Normal Forms for CFG's

- Eliminating Useless Variables
- Removing Epsilon
- Removing Unit Productions
- Chomsky Normal Form

Variables That Derive Nothing

- Consider a CFG:
  - S -> AB
  - A -> aA | a,
  - B -> AB
- Although A derives all strings of a's, B derives no terminal strings.
- Thus, S derives nothing, and the language is **empty.**

Testing Whether a Variable Derives Some Terminal String

- Basis: If there is a production A -> w, where w has no variables, then A derives a terminal string.
- Induction: If there is a production A -> α, where α consists only of terminals and variables known to derive a terminal string, then A derives a terminal string.

Algorithm to Eliminate Variables That Derive Nothing: Non Generating Symbols

1. Discover all variables that derive terminal strings.
2. For all other variables, remove all productions in which they appear either on the left or the right.

Example: Eliminate Non Generating Variables

S -> AB | C, A -> aA | a, B -> bB, C -> c

- Basis: A and C are identified because of A -> a and C -> c.
- Induction: S is identified because of S -> C.
- Nothing else can be identified.
- Result: S -> C, A -> aA | a, C -> c

Unreachable Symbols

- Another way a terminal or variable deserves to be eliminated is if it cannot appear in any derivation from the start symbol.
- Basis: We can reach S (the start symbol).
- Induction: if we can reach A, and there is a production A -> α, then we can reach all symbols of α.

Unreachable Symbols

- Easy inductions in both directions show that when we can discover no more symbols, then we have all and only the symbols that appear in derivations from S.
- Algorithm: Remove from the grammar all symbols not discovered reachable from S and all productions that involve these symbols.

Eliminating Useless Symbols

- A symbol is *useful* if it appears in some derivation of some terminal string from the start symbol.
- Otherwise, it is *useless*.
  Eliminate all useless symbols by:
  1. Eliminate symbols that derive no terminal string.
  2. Eliminate unreachable symbols.

Example: Useless Symbols

S -> AB|AC, A -> C, C -> c, B -> bB

- If we eliminated unreachable symbols first, we would find everything is reachable. So no unreachable here.
- For testing generating symbol, we can get A , C and S are generating and B is not.
- A, C,S would never get eliminated but B is eliminated.
- So after eliminating useless symbols:

S -> AC, A -> C, C -> c

Epsilon Productions

- We can almost avoid using productions of the form A -> ϵ (called *ϵ-productions*).
  - The problem is that ϵ cannot be in the language of any grammar that has no ϵ–productions.
- Theorem: If L is a CFL, then L-{ϵ} has a CFG with no ϵ-productions.

Nullable Symbols

- To eliminate ϵ-productions, we first need to discover the *nullable variables* = variables A such that A =>* ϵ.
- Basis: If there is a production A -> ϵ, then A is nullable.
- Induction: If there is a production A -> α, and all symbols of α are nullable, then A is nullable.

Example: Nullable Symbols

S -> AB, A -> aA | ϵ, B -> bB | A

- Basis: A is nullable because of A -> ϵ.
- Induction: B is nullable because of B -> A.
- Then, S is nullable because of S -> AB.

Eliminating ϵ-Productions

- Key idea: turn each production $A \to X_1 \ldots X_n$ into a family of productions.
- For each subset of nullable X's, there is one production with those eliminated from the right side "in advance."
  - Except, if all X's are nullable, do not make a production with ϵ as the right side.

Example: Eliminating ϵ-Productions

S -> ABC, A -> aA | ϵ, B -> bB | ϵ, C -> ϵ

- A, B, C, and S are all nullable.
- New grammar:

S -> ~~ABC~~ | AB | ~~AC~~ | ~~BC~~ | A | B | ~~C~~

A -> aA | a

B -> bB | b

Note: C is now useless.
Eliminate its productions.

Removal of Unit Productions

- A *unit production* is one whose right side consists of exactly one variable.
- These productions can be eliminated.
- Key idea: If A =>* B by a series of unit productions, and B -> α is a non-unit-production, then add production A -> α.
- Then, drop all unit productions.

