Formal Languages & Autometa Theory

Lecture Material on
Context Free Grammars

PNRL Chandra Sekhar

Department of Computer Science & Engineering GST, GITAM

September 21, 2023

The following material will give you an overview of what you learn in the class

1 Introduction

Context-free languages (CFL) are a larger set of languages than regular languages. All regular languages are CFLs, whereas Not all CFLs are regular. CFLs are helpful in describing "nested" structures (e.g., expressions with properly balanced parentheses) that occur commonly in programming languages but are known to be not regular. CFLSs are fundamental because most programming languages are context-free.

Context-free languages are described by grammars known as context-free grammars (CFG). Different context-free grammars can generate the same context-free language. It is essential to distinguish the properties of the language (intrinsic properties) from the properties of a particular grammar (extrinsic properties).

1.1 Context-Free Grammars

In formal language theory, formal grammar is a set of production rules that describe all possible strings in a given formal language. Production rules are simple replacements. A context-free grammar is a formal grammar that describes a language drawn from a useful set called Context-free Languages. In particular, in a context-free grammar, each production rule is of the form: $A \to \alpha$ where A, a single non-terminal symbol, and α string of terminals and/or non-terminals (α can be empty).

Languages generated by context-free grammar are known as context-free languages (CFL). Different context-free grammars can generate the same context-free language. It is essential to distinguish the properties of the language (intrinsic properties) from the properties of a particular grammar (extrinsic properties).

1.2 Formal definition of CFG

A context-free grammar G is defined by the four-tuple (V, T, P, S) Where

- V is a finite set, and each element $\in V$ is called a non-terminal or variable. Each variable represents a different phrase or clause in the sentence or can be replaced with a string of variables and terminals. Variables are also sometimes called syntactic categories. Each variable defines a sub-language of the language defined by G.
- T is a finite set of terminals, disjoint from V, which make up the actual content of the sentence. The set of terminals is the alphabet of the language Σ defined by the grammar G.
- P is a finite set of productions that dictate how the variables get replaced to derive strings of the grammar. Also referred to as R, a finite relation in $VX(V \cup T)^*$.
- S $\in V$ referred to as starting variable, which is the start point of computation used to represent the whole sentence (or program) similar to the initial state of the automaton.

1.3 Examples

Languages to CFG

Problem 1

Find the equivalent grammar for the language L = $\{a^n b^n | n \geq 0\}$

Solution. Since the language L={ϵ,ab,aabb,aaabbb....}, the sequence ab is repeated so, the productions are:
S→aSb
S→ϵ
In order to check for a string ”aaabbb”
S→aSb
S→aaSbb {since S→aSb}
S→aaaSbbb {since S→aSb}
S→aaabbb {since S→ ϵ}

Problem 2

Find the equivalent grammar for the language L={w—w∈ $\{a, b\}^*$ and w is a palindrome}.

Solution. Since the language L = {aa,aba,bab,abba,baaab,...}, where the beginning and ending symbol must be same. So, the productions are:
S→aSa
S→bSb
S→a
S→b
S→ ϵ
In order to check for a string ”abba”
S→aSa
S→abSba {since S→bSb}
S→abba {since S→ ϵ}

Regular Expression to CFG

Problem 3

Find grammar for the r.e. $(011 + 1)^*(01)^*$

Solution. Let A = $(011 + 1)^*$ and B = $(01)^*$

S→AB
since $(011 + 1)^*$ repeats many times it leads to:
A→CA
A→ ϵ
C→ 011
C→ 1
since $(01)^*$ repeats many times it leads to:
B→DB
B→ ϵ
D→ 01
In simple, the resultant grammar can be written as:
S→ AB
A→ 011A | 1A |ϵ
B→ 01B |ϵ

Problem 4

Find the grammar for the language containing all the strings of different first and last symbols over {0,1}.

Solution. The r.e. for the above L are $0(0 + 1)^*1 + 1(0 + 1)^*0$. So, the productions are:
S→ 0A1 | 1A0
A→ 0A | 1A ϵ

2 Conventions

The following are the conventions that we follow during the course.

- Upper case letters A, B, C,... are denoted as variables. The variable is also called as non-terminal.
- Lowercase letters a, b, c,... are denoted as terminals.
- The Italic capital letters X, Y, and Z are denoted as either terminals or variables.
- The lowercase letters u,v, w are denoted as strings of terminals.
- The Greek symbols α, β, γ... are denoted as strings of terminals and/or variables.
- If there are multiple singleton productions A→ $\alpha 1$, A→ $\alpha 2$, A→ $\alpha 2$, ... A→ αn then they can be written as A→ $\alpha 1|\alpha 2|\alpha 3|...|\alpha n$

3 Derivation of CFG

Definition: Derivation is the process of defining the strings of a grammar by applying the production rules recursively from start symbol.

i.e., the "productions for A" have A on the left side of the →. We say $\alpha A\beta \implies \alpha\gamma\beta$ if $A \to \gamma$ is a production.

Steps for Deriving a string from G

a) The derivation begins with the start symbol.

b) Then choose the left-hand or right-hand side non-terminal or variable and expand it with the respective production.

c) Repeat the above step until no variables are in the expanded string.

d) If the expanded string contains all terminals, then such string is called a sentence (w) of the grammar; otherwise, if it is in the combination of variable and terminals, it is called sentential form.

3.1 Examples

Problem 5

From the grammar (S → aS | bS | a | b) derive the string abbab.

Solution. S→aSa
S→abS {since S→bS}
S→abbS {since S→bS }
S→abbaS {since S→aS }
S→abbab {since S→b }
Since the derived string abbab is the same as input string, the given string is accepted by the grammar and is called as the sentence of the grammar.

Problem 6

From the grammar (S → XaaX, X → aX | bX |ϵ) derive the string abbaaba.

Solution. S→XaaX {Start with start symbol}
S→aXaaX {since X→aX}
S→abXaaX {since X→bX}
S→abbXaaX {since X→aX}
S→abbaaX {since X→ ϵ}
S→abbaabX {since X→bX}
S→abbaabaX {since X→aX}
S→abbaaba {since X→ ϵ}

Since the derived string abbaaba is the same as the input string, the given string is accepted by the grammar and is called the sentence of the grammar.

3.2 Notations for Derivation

Define $\Longrightarrow$, $\overset{*}{\Longrightarrow}$ between $(V \cup T)^*$.

