南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

CS323 Lab 5

Yepang Liu

liuyp1@sustech.edu.cn

Outline

- Grammar design
- Bison exercise

Grammar Design

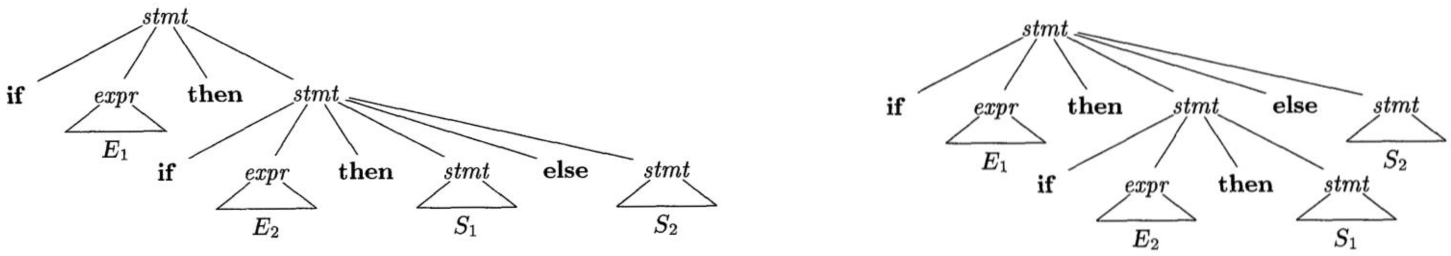
- CFGs are capable of describing most, but not all, of the syntax of programming languages
  - “Identifiers should be declared before use” cannot be described by a CFG
  - Subsequent phases must analyze the output of the parser to ensure compliance with such rules

- Before parsing, we typically apply several transformations to a grammar to make it more suitable for parsing
  - Eliminating ambiguity (消除二义性)
  - Eliminating left recursion (消除左递归)
  - Left factoring (提取左公因子)

Eliminating Ambiguity (1)

$$\begin{array}{lcl} stmt & \to & \textbf{if } expr \textbf{ then } stmt \\ & | & \textbf{if } expr \textbf{ then } stmt \textbf{ else } stmt \\ & | & \textbf{other} \end{array}$$

Two parse trees for **if** E_1 **then if** E_2 **then** S_1 **else** S_2

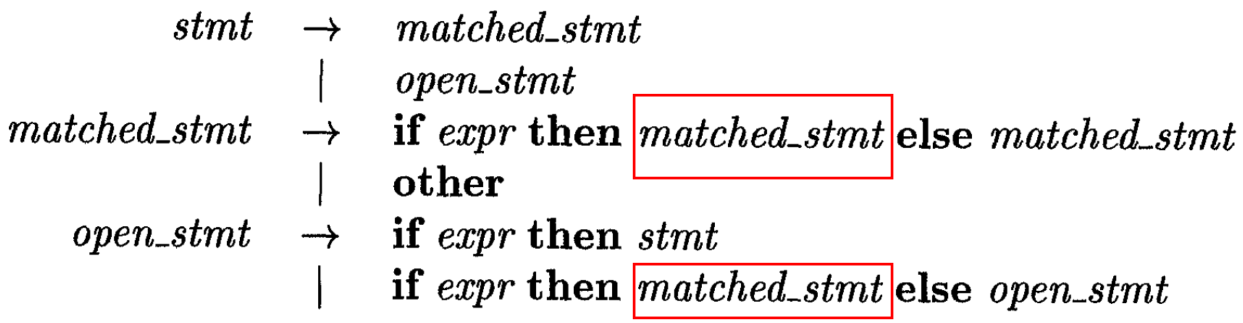


Which parse tree is preferred in programming?

(i.e., else matches which then?)

Eliminating Ambiguity (2)

- **Principle of proximity:** match each `else` with the closest unmatched `then`
  - **Idea of rewriting:** A statement appearing between a `then` and an `else` must be matched (must not end with an unmatched `then`)



$$\begin{array}{rcl} \mathit{stmt} & \to & \mathit{matched\_stmt} \\ & | & \mathit{open\_stmt} \\ \mathit{matched\_stmt} & \to & \textbf{if } \mathit{expr} \textbf{ then } \boxed{\mathit{matched\_stmt}} \textbf{ else } \mathit{matched\_stmt} \\ & | & \textbf{other} \\ \mathit{open\_stmt} & \to & \textbf{if } \mathit{expr} \textbf{ then } \mathit{stmt} \\ & | & \textbf{if } \mathit{expr} \textbf{ then } \boxed{\mathit{matched\_stmt}} \textbf{ else } \mathit{open\_stmt} \end{array}$$

Rewriting grammars to eliminate ambiguity is difficult.
There are no general rules to guide the process.

