Unit 4.1 Context-Free Grammars

- Introduction to Context Free Grammar (CFG)
- Components of CFG
- Context Free Language (CFL)
- Derivation
- BNF Notation

Introduction

- Any language(Formal/Natural) has grammar to describe that language.
- A *context-free grammar* is a notation for describing formal languages.
- It is more powerful tool than finite automata or RE's, but still cannot define all possible languages.
- Useful for nested structures, e.g., parentheses in programming languages.

Introduction

- Basic idea is to use "variables" to stand for sets of strings (i.e., languages).
- These variables are defined recursively, in terms of one another.
- Recursive rules ("productions") involve only concatenation.
- Alternative rules for a variable allow union.

Example: CFG for $\{ 0^n1^n \mid n \geq 1\}$

- Productions:

  S -> 01

  S -> 0S1
- Basis: 01 is in the language.
- Induction: if w is in the language, then so is 0w1. (Recursive Rule)

Components of CFG

- *Terminals* = symbols of the alphabet of the language being defined.
- *Variables* = *nonterminals* = a finite set of other symbols, each of which represents a language construct.
- *Start symbol* = the variable whose language is the one being defined.

Components of CFG

- A *production* has the form
  - variable -> string of variables and terminals.
- Convention:
  - A, B, C,… are variables.
  - a, b, c,… are terminals.
  - …, X, Y, Z are either terminals or variables.
  - …, w, x, y, z are strings of terminals only.
  - α, β, γ,… are strings of terminals and/or variables.

Example: Formal CFG

- Here is a formal CFG for $\{ 0^n1^n \mid n \geq 1\}$.
- Terminals = {0, 1}.
- Variables = {S}.
- Start symbol = S.
- Productions =

  S -> 01

  S -> 0S1

Formal Definition of CFG

- A Context Free Grammar(CFG) is defined by 4-tuples as G=(V,T,P,S) where
  - V=Set of Variables
  - T= Set of Terminals
  - P=Set of Productions
  - S= Start Variable, S ϵ V

Derivations – Intuition

- We *derive* strings in the language of a CFG by starting with the start symbol, and repeatedly replacing some variable A by the right side of one of its productions.
  - That is, the "productions for A" are those that have A on the left side of the ->.

Derivations :Example

- We say $\alpha A\beta \Rightarrow \alpha\gamma\beta$ if A -> γ is a production.
- Example: S -> 01; S -> 0S1.
- S => 0S1 => 00S11 => 000111.

Iterated Derivation

- $=>^*$ means "zero or more derivation steps."
- Basis: $\alpha =>^* \alpha$ for any string α.
- Induction: if $\alpha =>^* \beta$ and $\beta => \gamma$, then $\alpha =>^* \gamma$.

Example: Iterated Derivation

- Let a CFG is:
  - S -> 01
  - S -> 0S1.
- S => 0S1 => 00S11 => 000111.
- So, S =>* S; S =>* 0S1;
  S =>* 00S11; S =>* 000111.

Sentential Forms

- Any string of variables and/or terminals derived from the start symbol is called a *sentential form*.
- Formally, α is a sentential form iff $S \Rightarrow^* \alpha$.
- In previous example,

$S \Rightarrow^* S$; $S \Rightarrow^* 0S1$;

$S \Rightarrow^* 00S11$; $S \Rightarrow^* 000111$.

All are sentential forms

Language of a Grammar

- If G is a CFG, then L(G), the *language of G*, is {w | S =>* w}.
  - Note: w must be a terminal string, S is the start symbol.
- Example: G has productions S -> ε and S -> 0S1.
- L(G) = $\{0^n1^n \mid n \geq 0\}$.

Note: ε is a legitimate right side.

Context-Free Languages

- A language that is defined by some CFG is called a *context-free language*.
- There are CFL’s that are not regular languages, such as the example just given above and we have proved that the language is not regular using pumping lemma.
- But not all languages are CFL’s.
- The programming languages are CFL since they are described by CFG

Top-down and Bottom-up Derivations

- **Top-down:** Derivations of string starting from start variable and by replacing variable at each step to reach up to the string.
- **Bottom-up:** Derivation process starting from a string and reducing the substrings by a variable applying any production to get start variable.

Example: Top-down Derivations

- Grammar for Strings of Balanced-parentheses

  S -> SS | (S) | ()

- S => SS

  => S()

  => (S)()

  => (())() is from top-down derivation

Example: Bottom-up Derivations

- Grammar for Strings of Balanced-parentheses

  S -> SS | (S) | ()

Bottom-up derivation for string: (())()

String	Variable	Production	String used
()	S	S->()	-
(())	S	S->(S)	()
(())()	S	S->SS	(()) and ()

Hence we got from bottom up from string to start variable as:

(())() => (S)() => S() => SS => S

Leftmost and Rightmost Derivations

- Derivations allow us to replace any of the variables in a string.
- Leads to many different derivations of the same string.
- By forcing the leftmost variable (or alternatively, the rightmost variable) to be replaced, the derivation can be Leftmost or rightmost.