- if A$\rightarrow$ β is a production and α, γ are any string in $(V \cup T)^*$ then $\alpha A \gamma \Longrightarrow \alpha\beta\gamma$
  i.e., $\alpha A \gamma$ derives $\alpha\beta\gamma$ by using the production $A \rightarrow \beta$
  The relation $\Longrightarrow$ is used when one is obtained from another.
- Suppose $\alpha_1, \alpha_2, \alpha_3, ...\alpha_m (m \geq 1)$ are strings in $(V \cup T)^*$ and $\alpha_1 \Longrightarrow \alpha_2$, $\alpha_2 \Longrightarrow \alpha_3$ ... $\alpha_{m-1} \Longrightarrow \alpha_m$, then we say $\alpha_1 \overset{*}{\Longrightarrow} \alpha_m$ where α_1 derives α_m in G by applying 0 or more productions.
- $\overset{*}{\Longrightarrow}$ is reflexive and transitive closure of $\Longrightarrow$

3.3 Types of Derivations

The derivations of two types are based on the kind of variable chosen to expand the string.

- Left-most derivation - where the left-most variable is always used to expand the string during derivation.
- Right-most derivation - where the right-most variable is always used to expand the string during derivation.

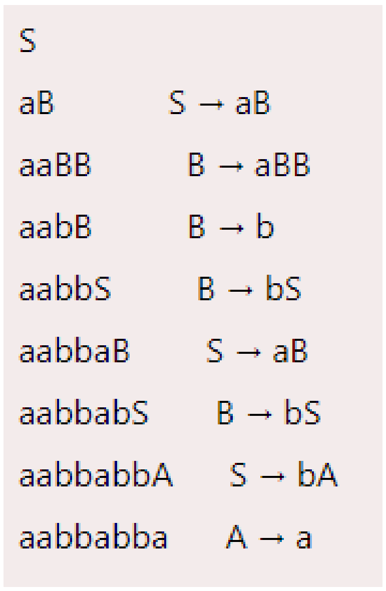
Example:

Problem 7

Derive the string "aabbabba" for leftmost derivation and rightmost derivation using the given grammar G $\{S \rightarrow aB|bA, S \rightarrow a|aS|bAA, S \rightarrow b|aS|aBB\}$.

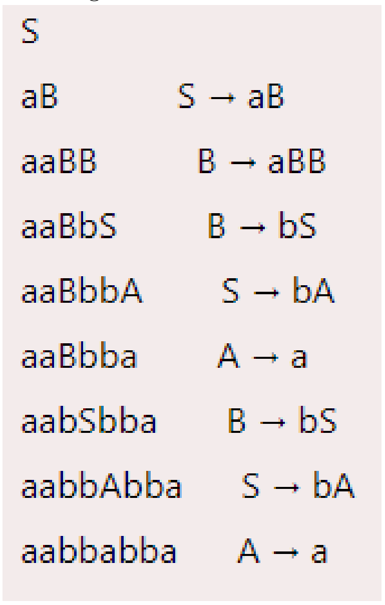
Solution.

Left-Most Derivation



S	
aB	S → aB
aaBB	B → aBB
aabB	B → b
aabbS	B → bS
aabbaB	S → aB
aabbabS	B → bS
aabbabbA	S → bA
aabbabba	A → a

Right-Most Derivation



S	
aB	S → aB
aaBB	B → aBB
aaBbS	B → bS
aaBbbA	S → bA
aaBbba	A → a
aabSbba	B → bS
aabbAbba	S → bA
aabbabba	A → a

3.4 Derivation or Parse Tree

Definition: The pictorial representation of the derivation of a string from a given grammar G is known as derivation tree also known as parse tree.

A Parse Tree for a CFG G =(V,T,P,S) is a tree satisfying the following conditions

- The Root of a node has the label S, where S is the start symbol.
- Each node of the parse tree has a label, which can be a variable (V), terminal (T), or ϵ.
- If $A \to C_1, C_2 \ldots\ldots C_n$ is a production, then $C_1, C_2 \ldots\ldots C_n$ are the children of node labeled A.
- Leaf Nodes are terminal (T), and Interior or internal nodes are variables (V).
- If a node A has k children with labels $A_1, A_2 \ldots\ldots A_k$,then $A \to A_1 A_2 \ldots\ldots A_k$ will be the production in context-free grammar G.

The **yield** of the Derivation Tree is the concatenation of labels of the leaves in left-to-right ordering.

Example:

Problem 8

For the given grammar G ({ $S \to bB|aA, A \to b|bS|aAA, B \to a|aS|bBB$}) find the left-most and right-most derivation and obtain the respective derivation or parse trees.

Solution.