Unit Productions – (2)

- Find all pairs (A, B) such that A =>* B by a sequence of unit productions only.
- Basis: Surely (A, A).
- Induction: If we have found (A, B), and B -> C is a unit production, then add (A, C).

Proof That We Find Exactly the Right Pairs

- By induction on the order in which pairs (A, B) are found, we can show A =>* B by unit productions.
- Conversely, by induction on the number of steps in the derivation by unit productions of A =>* B, we can show that the pair (A, B) is discovered.

Simplification of CFG

- Theorem: if L is a CFL, then there is a CFG for L – {ϵ} that has:
  1. No useless symbols.
  2. No ϵ-productions.
  3. No unit productions.
- i.e., every right side is either a single terminal or has length $\geq$ 2.

Simplification of CFG

- Proof: Start with a CFG for L.
- Perform the following steps in order:
  1. Eliminate ϵ-productions(Nullable).
  2. Eliminate unit productions.
  3. Eliminate variables that derive no terminal string (useless).
  4. Eliminate variables not reached from the start symbol (useless).

Must be first. Can create unit productions or useless variables.

Chomsky Normal Form(CNF)

- A CFG is said to be in *Chomsky Normal Form* if every production is of one of these two forms:
  1. A -> BC (right side is two variables).
  2. A -> a (right side is a single terminal).
- Theorem: If L is a CFL, then L – {ϵ} has a CFG in CNF.

Proof of CNF Theorem

- Step 1: "Simplify" the grammar, so every production right side is either a single terminal or of length at least 2.
- Step 2: For each right side $\neq$ a single terminal, make the right side all variables.
  - For each terminal *a* create new variable A_a and production A_a -> a.
  - Replace *a* by A_a in right sides of length > 2.

Example: Step 2

- Consider production A -> BcDe.
- We need variables A_c and A_e. with productions A_c -> c and A_e -> e.
  - Note: you create at most one variable for each terminal, and use it everywhere it is needed.
- Replace A -> BcDe by A -> BA_cDA_e.

CNF Proof – Continued

- Step 3: Break right sides longer than 2 into a chain of productions with right sides of two variables.
- Example: A -> BCDE is replaced by A -> BF, F -> CG, and G -> DE.
  - F and G must be used nowhere else.

Example of Step 3 – Continued

- Recall A -> BCDE is replaced by A -> BF, F -> CG, and G -> DE.
- In the new grammar, A => BF => BCG => BCDE.
- More importantly: Once we choose to replace A by BF, we must continue to BCG and BCDE.
  - Because F and G have only one production.

CNF: Example

- Given CFG, Convert to CNF
  - **S -> AAC**
  - **A-> aAb|ϵ**
  - **C -> aC | a**
- Eliminating ϵ-Production:
  - Here Variable A is Nullable since A -> ϵ , so eliminating ϵ-production
    - **S -> AAC|AC|C**
    - **A -> aAb | ab**
    - **C -> aC | a**

CNF: Example

- Eliminating unit-Production:
  - Discover the unit pair first
  - (S,C) are unit pairs since S -> C , so eliminating Unit productions we have
    - **S -> AAC|AC|aC|a**
    - **A -> aAb | ab**
    - **C -> aC | a**
- **Here are no useless symbols since S, A and C all variables are generating and reachable**

CNF: Example

- **Hence Simplified CFG is:**
  - **S -> AAC|AC|aC|a**
  - **A -> aAb | ab**
  - **C -> aC | a**
- **Converting to CNF**
  - **For each production of the form**

    A ->$X_1X_2X_3$… … X_n for which any X_i introduce new variable for that terminal , So
  - **S->AAC |AC |X_aC |a**
  - **X_a -> a**
  - **A->X_aAX_b | X_aX_b**
  - **X_b -> b**
  - **C ->X_aC |a**