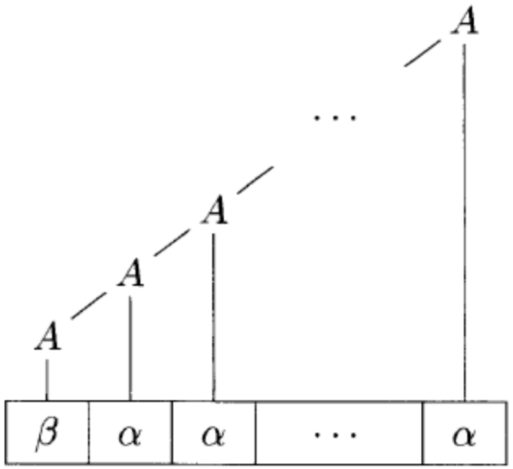
* open_stmt means the last then may not have matching else

Eliminating Left Recursion

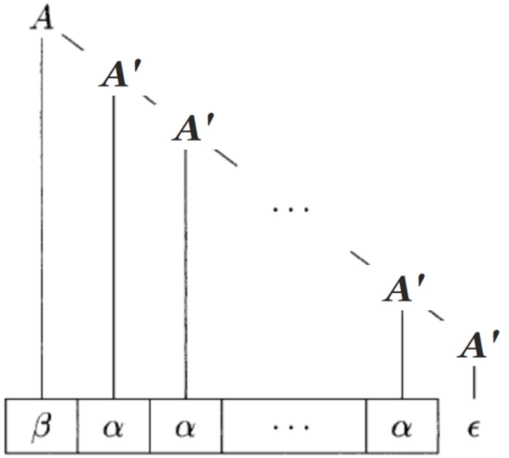
- A grammar is **left recursive** if it has a nonterminal A such that there is a derivation $A \overset{+}{\Rightarrow} A\alpha$ for some string α
  - $S \to Aa \mid b$
  - $A \to Ac \mid Sd \mid \epsilon$
  - Because $S \Rightarrow Aa \Rightarrow Sda$
- **Immediate left recursion (立即左递归):** the grammar has a production of the form $A \to A\alpha$
- Top-down parsing methods cannot handle left-recursive grammars (bottom-up parsing methods can handle…)

Eliminating Immediate Left Recursion

- Simple grammar: $A \rightarrow A\alpha \mid \beta$
  - It generates sentences starting with the symbol β followed by zero or more α's



- Replace the grammar by:
  - $A \rightarrow \beta A'$
  - $A' \rightarrow \alpha A' \mid \epsilon$
  - It is right recursive now



Eliminating Immediate Left Recursion

- The general case: $A \rightarrow A\alpha_1 \mid \ldots \mid A\alpha_m \mid \beta_1 \mid \ldots \mid \beta_n$

- Replace the grammar by:
  - $A \rightarrow \beta_1 A' \mid \ldots \mid \beta_n A'$
  - $A' \rightarrow \alpha_1 A' \mid \ldots \mid \alpha_m A' \mid \epsilon$

Example

$$E \longrightarrow E + T \mid T$$

$$T \longrightarrow T * F \mid F$$

$$F \longrightarrow (E) \mid \mathbf{id}$$

$$\Longrightarrow$$

$$E \longrightarrow T E'$$

$$E' \longrightarrow + \; T E' \mid \epsilon$$

$$T \longrightarrow F T'$$

$$T' \longrightarrow * \; F T' \mid \epsilon$$

$$F \longrightarrow (\; E \;) \mid \mathbf{id}$$

Left Factoring (提取左公因子)

- If we have the following two productions

$$\begin{aligned} \mathit{stmt} &\rightarrow \textbf{if}\ \mathit{expr}\ \textbf{then}\ \mathit{stmt}\ \textbf{else}\ \mathit{stmt} \\ &\mid\ \textbf{if}\ \mathit{expr}\ \textbf{then}\ \mathit{stmt} \end{aligned}$$

- On seeing input **if**, we cannot immediately decide which production to choose
- In general, if $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2$ are two productions, and the input begins with a nonempty string derived from α. We may defer choosing productions by expanding A to $\alpha A'$ first

$$\boxed{\begin{aligned} A &\rightarrow \alpha A' \\ A' &\rightarrow \beta_1 \mid \beta_2 \end{aligned}}$$

Outline

- Grammar design
- Bison exercise

Validate IP Address (leetcode #468)

- Use Bison and Flex to complete the following task:
  - Given a string *queryIP*, output "IPv4" if *queryIP* is a valid IPv4 address, "IPv6" if *queryIP* is a valid IPv6 address or "Invalid" otherwise

- A valid IPv4 address is an IP in the form of "$x_1.x_2.x_3.x_4$":
  - Each x_i is a decimal integer in the range [0, 255]
  - x_i cannot contain leading zeros
  - Examples: 192.168.0.1 (valid), 192.168.01.1 (invalid), 192.168@1.1 (invalid)