Left-most derivation

$S \Rightarrow b\ \underline{B}$

$\Rightarrow bb\ \underline{B}B$

$\Rightarrow bba\underline{B}$

$\Rightarrow bbaa\underline{S}$

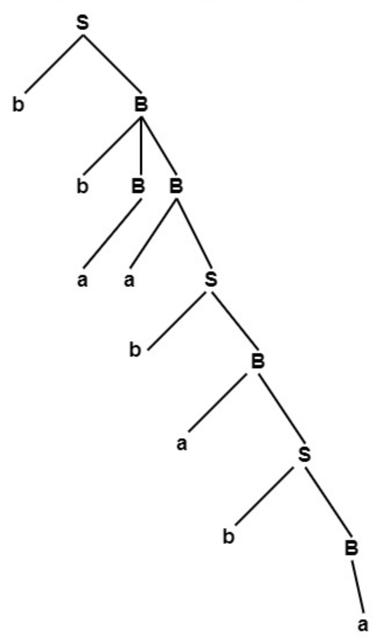
$\Rightarrow bb\ aa\underline{bB}$

$\Rightarrow bb\ aa\ b\ \underline{aS}$

$\Rightarrow bb\ aa\ bab\ \underline{B}$

$\Rightarrow bb\ aa\ ba\ ba$

Derivation Tree



Right-most derivation

$S \Rightarrow b\underline{B}$

$\Rightarrow bb\ B\underline{B}$

$\Rightarrow bbBa\underline{S}$

$\Rightarrow bbBab\underline{B}$

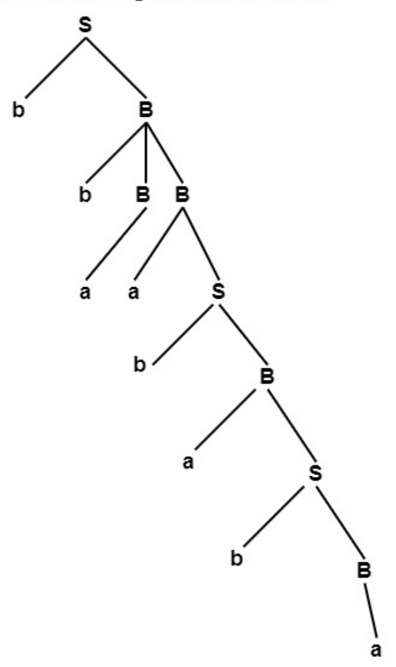
$\Rightarrow bbBaba\underline{S}$

$\Rightarrow bbBabab\underline{B}$

$\Rightarrow bb\underline{B}abab\ a$

$\Rightarrow bbaababa$

Derivation Tree



3.5 Language generated by Grammar

The language generated by a CFG G(V,T,P,S) is the set of all strings that are derived from the given grammar denoted by L(G).

L(G) = {w $|w \in T^*$ and S $\overset{*}{\Longrightarrow}$ w }

3.6 Examples

Problem 9

From the given CFG G({S,C},{a,b},{S→aCa, C→aCa/b},S) obtain the language.

Solution. S $\Longrightarrow$ aCa $\Longrightarrow$ aaCaa $\Longrightarrow$ ..... $\Longrightarrow$ aaa....Cbbb.... $\Longrightarrow$ aaa....nbbb.... (for n times)

So, the language L(G)={aba,aabaa,aaabaaa,... } i.e., {w $|w \in a^nba^n$.

Problem 10

Obtain the language from the grammar G ({S},{a,b},{S$toaS|bS|a|b$},S).

Solution. i) S $\Longrightarrow$ aS $\Longrightarrow$ abS $\Longrightarrow$ abaS....

ii)S $\Longrightarrow$ bS $\Longrightarrow$ baS $\Longrightarrow$ babS...

So, the language L(G) = {a,b,ab,ba,aba,bab,bbb,baa....} = $\{a, b\}^*$

4 Regular Grammar - Regular Languages

Regular grammar is a Type-3 grammar that generates regular language. In this grammar, each production has a single non-terminal on the left-hand side and the right-hand side consisting of a single terminal or single terminal followed by a non-terminal or vice versa.

i.e, The productions are of the form:

A → xB

A → x

A → Bx

where A, B ∈ V and x ∈ T*.

4.1 Types of Regular grammar

- Left Linear Grammar (LLG): is a regular grammar in which the productions are of form

  A → x

  A → Bx where A, B ∈ V and x ∈ T*

- Right Linear Grammar (RLG): is a regular grammar in which the productions are of form

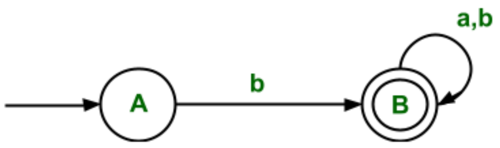
A $\rightarrow$ x
A $\rightarrow$ xB where A, B $\in$ V and x $\in$ T*

The language generated by type-3/regular grammar is a regular language for which a FA can be obtained, and from a given FA one can obtain the corresponding type-3/regular grammar.

Example:

Problem 11

The given FA, which accepts strings that start with b, obtains the corresponding regular grammar and language.



Solution. The regular grammar corresponding to FA is {A $\rightarrow$ bB, B$\rightarrow$ $aB|bB|\epsilon$}

The obtained grammar is RLG, which can be written directly through FA.

The regular language corresponding to RLG is L= {b, ba, bb, baa, bab ,bba,bbb ..... }

4.2 Regular Grammars versus Finite Automata

For a given right linear grammar(RLG) G there exist a language L(G), which is accepted by a finite automata (FA).

For any given finite automata M there exist a right linear grammar G such that L(M)=L(G).

4.2.1 RLG to FA

Method

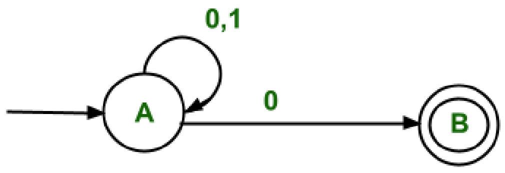
- Start from the first production and make the left side variable as initial state.
- For every production of the form A$\rightarrow$aB, have the transition $\delta(A, a) = B$.
- Those states which end up with terminals without further non-terminals, make them as final states.

Example:

Problem 12

The RLG for language L, represents a set of all strings which end with 0. as A $\rightarrow$ $0A|1A|0B$, B $\rightarrow$ ϵ.

Solution. The NFA corresponding to RLG can be depicted as:



Problem 13

Construct Finite automata to accept the language generated by the following grammar G({S→ $aA|B, A \rightarrow aaB, B \rightarrow bB|a$}).

Problem 14

Construct Finite automata to accept the language generated by the following grammar G({S→ $0S|1A|1, A \rightarrow 0A|0S|0|1$}).

Problem 15

Construct Finite automata to accept the language generated by the following grammar G({A→ $aA|bB|\epsilon, B \rightarrow bB|\epsilon$}).

4.2.2 FA to RLG

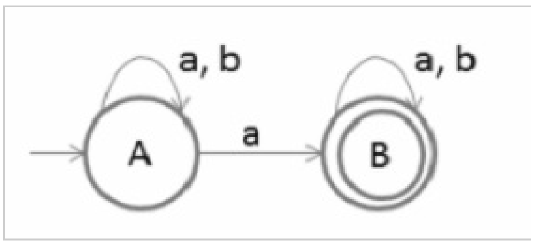
Method

- Begin from the initial state.
- On each state and for each transition $\delta(A, a) = B$ then write the production as A→aB.
- For the final state, add the production $A \rightarrow \epsilon$

Example

Problem 16

Obtain the RLG for the following FA.



Solution. For the State A, the productions as follows: $A \to aA|bA|aB$.

