Lecture 5

Context-Free Languages

COSE215: Theory of Computation

Seunghoon Woo

Fall 2023

Context-Free Languages

- **Practice:** $L = \{w \in \{a, b\}^* \mid n_a(w) = n_b(w)\}$
  - One possible production rule
    - $S \rightarrow aSb \mid bSa \mid SS \mid \lambda$
  - Question (1)
    - $S \rightarrow aA \mid Aa \mid \lambda$
    - $A \rightarrow bS \mid Sb$
  - Question (2)
    - $S \rightarrow abS \mid Sab \mid aSb \mid baS \mid Sba \mid bSa \mid \lambda$

Context-Free Languages

- **Practice:** $L = \{w \in \{a, b\}^* \mid n_a(w) = n_b(w)\}$
  - One possible production rule
    - $S \rightarrow aSb \mid bSa \mid SS \mid \lambda$
    - $SS \rightarrow aSbS \mid bSaS \mid SaSb \mid SbSa$ . .
  - Question (1)
    - $S \rightarrow aA \mid Aa \mid \lambda$
    - $A \rightarrow bS \mid Sb$
    - baab ☹
  - Question (2)
    - $S \rightarrow abS \mid Sab \mid aSb \mid baS \mid Sba \mid bSa \mid \lambda$
    - I think this answer is correct ☺

Midterm exam!

- **Date:** Oct. 24th (Tuesday), 16:30 – 17:45 (75 minutes)
- **Location:** 301 and 302, Aegineung (애기능생활관)
  - Please check your exam room on Blackboard
- **Coverage:** **Lecture 1 – Lecture 6_1 (CFG simplification)**
- **Format:** Closed book, closed notes, no programming questions
- **Failure to attend exam without permission => F**
- **Don't be late!**
  - You cannot enter the exam rooms after 17:00

Contents

- **Parsing and ambiguity**
- **Context-free grammars and programming languages**

Parsing and Ambiguity

- **We have focused on detecting L from a given G**
- **Membership algorithm**
  - Given a string w of terminals, we want to know whether or not w is in $L(G)$
- **Parsing**
  - If w is in $L(G)$, we then find a derivation of w
  - A sequence of productions by which a $w \in L(G)$ is derived

Parsing and Ambiguity

- **Example**
  - Consider the grammar $S \rightarrow SS \mid aSb \mid bSa \mid \lambda$
    - If the string $aabb$ is in $L(G)$?
    - If so, how the string can be derived?

Parsing and Ambiguity

- **Example**
  - Consider the grammar $S \rightarrow SS \mid aSb \mid bSa \mid \lambda$
    - If the string $aabb$ is in $L(G)$?
    - If so, how the string can be derived?

$S \Rightarrow SS$

$S \Rightarrow aSb$

$S \Rightarrow bSa$

$S \Rightarrow \lambda$

Parsing and Ambiguity

- **Example**
  - Consider the grammar $S \rightarrow SS \mid aSb \mid bSa \mid \lambda$
    - ❖ If the string $aabb$ is in $L(G)$?
    - ❖ If so, how the string can be derived?

$S \Rightarrow SS$

$S \Rightarrow aSb$

$S \Rightarrow \mathbf{S}S \Rightarrow SSS$
$S \Rightarrow \mathbf{S}S \Rightarrow aSbS$
$S \Rightarrow \mathbf{S}S \Rightarrow bSaS$
$S \Rightarrow \mathbf{S}S \Rightarrow S$

$S \Rightarrow a\mathbf{S}b \Rightarrow aSSb$
$S \Rightarrow a\mathbf{S}b \Rightarrow aaSbb$
$S \Rightarrow a\mathbf{S}b \Rightarrow abSab$
$S \Rightarrow a\mathbf{S}b \Rightarrow ab$

Parsing and Ambiguity

- **Example**
  - Consider the grammar $S \rightarrow SS \mid aSb \mid bSa \mid \lambda$
    - ❖ If the string $aabb$ is in $L(G)$?
    - ❖ If so, how the string can be derived?