CNF: Example

- **For $S \rightarrow AAC$, Replace with $S \rightarrow AT_1$, $T_1 \rightarrow AC$**
- **For $A \rightarrow X_aAX_b$, Replace with $A \rightarrow X_aT_2$, $T_2 \rightarrow AX_b$**

- **So grammar has now all the productions in any one of the form $A \rightarrow BC$ or $A \rightarrow a$. So the CNF grammar is:**
  - $S \rightarrow AT_1 \mid AC \mid X_aC \mid a$
  - $T_1 \rightarrow AC$
  - $X_a \rightarrow a$
  - $A \rightarrow X_aT_2 \mid X_aX_b$
  - $T_2 \rightarrow AX_b$
  - $X_b \rightarrow b$
  - $C \rightarrow X_aC \mid a$

Exercise: Convert to CNF

- **S -> AACD**
- **A-> aAb |ϵ**
- **C -> aC | a**
- **D -> aDa | bDb | ϵ**
- **Solution:**
  - **Removing ϵ productions:**
    - Here A and D are nullable since **A -> ϵ , D -> ϵ**. So removing ϵ-production
    - Replace **S ->AACD** by **S->AACD | ACD| AAC|AC|CD|C**
    - Replace **A->aAb** by **A->aAb|ab**
    - Replace **D->aDa** by **D->aDa | aa** and **D->bDb** by **D->bDb|bb**

- Grammar after removing ϵ-Prodution
  - **S -> AACD|ACD|AAC|AC|CD|C**
  - **A-> aAb |ab**
  - **C -> aC | a**
  - **D -> aDa |aa| bDb | bb**
- **Removing Unit Production:** only (S,C) is unit pair since S->C and no others, so removing unit production
  - **Replace S->C by S->aC|a i.e. non unit production of C**
- Now, Grammar will be
  - **S -> AACD|ACD|AAC|AC|CD|aC|a**
  - **A-> aAb |ab**
  - **C -> aC | a**
  - **D -> aDa |aa| bDb | bb**

Example….

- Removing the useless symbols: Here Variable S,A,C D all are generating symbols. Also all variables are reachable so no useless symbols. So simplified grammar is :
  - **S -> AACD|ACD|AAC|AC|CD|aC|a**
  - **A-> aAb |ab**
  - **C -> aC | a**
  - **D -> aDa |aa| bDb | bb**
- **Convert this grammar into CNF**
- **At first converting terminals which are not single at any productions.**
  - **Replace S->aC by S-> X_aC and X_a->a**
  - **Replace A->aAb by A-> X_aAX_b and X_b->b**
  - **Replace C->aC by C->X_aC**
  - **Replace D->aDa by D->X_aDX_a and D->bDb by D->X_bDX_b**

- **Now Grammar will be:**
  - **$S \to AACD|ACD|AAC|AC|CD|X_aC|a$**
  - **$X_a \to a$**
  - **$A \to X_aAX_b\ |X_aX_b$**
  - **$X_b \to b$**
  - **$C \to X_aC\ |\ a$**
  - **$D \to X_aDX_a\ |X_aX_a|\ X_bDX_b\ |\ X_bX_b$**
- Now replacing production on right side having more than 2 variables
  - Replace $S \to AACD$ by $S \to AE$, $E \to AF$ and $F \to CD$
  - Replace $S \to ACD$ by $S \to AF$
  - Replace $S \to AAC$ by $S \to AG$, $G \to AC$
  - Replace $A \to$ **X_aAX_b by $A \to X_aH$ and $H \to Ax_b$**
  - **Replace $D \to X_aDX_a$ by $D \to X_aI$ and $I \to DX_a$**
  - Replace $D \to$ **X_bDX_b by $D \to X_bJ$ and $J \to DX_b$**

Now Grammar in CNF will be:

- $S \to AE|AF|AG|AC|CD|X_aC|a$
- $E \to AF$
- $F \to CD$
- $G \to AC$
- $X_a \to a$
- $A \to X_aH \mid X_aX_b$
- $H \to AX_b$
- $X_b \to b$
- $C \to X_aC \mid a$
- $D \to X_aI \mid X_aX_a \mid X_bJ \mid X_bX_b$
- $I \to Dx_a$
- $J \to DX_b$

Greibach Normal Form

- A CFG is in Greibach Normal Form if the Productions are in the following forms –
- $A \rightarrow b$
- $A \rightarrow bD_1 \ldots D_n$

where $A, D_1, \ldots, D_n$ are non-terminals and b is a terminal.