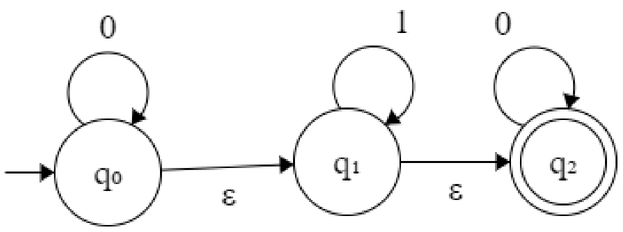
For the State B, the productions as follows: $A \to aB|bB|\epsilon$.

The resultant right linear grammar for the given finite automata is

G($\{A \to aA|bA|aB$,$A \to aB|bB|\epsilon\}$)

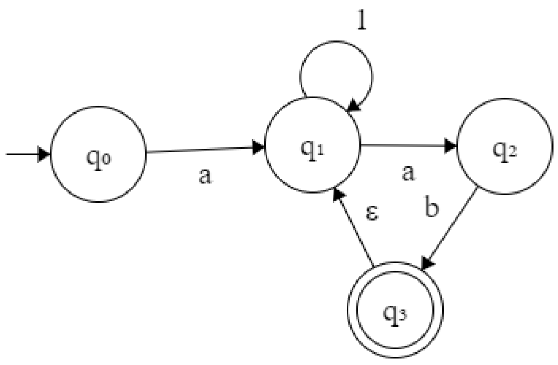
Problem 17

Obtain RGL from the following FA.



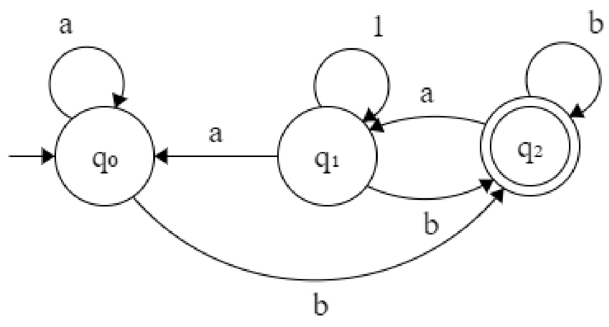
Problem 18

Obtain RGL from the following FA.



Problem 19

Obtain RGL from the following FA.



4.2.3 RLG to RE

For a given RLG G, a regular expression R that specifies L(G) can be directly obtained as follows:

- For each production replace the → as "=" to obtain a set of equations.
- For each equation of the form A=aA—b as A=a*b.
- Solve the remaining set of equations and obtain equation interms of start symbol S.
- The result is the regular expression specifying L(G).

Example

Problem 20

Obtain the regular expression for the given RLG G({ S→ $01B|0, B \rightarrow 1B|11\}$).

Solution. S→ $01B|0 \implies S = 01B|0$ —-(1)
$B \rightarrow 1B|11 \implies B = 1B|11$ —-(2)
(2) $\implies$ B=1*11 —-(3)
From (3) and (1) $\implies S = 011 * 11|0$ or (011*11+0).
The resultant regular expression for the RLG is (011*11+0).

Problem 21

Obtain the Regular expression for the given grammar G({S→ $baS|aA, A \rightarrow bbA|bb\}$).

Problem 22

Obtain the Regular expression for the given grammar G($\{S \rightarrow 0A|0|1B, A \rightarrow 1A|1, B \rightarrow 0B|1S\}$).

5 Ambiguous Grammars

A Grammar G is said to be ambiguous if there exist two or more left most derivations or right most derivations or two different parse trees for the given input string.

Example

Problem 23

The arithmetic grammar G({E→E+E|E*E|id}) generates two different LMD's for the string id+id*id.

Solution.

LMD 1: E $\Longrightarrow$ E+E $\Longrightarrow$ id+E $\Longrightarrow$ id+E*E $\Longrightarrow$ id+id*E $\Longrightarrow$ id+id*id.

LMD 2: E $\Longrightarrow$ E*E $\Longrightarrow$ E+E*E $\Longrightarrow$ id+E*E $\Longrightarrow$ id+id*E $\Longrightarrow$ id+id*id.

RMD 1: E $\Longrightarrow$ E+E $\Longrightarrow$ E+E*E $\Longrightarrow$ E+E*id $\Longrightarrow$ E+id*id $\Longrightarrow$ id+id*id.

RMD 1: E $\Longrightarrow$ E*E $\Longrightarrow$ E*id $\Longrightarrow$ E+E*id $\Longrightarrow$ E+id*id $\Longrightarrow$ id+id*id.

For the input string id+id*id, two different LMD's and RMD's exist, hence the grammar is said to be ambiguous.

Problem 24

Show that the grammar G ({S→aSbS — bSbS —ϵ}) is ambiguous with the string "abab".

Problem 25

Show that the grammar G({S→AB, B→ ab | aa | a, B→b}) is ambiguous with the string "aab".

Problem 26

Check the grammar G({S→ S+S |SS|(S)|S*|a}) is ambiguous or not with the string "(a+a)*a".

6 Simplification of CFG

The context-free grammar can effectively represent a variety of languages. Because not all grammar is streamlined, some grammars may contain superfluous symbols (non-terminal), some of the productions are not useful and are redundant. This is due to definition of CFGs does not restrict from making these redundant productions or symbols. These redundant productions lengthens the grammar

By simplifying CFGs we remove all these redundant productions from a grammar, while keeping the transformed grammar equivalent to the original grammar. Two grammars are called equivalent if they produce the same language. Simplifying CFGs is necessary to later convert them into Normal forms.

Types of redundant productions and symbols are:

a) Useless productions or variables (symbols).

b) Null productions.

c) Unit Productions.

6.1 Useless productions or variables

A production is said to be useless if it cannot take part in derivation of any string or does not derive a terminal. In other words, a Variable or symbol is useless if it can never take part in derivation of any string or if it cannot be reachable from start symbol.

i.e., if a variable is fails to exist in the derivation S $\overset{*}{\Longrightarrow}$ w, then the variable is said to be useless and can be removed from V.

6.1.1 Detecting variables that does not take part in derivation

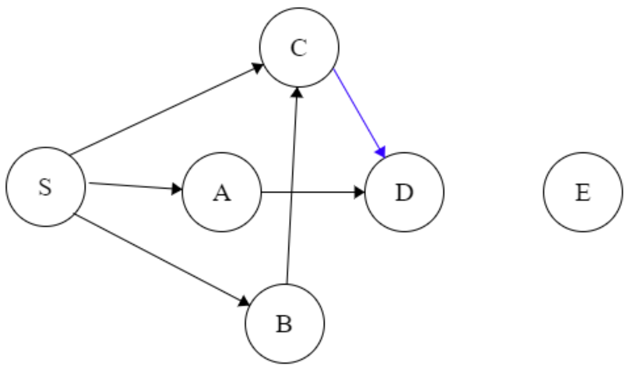
One way to find the useless variables which are not take part in derivation is, draw a dependency graph for all variables and which is not connected from the start symbol, consider it as useless and remove from the grammar.

