Theory of Automata
Chomsky Normal Form

Week 8

Contents

- Definition – CNF
- Theorem
- Left most derivation
- Parsing

Chomsky Normal Form (CNF)

Definition

If a CFG has only productions of the form

Non-Terminal → string of exactly two non-terminals

OR

Non-Terminal → one terminal

It is said to be in Chomsky Normal Form (CNF).

Theorem 25

- If L is a language generated by some CFG, then there exist another CFG that generate all the non-null words of L, all of whose productions are of one of the two basic forms:

Non-Terminal → string of only Non-terminals

Or

Non-Terminal → one Terminal

Proof

- The proof is a constructive algorithm by converting a given CFG in a CFG of the production of designed format.

- Let us suppose in given CFG the non-terminal are S, X_1, X_2, X_3, … . Let us also assume {a,b} are two terminals.

STEP-1

We now add two new non-terminals A and B and the productions

$A \rightarrow a$ $B \rightarrow b$

Proof Theorem 25 contd…

STEP-2

now for every previous production involving terminals, we replace each ‘a’ with the non-terminal A and each ‘b’ with B.

e.g. $X_3 \rightarrow X_4aX_1SbbX_7a$

Becomes

$X_3 \rightarrow X_4AX_1SBBX_7A$

Which is a string of solid non-terminals

$X_6 \rightarrow abaab$

Becomes

$X_6 \rightarrow ABAAB$

Example

$S \rightarrow X_1 \mid X_2aX_2 \mid aSb \mid b$

$X_1 \rightarrow X_2X_2 \mid b$

$X_2 \rightarrow aX_2 \mid aaX_2$

$S \rightarrow X_1 \mid X_2AX_2 \mid ASB \mid B$

$X_1 \rightarrow X_2X_2 \mid B$

$X_2 \rightarrow AX_2 \mid AAX_2$

$A \rightarrow a$

$B \rightarrow b$

Theorem

- If CFG G = (V, ∑, P, S) is in CNF form if each rule has one of the following form
  1. A → BC
  2. A → a
  3. S → Λ

  Where B, C ∈ V – {S}
- Pre-Conditions
  i. The start symbol is non-recursive
  ii. G does not have lambda rules other than S → Λ
  iii. G does not contain unit/chain rules
  iv. G does not contain useless symbols

Proof of theorem

- The rules of type ii, and iii already fulfill CNF condition so that become part of CNF directly
- Right sides of other production may contain terminals and non-terminals

  e.g. N → aM | NaMb
- We need to have string of solid non-terminal on right side we introduce new products as X→ a Y → b

And apply them to all rules getting

N → XM | NXMY

Proof contd.

- Introduce new non-terminals to restrict the length to 2
- N → XM | NXMY

$N \rightarrow XM$

$N \rightarrow NR_1$

$R_1 \rightarrow XR_2$

$R_2 \rightarrow MY$

$X \rightarrow a$

$Y \rightarrow b$

Theorem 26

Theorem 26:

For any context-free language L, the non-Λ words of L can be generated by a grammar in which all productions are in CNF.

Proof of Theorem 26

- The proof will be by constructive algorithm.
- From Theorems 23 and 24, we know that there is a CFG for L (or for all L except Λ) that has no Λ-productions and no unit productions.
- Suppose further that we have made the CFG fit the form specified in Theorem 25.
- For productions of the form

  non-terminal → one terminal

  we leave them alone.
- For each production of the form

  non-terminal → string of non-terminals

  we use the following expansion that involves new non-terminals R_1, R_2, ...

Proof of Theorem 26 contd.