$S \Rightarrow SS$

$S \Rightarrow \mathbf{S}S \Rightarrow SSS$
$S \Rightarrow \mathbf{S}S \Rightarrow aSbS$
$S \Rightarrow \mathbf{S}S \Rightarrow bSaS$
$S \Rightarrow \mathbf{S}S \Rightarrow S$

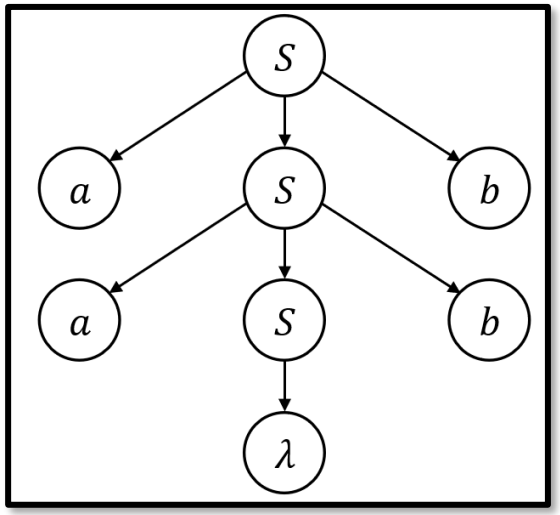
$S \Rightarrow aSb$

$S \Rightarrow a\mathbf{S}b \Rightarrow aSSb$
$S \Rightarrow a\mathbf{S}b \Rightarrow aaSbb$
$S \Rightarrow a\mathbf{S}b \Rightarrow abSab$
$S \Rightarrow a\mathbf{S}b \Rightarrow ab$

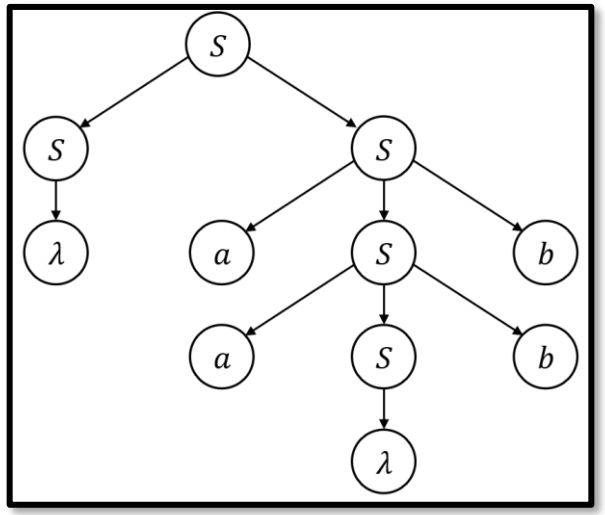
$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$$

Parsing and Ambiguity

- **Ambiguity**
  - A grammar is ambiguous if it derives some strings with two or more parse trees
  - Consider the grammar $S \rightarrow aSb \mid SS \mid \lambda$
    - ❖ The string $aabb$ can be derived from more than one parse tree



$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$



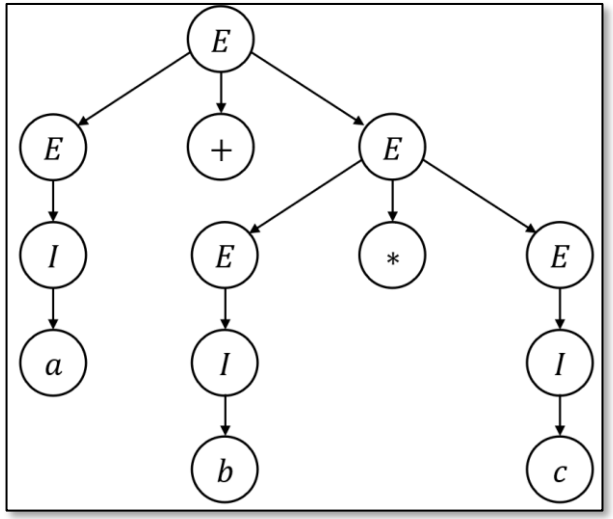
$S \Rightarrow SS \Rightarrow S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$