Example:

Problem 27

Eliminate useless symbols and productions from the following grammar: G($\{S \rightarrow AB|AC, A \rightarrow aAa|dDca, B \rightarrow cC|b, C \rightarrow CCD|c, D \rightarrow dd, E \rightarrow a\}$).

Solution. Draw a dependency graph over the variables in the grammar as shown in figure.



From the above figure, it is evident that the Variable E is unreachable from S. It is uselsess and remove from the grammar.

The resultant grammar is: G($\{S \rightarrow AB|AC, A \rightarrow aAa|dDca, B \rightarrow cC|b, C \rightarrow CCD|c, D \rightarrow dd\}$).

6.1.2 Detecting Variables that fail to generate Terminals

In order to identify the variable which derives a terminal or not, we need to check systematically. The process begins with those variables which directly derive variables and subsequently identify those variables that derive these variables.

The process of Identification is as follows:

- Let W_1 be the set of variables which directly derive terminals in G. i.e., $W_1 \subseteq V$ such that for each $X \in W_1, X \implies \Sigma^*$.
- Extend the set W_i as W_{i+1} as $W_{i+1} = W_i \cup \{$All the variables that derive the variables in W_i or the terminals$\}$.
- repeat above step until $W_i = W_{i+1}$
- The variables V - W_i as W_i are considered as usesless and remove them from G.

Examples

Problem 28

Eliminate useless symbols and productions from the following grammar: G($\{S \rightarrow AB|CA|BC, A \rightarrow a, C \rightarrow aB|b\}$).

Solution. W_1={A,C}

$W_2 = W_1 \cup \{S\}$ = {S,A,C} (since S→CA)

$W_3 = W_2 \cup \{\}$ = {S,A,C} (since no more variable which derives the variable S, A, C)

Iteration stops since $W_3 = W_2$ and W = V-W_3 = {S,A,B,C}-{S,A,C} = {B} which does not derive terminals and consider them as useless and remove from the grammar G.

The resultant Grammar is: G($\{S \rightarrow CA, A \rightarrow a, C \rightarrow b\}$)

Problem 29

Eliminate useless symbols and productions from the following grammar: G($\{S \rightarrow AB, A \rightarrow aA|a, B \rightarrow bB|cC|b, C \rightarrow CC, D \rightarrow d, E \rightarrow a\}$).

Solution. In the grammar the productions defined on variables D and E are not used or can't reach from start symbol. So, both variables D and E are useless and remove those productions from the grammar.

Also, C can't derive terminal so, it can't take part in any derivation of string of the grammar. So, it is useless and remove them.

The resultant Grammar is: G($\{S \rightarrow AB, A \rightarrow aA|a, B \rightarrow bB|b\}$)

Problem 30

Eliminate useless symbols and productions from the following grammar: G($\{S \rightarrow aAa, A \rightarrow bBB, B \rightarrow ab, C \rightarrow ab\}$).

6.2 Null Productions or Nullable Variables

The production of the form $A \rightarrow \epsilon$ is called Null production. These productions can only be removed from those grammars that do not generate ϵ (an empty string). It is possible for a grammar to contain null productions and yet not produce an empty string.

In a grammar G, if there is a variable X such that X $\overset{*}{\Longrightarrow}$ ϵ, then the variable X is said to be Nullable. A nullable variable decreases the length of its output string.

A CFG is a monotonically increasing grammar where each production either increases the length of the string or keeps it constant. Hence, null prodctions or nullable variables are not allowed in it.

Yet there is an exception. For example in the language L = $\{a^n b^n | n \geq 0$, ϵ is a valid string. The only way ϵ can be included in a CFL L by including the production $S \rightarrow \epsilon$ in the grammar G of L.

The process of removing null productions or nullable is as follows:

- Construct the set $W_1 = \{X | X \rightarrow \epsilon$ is a null production in G$\}$.
- Extend the set W_i to W_{i+1} as follows:

  $W_{i+1} = W_i \cup \{X \in V | X \rightarrow \alpha$ such that α contains a member of $W_i\}$.
- repeat above step until $W_i = W_{i+1}$
- The W_i contains nullable variables and can be carefully used to reconstruct the grammar G.
- Re-construction of modified grammar as follows:

  If the grammar contains productions of the form $S \rightarrow AB|BC, A \rightarrow a, B \rightarrow \epsilon, C \rightarrow \epsilon$ then S, B, C are nullable variables. Since B and C are null production and can be removed from G. The production $S \rightarrow BC$ can be removed from grammar as both B and C are nullable, whereas A is not nullable, $S \rightarrow AB$ can be written as $S \rightarrow A$. The resultant productions are $S \rightarrow A, A \rightarrow a$.

Examples

Problem 31

Eliminate null productions and nullable variable from the following grammar: G($\{S \rightarrow aS|AC|ABA \rightarrow \epsilon, B \rightarrow bB|bb, C \rightarrow \epsilon, D \rightarrow c\}$).

Solution. W_1=$\{$A,C$\}$ (since both are null productions)

$W_2 = W_1 \cup \{S\}$ = {S,A,C} (since S→AC and becomes nullable)

$W_3 = W_2 \cup \{\}$ = {S,A,C} (since no more variable can be nullable)

Iteration stops since $W_3 = W_2$ and W_3 = {S,A,C} = are nullable and reconstruct the grammar G as follows:

$S \rightarrow aS|B|a|\epsilon$ (since S, A and C are nullable)

The resultant Grammar is: G($\{S \rightarrow aS|B|a|\epsilon, B \rightarrow bB|bb, D \rightarrow c\}$)

Problem 32

Eliminate null productions and nullable variable from the following grammar: G($\{S \rightarrow AAA|B, A \rightarrow 0A|B, B \rightarrow \epsilon\}$).

Problem 33

Eliminate null productions and nullable variable from the following grammar: G($\{S \rightarrow ABaC, A \rightarrow BC, B \rightarrow b|\epsilon, C \rightarrow D|E, D \rightarrow d\}$).