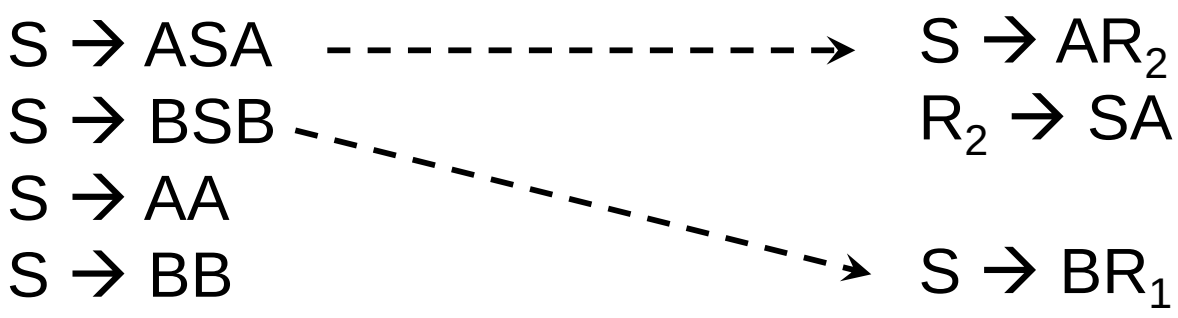
- For instance, the production $S \rightarrow X_1X_2X_3X_8$ should be replaced by $S \rightarrow X_1R_1$
  where $R_1 \rightarrow X_2R_2$ and where $R_2 \rightarrow X_3X_8$.
- Thus, we convert a production with long string of non-terminals into a sequence of productions with exactly two non-terminals.
- If we need to convert another production, we introduce new R's with different subscripts.
- The new grammar generates the same language as the old grammar with the possible exception of the word Λ.

Remarks

- Theorem 26 has enabled a form of algebra of productions by introducing non-terminals R_1, R_2, R_3 …
- The derivation tree associated with a derivation in Chomsky Normal Form grammar is a binary tree.

Example (Non-Palindrome)

$S \rightarrow aSa \mid bSb \mid a \mid b \mid aa \mid bb$



$S \rightarrow AR_2$
$R_2 \rightarrow SA$

$S \rightarrow BR_1$
$R_1 \rightarrow SB$

$S \rightarrow AA$
$S \rightarrow BB$
$S \rightarrow a$
$S \rightarrow b$
$A \rightarrow a$
$B \rightarrow b$

Leftmost Derivation

Definition:

- The **leftmost non-terminal** in a working string is the first *non-terminal* that we encounter when we scan the string from left to right.

Example: In the string abNbaXYa, the leftmost non-terminal is N.

Definition:

- If a word *w* is generated by a CFG such that at each step in the derivation, a production is applied to the leftmost non-terminal in the working string, then this derivation is called a **leftmost derivation**.