Leftmost Derivations

- Say $wA\alpha \Rightarrow_{lm} w\beta\alpha$ if w is a string of terminals only and $A \rightarrow \beta$ is a production.
- Also, $\alpha \Rightarrow^*_{lm} \beta$ if α becomes β by a sequence of 0 or more $\Rightarrow_{lm}$ steps.

Example: Leftmost Derivations

- Balanced-parentheses grammmar:

  S -> SS | (S) | ()

- $S \Rightarrow_{lm} SS \Rightarrow_{lm} (S)S \Rightarrow_{lm} (())S \Rightarrow_{lm} (())()$
- Thus, $\mathbf{S \Rightarrow^*_{lm} (())()}$
- $S \Rightarrow SS \Rightarrow S() \Rightarrow (S)() \Rightarrow (())()$ is a derivation, but not a leftmost derivation.

Rightmost Derivations

- Say $\alpha A w \Rightarrow_{rm} \alpha\beta w$ if w is a string of terminals only and $A \to \beta$ is a production.
- Also, $\alpha \Rightarrow^*_{rm} \beta$ if α becomes β by a sequence of 0 or more $\Rightarrow_{rm}$ steps.

Example: Rightmost Derivations

- Balanced-parentheses grammmar:

  S -> SS | (S) | ()

- $S \Rightarrow_{rm} SS \Rightarrow_{rm} S() \Rightarrow_{rm} (S)() \Rightarrow_{rm} (())()$
- Thus, $S \Rightarrow^*_{rm} (())()$
- $S \Rightarrow SS \Rightarrow SSS \Rightarrow S()S \Rightarrow ()()S \Rightarrow ()()()$ is neither a rightmost nor a leftmost derivation.

Example : Leftmost Derivation

- Given a grammar:
  - **E->E+E , E-> E*E, E->(E), E->E-E, E->a, E->b**
- Derivation for String **a*(a+b)-a**
  - E $=>_{lm}$ **E**-E

    $=>_{lm}$ **E***E-E

    $=>_{lm}$ a***E**-E

    $=>_{lm}$ a*(**E**)-E

    $=>_{lm}$ a*(**E**+E)-E

    $=>_{lm}$ a*(a+**E**)-E

    $=>_{lm}$ a*(a+b)-E

    $=>_{lm}$ **a*(a+b)-a**

Example : Rightmost Derivation

- Given a grammar:
  - **E->E+E , E-> E*E, E->(E), E->a, E->E-E, E->b**
- Derivation for String a*(a+b)-a
  - E $\Rightarrow_{rm}$ E-**E**
    $\Rightarrow_{rm}$ **E**-a
    $\Rightarrow_{rm}$ E*__E__-a
    $\Rightarrow_{rm}$ E*(**E**)-a
    $\Rightarrow_{rm}$ E*(E+**E**)-a
    $\Rightarrow_{rm}$ E*(**E**+b)-a
    $\Rightarrow_{rm}$ **E***(a+b)-a
    $\Rightarrow_{rm}$ a*(a+b)-a

BNF Notation

- Grammars for programming languages are often written in BNF (*Backus-Naur Form*).
- Variables are words in <…>;
  - Example: <statement>.
- Terminals are often multi-character strings indicated by boldface or underline;
  - Example: **while** or <u>WHILE</u>.

BNF Notation

- Symbol ::= is often used for ->.
- Symbol | is used for "or."
  - A shorthand for a list of productions with the same left side.
- Example: S -> 0S1 | 01 is shorthand for **S -> 0S1 and S -> 01.**

BNF Notation – Kleene Closure

- Symbol **...** is used for "one or more."
  - Example: **<digit> ::= 0|1|2|3|4|5|6|7|8|9**

    <unsigned integer> ::= <digit>...

- Translation: Replace α**...** with a new variable A and productions A -> Aα | α.

Example: Kleene Closure

- Grammar for unsigned integers can be replaced by:

  U -> UD | D

  D -> 0|1|2|3|4|5|6|7|8|9

In BNF:

<unsigned integer>::=<unsigned integer><digit>
|<digit>

<digit>::=0|1|2|3|4|5|6|7|8|9

BNF Notation: Optional Elements

- Surround one or more symbols by [...] to make them optional.
- Example: <statement> ::= **if** <condition> **then** <statement> [; **else** <statement>]
- Translation: replace [α] by a new variable A with productions A -> α | ϵ.

Example: Optional Elements

- Grammar for if-then-else can be replaced by:

S -> iCtSA

A -> ;eS | ϵ

BNF Notation – Grouping

- Use {…} to surround a sequence of symbols that need to be treated as a unit.
  - Typically, they are followed by a … for "one or more."
- Example: <statement list> ::= <statement> [{;<statement>}…]

Translation: Grouping

- You may, if you wish, create a new variable A for $\{\alpha\}$.
- One production for A: A -> α.
- Use A in place of $\{\alpha\}$.

Example: Grouping

L -> S [{;S}…]

- Replace by L -> S [A…]   A -> ;S
  - A stands for {;S}.
- Then by L -> SB B -> A… | ϵ A -> ;S
  - B stands for [A…] (zero or more A's).
- Finally by L -> SB   B -> C | ϵ
  C -> AC | A   A -> ;S
  - C stands for A… .