Problem 34

Eliminate null productions and nullable variable from the following grammar: G($\{S \rightarrow aA, A \rightarrow BB, B \rightarrow aBb|\epsilon\}$).

6.3 Unit Productions

In a CFG G(V,T,P,S) the set V contains a finite set of variables, each of which is supposed to represent a different syntactic category. In addition, each variable should have its own productions to generate its own strings of terminals. If two or more variables in V are always generating the same set of strings, then it is desirable to keep one of them in V and discard the rest. The unit productions in the CFG is an indicative of such problem.

A production of the form $X \rightarrow Y$ in a CFG is known as unit production. Let 'w' be some string such that Y $\overset{*}{\Longrightarrow}$ w.

Then, we also have X $\Longrightarrow$ Y $\overset{*}{\Longrightarrow}$ w or X $\overset{*}{\Longrightarrow}$ w . Wherever Y derives a string the same can also be derived from X. Therefore, it is advisable to replace all occurrences of Y from the production set by X and remove Y from the set of variables.

6.3.1 Identification and removal of Unit productions

- In the given G if a production of the form $A \rightarrow B$ found, add production $A \rightarrow x$ to the grammar whenever $B \rightarrow x$ occurs in the grammar. (x $\in Tor\epsilon$)
- Delete the production A → B from the grammar.
- Repeat the above step until all unit productions are removed.

Examples

Problem 35

Obtain the grammar after removal of unit productions from the given grammar G($\{S \rightarrow 0X|1Y|Z, X \rightarrow 0S|00, Y \rightarrow 1|X, Z \rightarrow 01\}$)

Solution. $S \rightarrow Z$ is a unit production. Since $Z \rightarrow 01$ is a production in G that gives a terminal, replace Z with 01 in the first production.

i.e., $S \rightarrow 0X|1Y|01$

Similarly, Y $\rightarrow$ X is also a unit production and modify it as $B \rightarrow 1|0S|00$

The resultant CFG is : G($\{S \rightarrow 0X|1Y|01, X \rightarrow 0S|00, Y \rightarrow 1|0S|00, Z \rightarrow 01\}$).

Problem 36

Obtain the grammar after removal of unit productions from the given grammar G($\{S \rightarrow ABC|ADE, A \rightarrow a|D, B \rightarrow b, C \rightarrow D, D \rightarrow E, E \rightarrow c\}$)

Problem 37

Obtain the grammar after removal of unit productions from the given grammar G($\{S \rightarrow ABCD, A \rightarrow a, B \rightarrow C|b, C \rightarrow D, D \rightarrow c\}$)

Example 2:

6.4 Simplify the CFG

To simplify the CFG, one has to remove the redundant productions and variables in the following order:

- Eliminate null productions, nullable variables.
- Eliminate unit productions,.
- Eliminate useless symbols (unreachable, not derive terminals)

Examples

Problem 38

Simplify the following grammar G($\{S \rightarrow aA|aBB, A \rightarrow aAA|\epsilon, B \rightarrow bB|bbC, C \rightarrow B\}$).

Solution. Using the above mentioned procedures, the following is the direct answers to obtain the resultant grammar.

1. **Elimination of nullable variables:**

$A \to \epsilon$ is null production, the nullable variables are A.

The resultant grammar is

G($\{S \to aA|a|aBB, A \to aAA|aA|a, B \to bB|bbC, C \to B\}$).

2. **Elimination of unit productions:**

$C \to B$ is the unit production and replace B with its definition.

The resultant grammar is

G($\{S \to aA|a|aBB, A \to aAA|aA|a, B \to bB|bbC, C \to bB|bbC\}$).

3. **Elimination of useless symbols:**

Since B and C does not derive terminals, they are useless symbols and remove them from the grammar.

The resultant grammar after simplification is G($\{S \to aA|a, A \to aAA|aA|a\}$).

Problem 39

Simplify the following grammar G($\{S \to Aa|bS|\epsilon, A \to aA|bB|\epsilon, B \to aA|bc|\epsilon, C \to aC|bc\}$).

Problem 40

Simplify the following grammar G($\{S \to ABaC, A \to BC, B \to b|\epsilon, C \to D|\epsilon, D \to d\}$).

7 Normal Forms

In a CFG G(V,T,P,S) all the productions are of the form $VX(V \cup T)^*$. For different application on the productions, sometimes it is necessary to restrict the production into certain form. Then such G is said to be in normal form.

There are two such normal forms which are most widely used are:

- Chomsky Normal Form (CNF)
- Greibach Normal Form (GNF)

7.1 Chomsky Normal Form

Definition: A CFG G is said to be in CNF if and only if it is free from useless symbols, ϵ-productions(except Start symbol), unit productions and all the productions are of the form $A \to BC$ (Two variables) or $A \to a$ (only a terminal).

Example: Consider the following grammars:

- G1 = $\{S \to a, S \to AZ, A \to a, Z \to z\}$ is in CNF

- G2 = {S$\to a, S \to aZ, Z \to a\} is not in CNF becase of aZ$

Points to remember:

For a grammar G, there can be more than one CNF's which are equivalent to G.

CNF is used as a pre-processing step for many algorithms such as e CYK(membership algorithm), bottom-up parsers etc.

In CNF, for deriving a string w of length n, it requires 2n-1 steps.

Any Context free Grammar that do not have ϵ in it's language has an equivalent CNF.

7.1.1 Method to convert CFG into CNF

- Simplify the grammar G by eliminating ϵ-productions, unit productions and useless symbols such that the rhs of production contains either single T or whole lenth is atleast 2 ($ge2$).
- Elimination of terminals (T's) on RHS
  - If the production of the form $A \rightarrow a$, then no need to change as it is in CNF.
  - If a terminal exits with other terminals or non-terminal then replace the terminal with a new variable. For ex. a production rule $X \rightarrow xY$ can be re-written as: $X \rightarrow ZY, Z \rightarrow x$ to bring back in to CNF.
- Elimination of variables on RHS:
  - If the production of the form $A \rightarrow BC$, then no need to change as it is in CNF.
  - if the production is of the form $A \rightarrow BCD$ whose lenth of RHS is $ge3$ can be decomposed as: $A \rightarrow PD, P \rightarrow BC$ to bring back it into CNF.