Example: Left derivation

S → aSX | b

X → Xb | a

aababa

S → aSX

→ aaSXX

→ aabXX

→ aabXbX

→ aababX

→ aababa

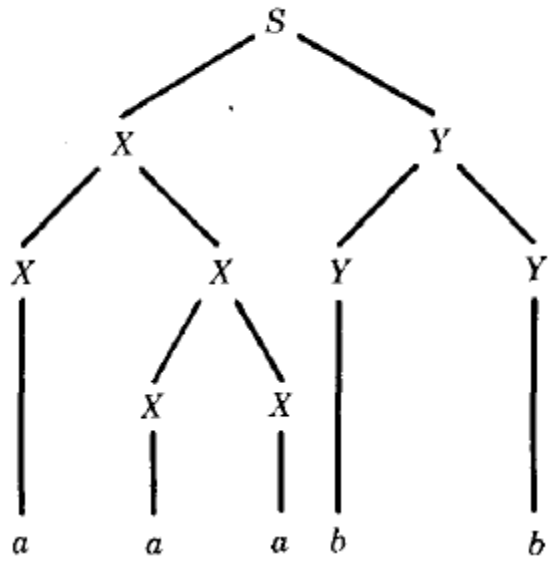
Left derivation

$S \rightarrow XY$

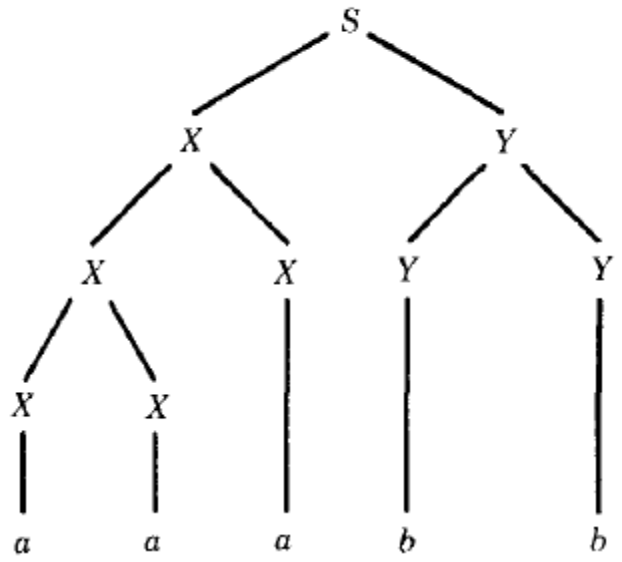
$X \rightarrow XX \mid a$

$Y \rightarrow YY \mid b$

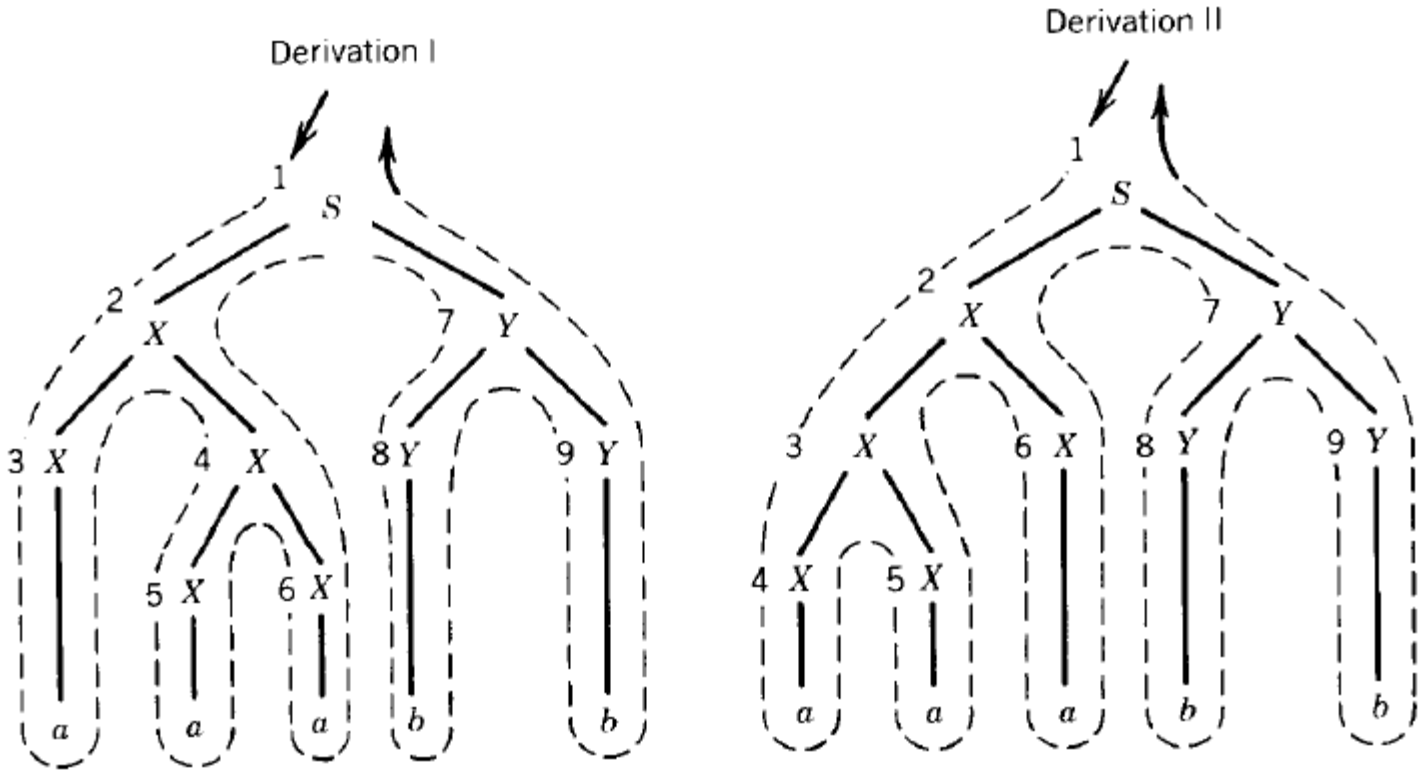
Derivation I



Derivation II



Left derivation



Left derivation

Derivation I

1. $S \Rightarrow \dot{X}Y$
2. $\Rightarrow \dot{X}XY$
3. $\Rightarrow a\dot{X}Y$
4. $\Rightarrow a\dot{X}XY$
5. $\Rightarrow aa\dot{X}Y$
6. $\Rightarrow aaa\dot{Y}$
7. $\Rightarrow aaa\dot{Y}Y$
8. $\Rightarrow aaab\dot{Y}$
9. $\Rightarrow aaabb$

