Context-Free Grammars

CSE 211 (Theory of Computation)

Tanjeem Azwad Zaman

Adjunct Lecturer
Department of Computer Science and Engineering
Bangladesh University of Engineering & Technology

Adapted from slides by

Dr. Muhammad Masroor Ali & Dr. Atif Hasan Rahman

A CFG for palindromes

- $P \rightarrow \epsilon$
- $P \rightarrow 0$
- $P \rightarrow 1$
- $P \rightarrow 0P0$
- $P \rightarrow 1P1$

Formal definition of a Context-Free Grammar

- A ***context-free grammar*** is a 4-tuple (V, Σ, R, S) or (V, T, P, S), where
  1. V is a finite set called the ***variables***
  2. Σ or T is a finite set, disjoint from V, called the ***terminals***
  3. R or P is a finite set of ***rules*** or ***productions***, with each rule being a variable and a string of variables and terminals

     $$\text{variable} \rightarrow \text{string of variables and terminals}$$

  4. $S \in V$ is the ***start variable***
     - The variable in the first rule if not explicitly mentioned

Formal definition of a Context-Free Grammar

- Context-free grammar for the palindromes

$$G_{pal} = \{\{P\}, \{0, 1\}, R, P\}$$

1. $\{P\}$ is the set of variables
2. { 0,1 } is the set of terminals
3. R is the set of rules
   - $P \rightarrow \epsilon$
   - $P \rightarrow 0$
   - $P \rightarrow 1$
   - $P \rightarrow 0P0$
   - $P \rightarrow 1P1$
4. P is the start variable

CFG - an example

- A CFG that represents a simplification of expressions in a typical programming language
  - Restricted to the operators + and *
  - Identifiers (letters followed by zero or more letters and digits) allow only the letters a and b and the digits 0 and 1
  - We need two variables in this grammar
  - E represents expressions.
    - It is the start symbol
    - Represents the language of expressions we are defining
  - I represents identifiers
    - Its language is actually regular
    - It is the language of the regular expression

$$(a + b)(a + b + 0 + 1)^*$$

CFG - an example

$$
\begin{array}{llll}
1. & E & \rightarrow & I \\
2. & E & \rightarrow & E + E \\
3. & E & \rightarrow & E * E \\
4. & E & \rightarrow & (E) \\
\\
5. & I & \rightarrow & a \\
6. & I & \rightarrow & b \\
7. & I & \rightarrow & Ia \\
8. & I & \rightarrow & Ib \\
9. & I & \rightarrow & I0 \\
10. & I & \rightarrow & I1
\end{array}
$$

Figure 5.2: A context-free grammar for simple expressions

Derivation using a CFG

- The sequence of substitutions to obtain a string is called a ***derivation***
- The symbol $\Longrightarrow$ is used to denote ***yields*** or ***derives***
- * is used to denote zero or more steps
- The inference that $a * (a + b00)$ is in the language of variable E

$$E \Rightarrow E * E \Rightarrow I * E \Rightarrow a * E \Rightarrow$$

$$a * (E) \Rightarrow a * (E + E) \Rightarrow a * (I + E) \Rightarrow a * (a + E) \Rightarrow$$

$$a * (a + I) \Rightarrow a * (a + I0) \Rightarrow a * (a + I00) \Rightarrow a * (a + b00)$$

Leftmost and Rightmost Derivations

- ***Leftmost derivation***
  - At each step we replace the leftmost variable by one of its production bodies
- ***Rightmost derivation***
  - At each step we replace the rightmost variable by one of its production bodies

Leftmost Derivation

$$E \underset{lm}{\Rightarrow} E * E \underset{lm}{\Rightarrow} I * E \underset{lm}{\Rightarrow} a * E \underset{lm}{\Rightarrow}$$

$$a * (E) \underset{lm}{\Rightarrow} a * (E + E) \underset{lm}{\Rightarrow} a * (I + E) \underset{lm}{\Rightarrow} a * (a + E) \underset{lm}{\Rightarrow}$$

$$a * (a + I) \underset{lm}{\Rightarrow} a * (a + I0) \underset{lm}{\Rightarrow} a * (a + I00) \underset{lm}{\Rightarrow} a * (a + b00)$$

We can also summarize the leftmost derivation by saying $E \overset{*}{\underset{lm}{\Rightarrow}} a * (a + b00)$, or express several steps of the derivation by expressions such as $E * E \overset{*}{\underset{lm}{\Rightarrow}} a * (E)$.

Rightmost Derivation

$$E \underset{rm}{\Rightarrow} E * E \underset{rm}{\Rightarrow} E * (E) \underset{rm}{\Rightarrow} E * (E + E) \underset{rm}{\Rightarrow}$$

$$E * (E + I) \underset{rm}{\Rightarrow} E * (E + I0) \underset{rm}{\Rightarrow} E * (E + I00) \underset{rm}{\Rightarrow} E * (E + b00) \underset{rm}{\Rightarrow}$$

$$E * (I + b00) \underset{rm}{\Rightarrow} E * (a + b00) \underset{rm}{\Rightarrow} I * (a + b00) \underset{rm}{\Rightarrow} a * (a + b00)$$

This derivation allows us to conclude $E \overset{*}{\underset{rm}{\Rightarrow}} a * (a + b00)$. $\square$