Problem 41

Convert the Following CFG G($\{S \rightarrow ASB, A \rightarrow aAS|a|\epsilon, B \rightarrow SbS|A|bb\}$)into CNF

Solution.

a) Simplify the Grammar. The resultant grammar after simplification is G($\{S \rightarrow AS|ASB|SB, A \rightarrow aAS|aS|a, B \rightarrow SbS|bb|aAS|aS|a\}$)

b) In production rule $A \rightarrow aAS|aS, B \rightarrow SbS|aAS|aS$ terminals a and b exist on RHS with other non-terminates.

The resultant grammar after removing them from RHS is:

G($\{S \rightarrow AS|ASB|SB, A \rightarrow XAS|XS|a, B \rightarrow SYS|VV|XAS|XS|a, X \rightarrow a, Y \rightarrow b, V \rightarrow b\}$)

c) In the productions, $S \rightarrow ASB, A \rightarrow XAS, B \rightarrow SYS, B \rightarrow XAX$ has more than two symbols.

Removing them from grammar by decompose.

The resultant grammar after decompose is:

G($\{S \rightarrow AS|PB|SB, A \rightarrow QS|XS|a, B \rightarrow RS|VV|TS|XS|a, X \rightarrow a, Y \rightarrow b, V \rightarrow b, P\text{ß}AS, Q\text{ß}XA, R\text{ß}SY, T\text{ß}XA\}$)

Problem 42

1. Convert the Following CFG G($\{S \to AB|aB, A \to aaB|\epsilon, B \to bbA\}$)into CNF.
2. Convert the Following CFG G($\{S \to ASB|\epsilon, A \to aAS|a, B \to SbS|A|bb\}$)into CNF.

7.2 Greibach Normal Form (GNF)

Definition: A CFG is said to be in GNF if all production are of the form $A \to a\alpha$ where $a \in T$ and $\alpha \in V^*$(i.e., a terminal followed by string of variables possibly empty string). In GNF there is no restriction on the length of RHS except the way the terminal and non-terminals present.

When a CFG G generates a language not containing ϵ then, we can find an equivalent grammar in GNF.

Consider the following grammars:

– G1 = $\{S \to aA|bB, B \to bB|b, A \to aA|a\}$ as the production are of the form $A \to a\alpha$ the grammar is in CNF (lenth wise) as well as in GNF.

– G2 = $\{S \to aA|bB, B \to bB|\epsilon, A \to aA|\epsilon\}$ as the productions contain ϵproduction, it is not in GNF. To make the grammar into GNF, first remove the unit and null production and then convert it into GNF.

To convert CFG into GNF, we need to know 2 lemmas:

– **Lemma1 (Substitution Rule):** Let G(V,T,P,S) be a CFG. Let $A \to B\alpha$ be a production in P and B is defined as $B \to \beta_1|\beta_2|...|\beta_m$, then the equivalent grammar can be obtained by substituting B in A.

The resulting grammar is: $A \to \beta_1\alpha|\beta_2\alpha|....\beta_m\alpha$

– **Lemma2 (Elimination of Left Recursion):** If the Grammar G contains a production of the form $A \to A\alpha|\beta$ then the grammar contains left recursion. (variable of LHS of production is equal to first symbol on the RHS of production).

To eliminate left recursion in the grammar, replace the production $A \to A\alpha|\beta$ by $\{A \to \beta A', A' \to \alpha A'|\epsilon\}$ or $\{A \to \beta A'|\beta, A' \to \alpha A'|\alpha\ \}$ (after removing ϵ production).

We can generate the grammar if the CFG has a production of the form

$A \to A\alpha_1|A\alpha_2|....|A\alpha_m|\beta_1|\beta_2|....|\beta_n$.

The equivalent grammar without left recursion as:

$A \to \beta_1 A'|\beta_2 A'|.....|\beta_n A'|\beta_1|\beta_2|.....|\beta_n$,

$A' \to \alpha_1 A'|\alpha_2 A'|....|\alpha_m A'|\alpha_1|\alpha_2|....|\alpha_m$.

7.2.1 Method to convert CFG into GNF

- Convert the grammar into CNF if it is not already there in CNF
- Rename all the variables as $A_1, A_2, ...A_n withS = A_1$
- For each production of the form $A_i \to A_j\alpha$, apply the following rules
  a) if $j > i$ - leave the production as it is in GNF.
  b) if $j = i$ - apply lemma2 (remove left recursion in the production.
  c) if $j < i$ - apply lemma1 (substitution rule)
- for each production of the form $A_i \to A_j\alpha$, with $j > i$, apply lemma1- substitution rule to bring A_i in to GNF if A_j is already in GNF.

7.2.2 Examples

Problem 43

Obtain the equivalent grammar in GNF from the given grammar G($\{S \to XA|BB, B \to b|SB, X \to b, A \to a\}$)

Solution.

a) Since the grammar is already is in CNF, there is no need of bringing into CNF.

b) Arrange all the variables in an order $\Longrightarrow$ {S, A, B, X} as $\{A_1, A_2, A_3, A_4\}$.

c) The new productions are:
$A_1 \to A_2A_3|A_4A_4 - - - (1)$,
$A_2 \to b, - - (2)$,
$A_3 \to a, - - (3)$,
$A_4 \to b|A_1A_4$ –(4).

d) From the production (4), apply substitution rule in (1) as:
$A_4 \to b|A_2A_3A_4|A_4A_4A_4$ –(5)

e) (5) contains left recursion and by eliminating them as:
$A_4 \to bZ|A_2A_3A_4Z|b|A_2A_3A_4 - -(6)Z \to A_4A_4Z|A_4A_4$ —(7)

f) from eq. (6 and 2) apply substitution rule to bring A_4 into GNf as:
$A_4 \to bZ|bA_3A_4Z|b|bA_3A_4$ –(8)

g) using eq. (8,2) bring (1) into GNF as:
$A_1 \to bA_3|bZA_4|bA_3A_4ZA_4|bA_4|bA_3A_4A_4$ —(9),

h) eq. (8) bring (7) into GNF as:
$Z \to bZA_4Z|bA_3A_4ZA_4Z|bA_4Z|bA_3A_4A_4Z|A_4A_4$ —(10)

i) the resultant grammar in GNF as:
$A_1 \to bA_3|bZA_4|bA_3A_4ZA_4|bA_4|bA_3A_4A_4$,
$A_2 \to b$,

$A_3 \rightarrow a,$
$A_4 \rightarrow bZ|bA_3A_4Z|b|bA_3A_4,$
$Z \rightarrow bZA_4Z|bA_3A_4ZA_4Z|bA_4Z|bA_3A_4A_4Z|A_4A_4$

Problem 44

Conver the following CFG into GNF G($\{S \rightarrow AA|a, A \rightarrow SS|b\}$).