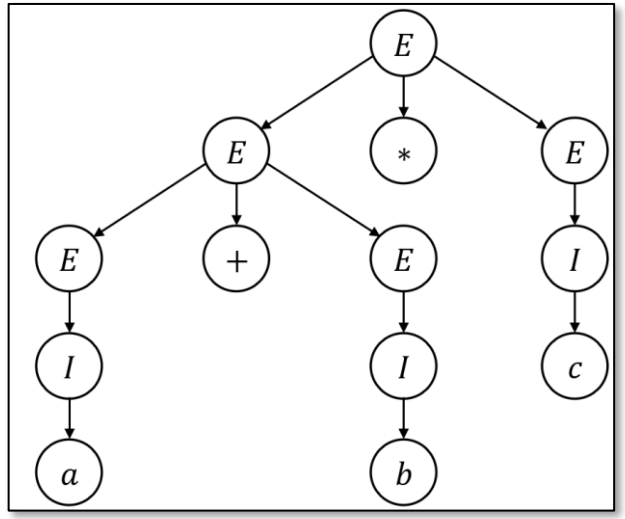
Parsing and Ambiguity

- **Ambiguity**
  - Unfortunately…
    - There is no general algorithm to remove ambiguity in CFGs
    - There is also no algorithm that determines that a CFG is ambiguous
  - Alternatively, we can develop an unambiguous grammar
    - By the use of precedence and associativity

Parsing and Ambiguity

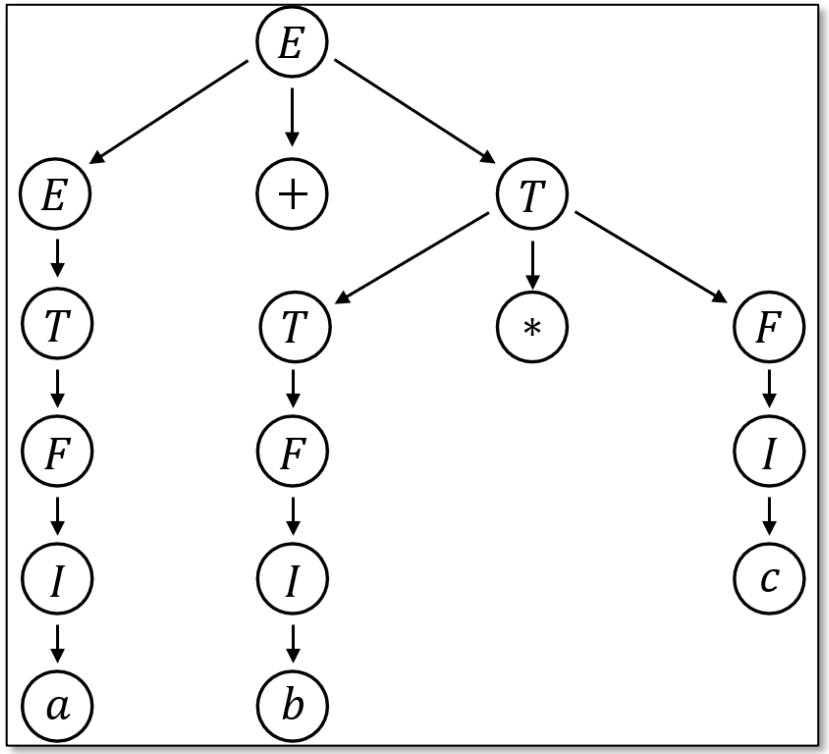
- **Eliminating ambiguity**
  - Consider the grammar $G = (\{E, I\}, \{a, b, c, +, *, (,)\}, E, P)$ with P given by
    - $E \rightarrow I$
    - $E \rightarrow E + E$
    - $E \rightarrow E * E$
    - $E \rightarrow (E)$
    - $I \rightarrow a \mid b \mid c$
  - This grammar is ambiguous
    - Consider the string $a + b * c$