Algorithm to Convert a CFG into GNF

- **Step 1** – If the start symbol **S** occurs on some right side, create a new start symbol **S’** and a new production **S’** → **S**.
- **Step 2** – Remove Null productions. (Using the Null production removal algorithm discussed earlier)
- **Step 3** – Remove unit productions. (Using the Unit production removal algorithm discussed earlier)
- **Step 4** – Remove all direct and indirect left-recursion.
- **Step 5** – Do proper substitutions of productions to convert it into the proper form of GNF.

GNF: Example

- Convert the following CFG into CNF
  - $S \rightarrow XY \mid Xb \mid d$
  - $X \rightarrow aX \mid a$
  - $Y \rightarrow Xb \mid c$

Solution

- Here, **S** does not appear on the right side of any production and there are no unit or null productions in the production rule set. So, we can skip Step 1 to Step 3.

GNF:Example

Step 4

- Now after replacing
  - X in $\mathbf{S \to XY \mid Xc \mid d}$

  With $\mathbf{aX \mid a}$ we obtain
  - $\mathbf{S \to aXY \mid aY \mid aXc \mid mc \mid d.}$
- And after replacing X in $Y \to Xb \mid c$ with the right side of $X \to aX \mid a$ we obtain $\mathbf{Y \to aXb \mid ab \mid c.}$

GNF: Example

- Two new productions $C \rightarrow c$ and $D \rightarrow d$ are added to the production set and then we came to the final GNF as the following –
  - **$S \rightarrow aXY \mid aY \mid aXC \mid aC \mid d$**
  - **$X \rightarrow aX \mid a$**
  - **$Y \rightarrow aXD \mid aD \mid c$**
  - **$C \rightarrow c$**
  - **$D \rightarrow d$**
- **Now all productions are in the GNF**

Left Recursion

- A grammar becomes left-recursive if it has any non-terminal ‘A’ whose derivation contains ‘A’ itself as the left-most symbol.
- Left-recursive grammar is considered to be a problematic situation for top-down parsers.
- It becomes hard for it to judge when to stop parsing the left non-terminal and it goes into an infinite loop.