Problem 45

Conver the following CFG into GNF G($\{S \rightarrow ABA, A \rightarrow AA|\epsilon, B \rightarrow bB|\epsilon\}$).

Problem 46

Conver the following CFG into GNF G($\{S \rightarrow XA|BB, B \rightarrow b|SB, X \rightarrow b, A \rightarrow a\}$).

8 Pumping Lemma for Context-Free Languages

In regular languages, if a string is given whose length is much greater than the number of states in a finite automaton, then a sub-string will be repeated many times to be part of the regular set/language.

Similarly, in CFL, for a string, there are always two short sub-strings that are close to each other and can be repeated/pumped as many times as required to be part of the context-free language.

The pumping lemma theorem is used as a tool to prove certain languages are not context-free.

Theorem: Let there be a CFL L, then a positive integer 'n' such that for any string $w \in L$, and $|w| \geq n$, w can be decomposed into $w = uvxyz$ such that

a) $|uv| \geq 1$
b) $|vwx| \leq n$ and
c) for any positive integer i, $uv^ixw^iy \in L$.

Problem 47

Show the the L=$\{0^n1^n2^n|n > 0\} is not CFL.$

Solution. Assume that L = {012,00112,000111222,....} is a CFL.

let $w = 001122 \in L$ where $|w| \geq 3$ (n=3 a positive integer).

Since L is context-free, by Pumping Lemma, the string w can be decomposed in to uvwxy as u=0, v=0, w=1, x=1, y=22 such that

- $|vx| = 2 \geq 1$
- $|vwx| = 3 \leq 3$
- a positive integer i=2, $uv^2wx^2y = 00^2 11^2 22 = 0^3 1^3 2^2$

  The obtained string $0^3 1^3 2^2 \notin L$

  Hence, our assumption that L is CFL is wrong, so L is not a CFL.

Problem 48

Show the the L=$\{a^p | p$ is prime$\}$ is not CFL.

Solution. Assume that L = {aa,aaa,aaaaa,aaaaaaa,....} is a CFL.

let m be a positive integer.

let p be a prime no such that $p \geq m$ and $w = a^p \in L$

let $w = uvwxy$ and $|w| = p$.

since $w = uvwxy \in L$ by pumping lemma $uv^k wx^k y \in L$

$|uv^k wx^k y| = |uvwxy| + (k-1)|vx| = p + (k-1)|vx|$

let $|vx| = r$ then $|uv^k wx^k y| = p + (k-1)r$

let k=p+1 then $|uv^k wx^k| = p + pr = p(1+r)$ which is a composite no, not prime no.

For the given CFL the string $w = uvwxy \in L$ but $uv^k wx^k \notin L$. So, L is not a CFL.

9 Closure Properties of CFL

A Context Free Language (CFL) is a language produced by a Context Free Grammar CFG G(V,T,P,S).

As Regular languages, context Free Languages are closed under the following basic operations:

- Union
- Concatenation
- Kleene Closure

9.1 Union

Theorem: If L_1 and L_2 are two CFLs, then $L_1 \cup L_2$ will also be Context Free.

Proof: if L_1, L_2 are two CFLs obtained from the respective CFGs $G_1(V_1, T, P_1, S_1), G_2(V_2, T, P_2, S_2)$ as $L_1 = L(G_1), L_2 = L(G_2)$

Define $L_1 \cup L_2 \implies L(G_1) \cup L(G_2) \implies L(G_1 \cup G_2)$

Now, define $G_1 \cup G_2 = G(S \cup V_1 \cup V_2, T, S \rightarrow S_1 | S_2 \cup P_1 \cup P_2, S)$ which is a CFG. So, $L(G_1 \cup G_2)$ is a CFL.

So, $L_1 \cup L_2 = L(G_1 \cup G_2)$ is also a CFL.

9.2 Concatenation

Theorem: If L_1 and L_2 are two CFLs, then $L_1.L_2$ will also be Context Free.

Proof: if L_1, L_2 are two CFLs obtained from the respective CFGs $G_1(V_1, T, P_1, S_1), G_2(V_2, T, P_2, S_2)$ as $L_1 = L(G_1), L_2 = L(G_2)$

Define $L_1.L_2 \implies L(G_1).L(G_2) \implies L(G_1.G_2)$

Now, define $G_1.G_2 = G(S \cup V_1 \cup V_2, T, S \rightarrow S_1.S_2 \cup P_1 \cup P_2, S)$ which is a CFG. So, $L(G_1.G_2)$ is a CFL.

So, $L_1.L_2 = L(G_1.G_2)$ is also a CFL.

9.3 Kleen Closure

Theorem: If L_1 is a CFL, then L_1^* will also be Context Free language.

Proof: if L_1 is a CFL obtained from CFG $G_1(V_1, T, P_1, S_1)$ as $L_1 = L(G_1)$

Define $L_1^* \implies \{L(G_1)\}^*$

Now, define a grammar for $\{L(G_1)\}^* = G(S \cup V_1, T, S \rightarrow S_1 S | \epsilon \cup P_1, S)$ which is a CFG. So, $\{L(G_1)^*\}$ is a CFL.

So, L_1^* is also a CFL.

9.4 Context-free languages are not closed under

Intersection: If L_1, L_2 are context free languages, then $L_1 \cap L_2$ is not necessarily context free.

Proof: Define $L_1 = \{a^n b^n c^m | n \geq 0 \& m \geq 0\}$ and $L_2 = (a^m b^n c^n | n \geq 0 \& m \geq 0\}$

Define $L_3 = L_1 \cap L_2 = \{a^n b^n c^n | n \geq 0 \}$ and it is not not context-free. So, $L_1 \cap L_2$ is need not necessarily a context-free.

Intersection with Regular Language: If L_1 is a regular language and L_2 is a context-free language, then $L_1 \cap L_2$ is a context-free language.

Complement: If L_1 is a context free language, then $\bar{}_{L_1}$ may not be context-free.