Language of a grammar

- If $G = (V, \Sigma, R, S)$ is a CFG, the ***language*** of G denoted $L(G)$ is the set of terminal strings that have derivations from the start symbol

$$\text{The } \textit{language of the grammar} \text{ is } \{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\}.$$

- If a language L is the language of some context-free grammar, then L is said to be a ***context-free language*** or ***CFL***

Parse Trees

- Let $G = (V, \Sigma, R, S)$ be a CFG, the ***parse trees*** for G are trees with following conditions
  1. Each interior node is labeled by a variable in V
  2. Each leaf is labeled by either a variable, a terminal, or ϵ
     - However, if the leaf is labeled ϵ then it must be the only child of its parent
  3. If an interior node is labeled A, and its children are labeled

     $$X_1, X_2 \dots X_k$$

     respectively, from the left, then $A \to X_1 X_2 \dots X_k$ is a rule in R.
     - Note that the only time one of the X's can be ϵ is if that is the label of the only child, and $A \to \epsilon$ is a rule of G

Parse Tree - example

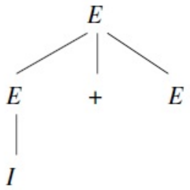

Figure 5.4: A parse tree showing the derivation of $I + E$ from E

Parse Tree - example

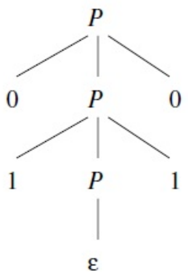

Figure 5.5: A parse tree showing the derivation $P \overset{*}{\Rightarrow} 0110$

Yield of a Parse Tree

- If we look at the leaves of any parse tree and concatenate them from the left, we get a string, called the ***yield*** of the tree
- Of special importance are those parse trees such that
  1. The yield is a terminal string, That is, all leaves are labeled either with a terminal or with ϵ
  2. The root is labeled by the start symbol
- These are the parse trees whose yields are strings in the **language of the underlying grammar**
- The **language of a grammar** is the set of yields of those parse trees having the **start symbol at the root** and a **terminal string as yield**

Parse Tree - example

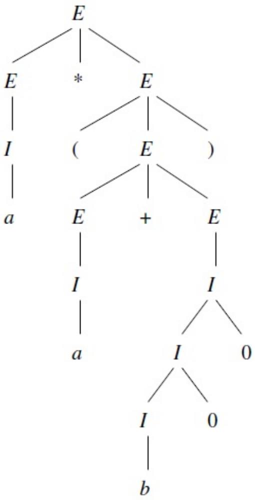

Figure 5.6: Parse tree showing $a * (a + b00)$ is in the language of our expression grammar

Inference, Derivations, and Parse Trees

- The following are equivalent
  - Recursive inference is a body to head process to infer membership of a string in a language

1. The recursive inference procedure determines that terminal string w is in the language of variable A.
2. $A \overset{*}{\Rightarrow} w$.
3. $A \underset{lm}{\overset{*}{\Rightarrow}} w$.
4. $A \underset{rm}{\overset{*}{\Rightarrow}} w$.
5. There is a parse tree with root A and yield w.

Inference, Derivations, and Parse Trees

- Proofs use induction

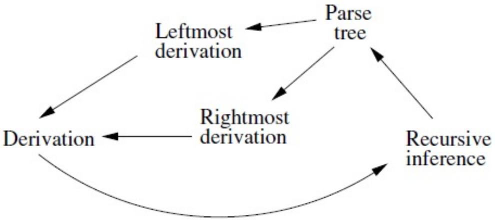

Figure 5.7: Proving the equivalence of certain statements about grammars

Designing CFGs

- Design a CFG, for the alphabet $\Sigma = \{0, 1\}$, for the language

$$\{w | w \text{ contains at least three 1s}\}$$

  - 100101
  - 01100101
  - 1111
  - ϵ
  - 00101000

Designing CFGs

- CFG for the language $\{w | w \text{ contains at least three 1s}\}$

Designing CFGs

- CFG for the language $\{w|w \text{ contains at least three 1s}\}$
- Three 1s separated, preceded, followed by any string

Designing CFGs

- CFG for the language $\{w|w \text{ contains at least three 1s}\}$
- Three 1s separated, preceded, followed by any string
  - $S \rightarrow X1X1X1X$

Designing CFGs

- CFG for the language $\{w|w \text{ contains at least three 1s}\}$
- Three 1s separated, preceded, followed by any string
  - $S \rightarrow X1X1X1X$
- Xs need to generate any string over $\Sigma = \{0, 1\}$

Designing CFGs

- CFG for the language $\{w|w \text{ contains at least three 1s}\}$
- Three 1s separated, preceded, followed by any string
  - $S \to X1X1X1X$
- Xs need to generate any string over $\Sigma = \{0, 1\}$
  - $X \to 0X|1X|\epsilon$

Designing CFGs

- Designing CFGs require creativity
- Some techniques are helpful
  - Some CFLs are the union of simpler CFLs. Construct a grammar for each piece and combine them with a rule like

$$S \rightarrow S_1 | S_2 | \cdots | S_k$$

  - If the CFL happens to be regular and if you can construct a DFA, it can be easily converted to a CFG
    - Make a variable R_i for each state q_i in the DFA
    - Add the rule $R_i \rightarrow aR_j$ if $\delta(q_i, a) = q_j$ is a transition
    - Add the rule $R_i \rightarrow \epsilon$ if q_i is an accept state

Exercise

- Design a DFA for the language $\{w | w$ contains at least three 1s$\}$ and convert it to CFG