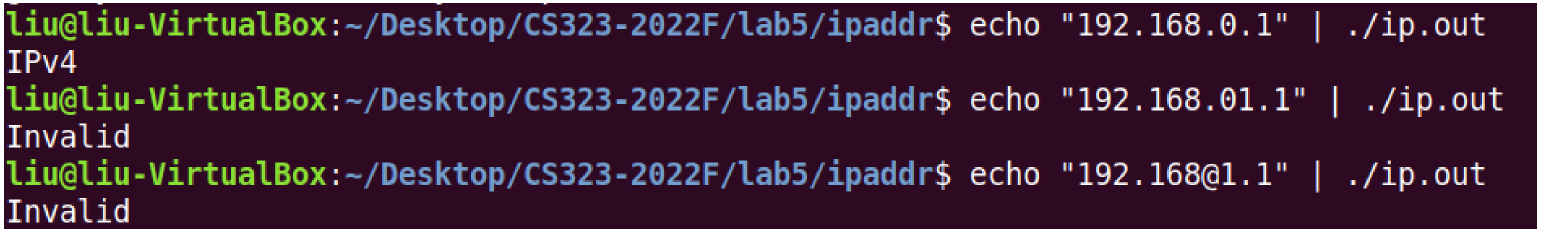
Validate IP Address (leetcode #468)

- A valid IPv6 address is an IP in the form "x_1:x_2:x_3:x_4:x_5:x_6:x_7:x_8":
  - The length of each x_i is in the range [1, 4]
  - x_i is a hexadecimal string which may contain digits, lowercase English letter ('a' to 'f') and upper-case English letters ('A' to 'F')
  - Valid examples:
    - 2001:0db8:85a3:0000:0000:8a2e:0370:7334
    - 2001:db8:85a3:0:0:8A2E:0370:7334
  - Invalid examples:
    - 2001:0db8:85a3::8A2E:037j:7334
    - 02001:0db8:85a3:0000:0000:8a2e:0370:7334

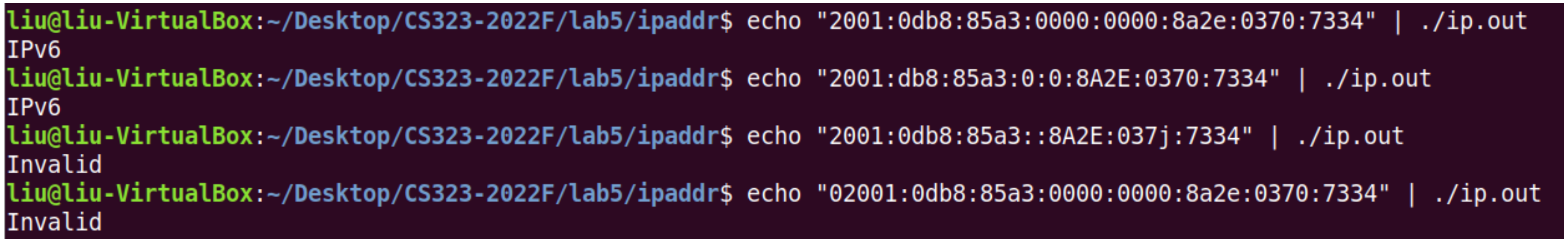
More instructions

- Clone the `lab5/ipaddr` directory
- The `lex.l` file is provided to recognize x strings (but does not check its validity), the dot and colon in IP addresses. Please use it as is.
- Complete the `syntax.y` file and providing production rules, semantic actions, as well as necessary supporting functions
- 100 points (finish during lab), 80 points (finish before Friday)

Test Inputs and Sample Outputs



```
liu@liu-VirtualBox:~/Desktop/CS323-2022F/lab5/ipaddr$ echo "192.168.0.1" | ./ip.out
IPv4
liu@liu-VirtualBox:~/Desktop/CS323-2022F/lab5/ipaddr$ echo "192.168.01.1" | ./ip.out
Invalid
liu@liu-VirtualBox:~/Desktop/CS323-2022F/lab5/ipaddr$ echo "192.168@1.1" | ./ip.out
Invalid
```



```
liu@liu-VirtualBox:~/Desktop/CS323-2022F/lab5/ipaddr$ echo "2001:0db8:85a3:0000:0000:8a2e:0370:7334" | ./ip.out
IPv6
liu@liu-VirtualBox:~/Desktop/CS323-2022F/lab5/ipaddr$ echo "2001:db8:85a3:0:0:8A2E:0370:7334" | ./ip.out
IPv6
liu@liu-VirtualBox:~/Desktop/CS323-2022F/lab5/ipaddr$ echo "2001:0db8:85a3::8A2E:037j:7334" | ./ip.out
Invalid
liu@liu-VirtualBox:~/Desktop/CS323-2022F/lab5/ipaddr$ echo "02001:0db8:85a3:0000:0000:8a2e:0370:7334" | ./ip.out
Invalid
```