实验室的S.C. Johnson基于这些理论在1975年到1978年写成的。到了1985年，当时在UC Berkeley的一个研究生Bob Corbett在BSD下重写了Yacc ，后来GNU Project接管了这个项目，为其增加了许多新的特性，于是就有了我们今天所用的GNU Bison。

Interacting Bison with Flex

- Calculator Example

```
root@8d8d6a7d0f25:/mnt/Workspace/calc# make calc
flex lex.l
bison -t -d syntax.y
gcc syntax.tab.c -lfl -ly -D CALC_MAIN -o calc.out
root@8d8d6a7d0f25:/mnt/Workspace/calc# echo "1+1=" | ./calc.out
 = 2
root@8d8d6a7d0f25:/mnt/Workspace/calc# echo "2*3=" | ./calc.out
 = 6
root@8d8d6a7d0f25:/mnt/Workspace/calc# echo "5-2*2=" | ./calc.out
 = 1
root@8d8d6a7d0f25:/mnt/Workspace/calc#
```

Bison Exercise

- Valid Parentheses

  Leetcode 20, solve it with Flex/Bison in the lab

```
root@8d8d6a7d0f25:/mnt/Workspace/parentheses# make libparen
flex lex.l
bison -t -d syntax.y
gcc syntax.tab.c -lfl -ly -fPIC --shared -o libparen.so
root@8d8d6a7d0f25:/mnt/Workspace/parentheses# python3 paren_test.py
All tests passed!
root@8d8d6a7d0f25:/mnt/Workspace/parentheses#
```