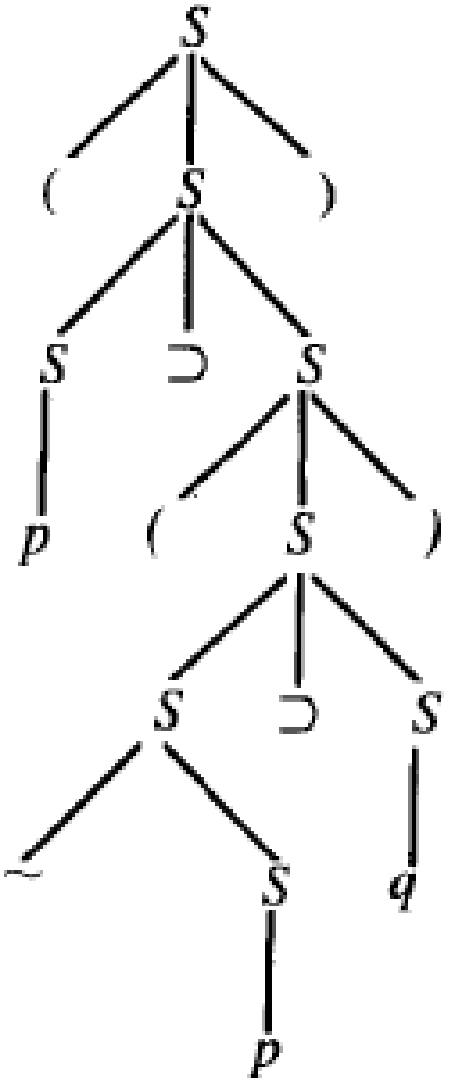
Derivation II

1. $S \Rightarrow \dot{X}Y$
2. $\Rightarrow \dot{X}XY$
3. $\Rightarrow \dot{X}XXY$
4. $\Rightarrow a\dot{X}XY$
5. $\Rightarrow aa\dot{X}Y$
6. $\Rightarrow aaa\dot{Y}$
7. $\Rightarrow aaa\dot{Y}Y$
8. $\Rightarrow aaab\dot{Y}$
9. $\Rightarrow aaabb$

$$S \rightarrow S \supset S \mid \sim S \mid (S) \mid p \mid q$$

$$(p \supset (\sim p \supset q))$$

$$\begin{aligned}
S &\Rightarrow (\dot{S}) \\
&\Rightarrow (\dot{S} \supset S) \\
&\Rightarrow (p \supset \dot{S}) \\
&\Rightarrow (p \supset (\dot{S})) \\
&\Rightarrow (p \supset (\dot{S} \supset S)) \\
&\Rightarrow (p \supset (\sim \dot{S} \supset S)) \\
&\Rightarrow (p \supset (\sim p \supset \dot{S})) \\
&\Rightarrow (p \supset (\sim p \supset q))
\end{aligned}$$



Parsing

Definition: the process of finding the derivation of word generated by particular grammar is called **parsing**.

There are different parsing techniques, containing the following three

1. Top down parsing.
2. Bottom up parsing.
3. Parsing technique for particular grammar of arithmetic expression.

Top down parsing

Following is an example showing top down parsing technique

As can be observed from the name of top down parsing, the parsing starts from the non-terminal S and the structure similar to that of total language tree is developed. The branches of the tree are extended till the required word is found as a branch.

Convert to CNF

- E -> E+E
- E -> E*E
- E -> (E)
- E -> 7

- S->ABABAB
- A->a|Δ
- B->b| Δ

- S->SaS |SaSbS |SbSaS | Δ