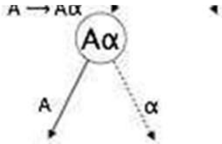
Example of Left Recursion

1. $A \Rightarrow A\alpha \mid \beta$ direct left recursion for A

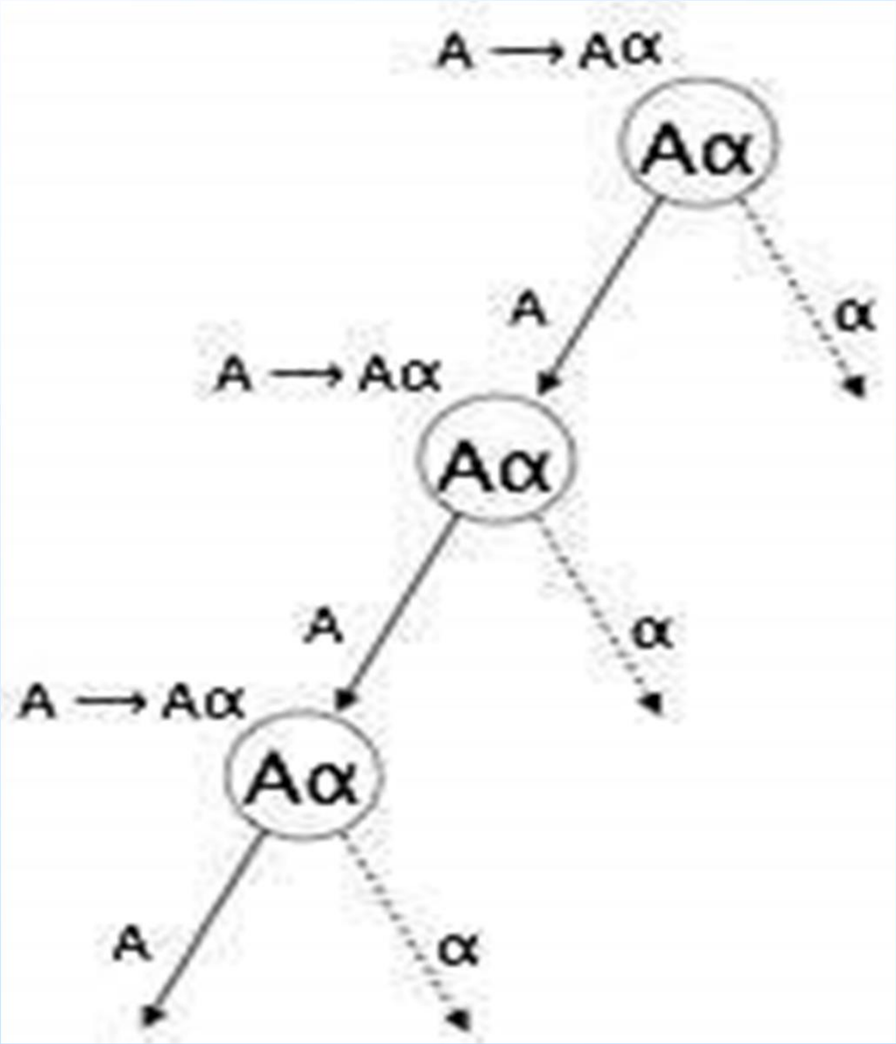
2. $S \Rightarrow A\alpha \mid \beta$

   $A \Rightarrow Sd$ indirect left recursion for S

- (1) is an example of immediate left recursion, where A is any non-terminal symbol and α represents a string of non-terminals.
- (2) is an example of indirect-left recursion.



Pictorial View of Derivation from left –recursive grammar



Removal of Left Recursion

- One way to remove left recursion is to use the following technique:
- The production $A \to A\alpha \mid \beta$ is converted into following productions
  - A -> βA'
  - A` -> αA' | ε

  Or without ε we can write grammar as:
  - A -> βA' |β
  - A’ -> αA' | α

Removal of Left Recursion

- In general, for grammar
- $A \rightarrow A\alpha_1 \mid A\alpha_2 \mid A\alpha_3 \mid \ldots \mid A\alpha_m \mid \beta_1 \mid \beta_{2\ldots} \mid \beta_n$ is converted into following productions
  - $A \rightarrow \beta_1 A' \mid \beta_2 A` \mid \ldots\ldots \mid \beta_n A'$
  - $A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \alpha_3 A' \mid \ldots\ldots \mid \alpha_1 A' \mid \varepsilon$

  Or without ε we can write grammar as:
  - $A \rightarrow \beta_1 A' \mid \beta_2 A` \mid \ldots\ldots \mid \beta_n A` \mid \beta_1 \mid \beta_2 \mid \ldots\ldots \mid \beta_n$
  - $A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \alpha_3 A' \mid \ldots\ldots \mid \alpha_m A' \mid \varepsilon\ \alpha_1 \mid \alpha_2 \mid \alpha_3 \mid \ldots\ldots \mid \alpha_m$

Exercise: Remove left recursion

1.

- **E -> E + T | T**
- **T -> T*F | F**
- **F -> (E) | a**

2. **S -> S+S | S-S| S*S| S/S | (S) | a**