Parsing and Ambiguity

- **Eliminating ambiguity**
  - We can resolve the ambiguity by prioritizing operators (e.g., $* \gg +$)
    - $E \rightarrow T$
    - $T \rightarrow F$
    - $F \rightarrow I$
    - $E \rightarrow E + T$
    - $T \rightarrow T * F$
    - $F \rightarrow (E)$
    - $I \rightarrow a \mid b \mid c$



Context-Free Grammars and Programming Languages

- **CFG can be used to represent a programming language**
  - One of the most important uses of the theory of formal languages
    - The definition of programming languages
    - The construction of interpreters and compilers

Context-Free Grammars and Programming Languages

- **CFG can be used to represent a programming language**
  - One of the most important uses of the theory of formal languages
    - ❖ The definition of programming languages
    - ❖ The construction of interpreters and compilers
  - Regular languages
    - ❖ Recognition of simple patterns
  - Context-free languages
    - ❖ Model more complicated aspects

Context-Free Grammars and Programming Languages

- **Backus-Naur Form (BNF)**
  - A form used in programming languages to express the grammar of the language as a mathematical formula
  - E.g.,

    <expression> ::= <term> | <expression> + <term>,

    <term> ::= <factor> | <term> * <factor>, …

    where * and + are terminal symbols, and ::= represents →

Context-Free Grammars and Programming Languages

- **Backus-Naur Form (BNF)**
  - A form used in programming languages to express the grammar of the language as a mathematical formula
  - E.g.,

    <expression> ::= <term> | <expression> + <term>,

    <term> ::= <factor> | <term> * <factor>, …

    where * and + are terminal symbols, and " ::= " represents " → "
  - E.g., the while statement in C language

    <while_statement> ::= while <expression> <statement>

Context-Free Grammars and Programming Languages

- **CFG can be used to represent a programming language**

$$
\begin{array}{lcl}
a & \rightarrow & n \mid x \mid a_1 + a_2 \mid a_1 \star a_2 \mid a_1 - a_2 \\
b & \rightarrow & \texttt{true} \mid \texttt{false} \mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg b \mid b_1 \wedge b_2 \\
c & \rightarrow & x := a \mid \texttt{skip} \mid c_1; c_2 \mid \texttt{if } b\ c_1\ c_2 \mid \texttt{while } b\ c
\end{array}
$$

<Excerpts from Professor Hakjoo Oh's lecture materials>

Context-Free Grammars and Programming Languages

- **CFG can be used to parse a programming language**
  - ANTLR (https://www.antlr.org)
    - ❖A parser generator widely used for reading or parsing structured text or binary files
    - ❖From a grammar, it generates a parser that can build and walk parse trees
  - LLVM (https://llvm.org)
    - ❖Compiler tool chain including parsers for popular languages

```
statement
    :   block
    |   ASSERT expression (':' expression)? ';'
    |   'if' parExpression statement ('else' statement)?
    |   'for' '(' forControl ')' statement
    |   'while' parExpression statement
    |   'do' statement 'while' parExpression ';'
    |   'try' block (catches finallyBlock? | finallyBlock)
    |   'try' resourceSpecification block catches? finallyBlock?
    |   'switch' parExpression '{' switchBlockStatementGroups '}'
    |   'synchronized' parExpression block
    |   'return' expression? ';'
    |   'throw' expression ';'
    |   'break' Identifier? ';'
    |   'continue' Identifier? ';'
    |   ';'
    |   statementExpression ';'
    |   Identifier ':' statement
    ;
```

Next Lecture

- **Simplification of Context-Free Grammars and Normal Forms**