THEORY OF COMPUTATION

UNIT - 2

K CHAIRMADURAI ME,.. AP
Department of Computer Science and Engineering
Adhiparasakthi Engineering College , Melmaruvathur

Unit II Regular Expressions and Languages

Syllabus:

- **Regular Expressions**
- **FA and Regular Expressions**
- **Proving Languages not to be regular**
- **Closure properties of Regular languages**
- **Equivalence and minimization of Automata**

Regular Expressions

Now we switch our attention from machine like descriptions of languages deterministic and non deterministic finite automata to an **algebraic description** the **regular expression.**

We shall find that regular expressions can define exactly the same languages that the various forms of automata describe the regular languages.

The languages accepted by finite automata are easily **described by simple expressions** called **regular expressions.**

Regular expression uses a set of operators (like +,., *,….) to represent the language.

The set of strings accepted by finite automata is known as **regular language**.

Regular Expressions - Applications

- Thus regular expressions serve as the input language for many systems that process strings.

1. **Search commands such as the UNIX grep** or equivalent commands for finding strings that one sees in Web browsers or **text formatting systems.**

2. Different **search systems** convert the **regular expression into either a DFA or an NFA** and simulate that automaton on the file being searched.

3. **Lexical Analyzer** accepts tokens as inputs and produces a DFA that recognizes which token appears next on the input.

The Operators of Regular Expressions

- For a simple example: The regular expression **01* 10*** denotes the language consisting of all strings that are either a single 0 followed by any number of 1s or a single 1 followed by any numberof 0s
- There are **three operations** on languages that the operators of regular expressions represent.
- These operations are
    1. Union
    2. Concatenation
    3. Closure

Operators of Regular Expressions - UNION

- The **union** of two languages, L and M is denoted by **LUM** which are a set of strings that are either in L or M or both.
- Example:

  L = {001, 10, 111}

  M = {**ε**, 001}

  LUM = {**ε**, 10,001,111}

Operators of Regular Expressions - CONCATENATION

- The concatenation of languages L and M, denoted **L.M** which are a set of strings that can be formed by taking any string in L and concatenating with any string in M.
- Example:

  L = {001, 10, 111}

  M = {**ε**, 001}

  L.M = {001, 10, 111, 001001, 10001, 111001}

Operators of Regular Expressions - CLOSURE

- The **closure** of a language, L is denoted as **L*** represent the set of those strings that can be formed by taking any number of strings from L, possibly with repetitions and concatenating all of them.
- Example:

L = {0, 1} then L* is the all strings of 0's and 1's

RE's: Definition
Building Regular Expressions

For each regular expression E we describe the language it represents which we denote L(E).

Basis 1: If *a* is any symbol, then **a** is a RE, and L(**a**) = {a}.

- Note: {a} is the language containing one string, and that string is of length 1.

- Basis 2: The constant ϵ is a RE, and L(ϵ) = {ϵ}.
- Basis 3: The constant ∅ is a RE, and L(∅) = ∅.

RE's: Definition
Building Regular Expressions

- Induction 1: If E_1 and E_2 are regular expressions, then E_1+E_2 is a regular expression denoting the union of L(E1) and L(E2).

$$L(E_1+E_2) = L(E_1) \cup L(E_2)$$

- Induction 2: If E_1 and E_2 are regular expressions, then E_1E_2 is a regular expression denoting the concatenation of L(E1) and L(E2).

$$L(E_1E_2) = L(E_1).L(E_2)$$

  - *Concatenation* : the set of strings wx such that w is in $L(E_1)$ and x is in $L(E_2)$.

- Induction 3: If E is a RE, then E* is a RE denoting the closure of L(E)

$$L(E^*) = (L(E))^*.$$

  - *Closure*, or "Kleene closure" = set of strings $w_1w_2...w_n$, for some $n \geq 0$, where each w_i is in L(E).
  - Note: when n=0, the string is ϵ.

Precedence of Operators

- Parentheses may be used wherever needed to influence the grouping of operators.
- Order of precedence is * (highest), then concatenation **.**, then + (lowest).

1. The star operator* is of highest precedence. That is it applies only to

the smallest sequence of symbols to its left that is a well formed regular expression.

2. Next in precedence comes the concatenation or dot operator. After grouping all stars to their operands, we group concatenation operators to their operands.

3. Finally, all unions(+operators) are grouped with their operands.

EX: The expression **01*1** is grouped into **(0(1*))+1**

Example

- **Set of strings of a's and b's of any length including the null string.**
- L = { ε, a, b, aa , ab , bb , ba, aaa…….}

(a+b)*

- **Set of strings of a's and b's ending with the string 'abb'**

L = {abb, aabb, babb, aaabb, ababb, ……..}

(a+b)*abb

- **Set of strings consisting of even number of a's followed by odd number of b's**

L = {b, aab, aabbb, aabbbbb, aaaab, aaaabbb, ……..}

(aa)*(bb)*b

Example

- **Write the regular expression for the language accepting all the string which are starting with 1 and ending with 0, over ∑ = {0, 1}.**

  1 (0+1)* 0

- **Write the regular expression for the language starting and ending with a and having any having any combination of b's in between.**

  a b* b

- **Write the regular expression for the language starting with a but not having consecutive b's.**

  L = {a, aba, aab, aba, aaa, abab, ...}

  (a + ab)*

Example

- Write regular expressions for the following languages

The set of strings over alphabet {a,b,c} containing at least one 'a' and atleast one 'b'.

soln:

- Set of Strings over {a,b,c} = **(a+b+c)***
- containing atleast one 'a' = **(a+b+c)*a**
- containing atleast one 'b' = **(a+b+c)*b**

- Then the result is **(a+b+c)*a (a+b+c)*b(a+b+c)***

Example

- Write the regular expression for the language accepting all the string in which any number of a's is followed by any number of b's is followed by any number of c's.

a* b* c*

- Write the regular expression for the language over ∑ = {0} having even length of the string.

L = {ε, 00, 0000, 000000, ……}

(00)*

Home work

Write regular expressions for the following languages

* a) The set of strings over alphabet $\{a, b, c\}$ containing at least one a and at least one b.

b) The set of strings of 0's and 1's whose tenth symbol from the right end is 1.

c) The set of strings of 0's and 1's with at most one pair of consecutive 1's.

a) The set of all strings of 0's and 1's such that every pair of adjacent 0's appears before any pair of adjacent 1's.

b) The set of strings of 0's and 1's whose number of 0's is divisible by five.

Home work

a) The set of all strings of 0's and 1's not containing 101 as a substring.

b) The set of all strings with an equal number of 0's and 1's, such that no prefix has two more 0's than 1's, nor two more 1's than 0's.

c) The set of strings of 0's and 1's whose number of 0's is divisible by five and whose number of 1's is even.

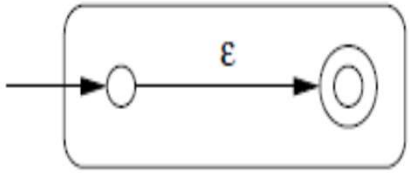
write a regular expression for the set of strings that consist of alternating 0's and 1's

Converting Regular Expressions to Automata

- **Showing that every language L that is L(R) for some regular expression R, is also L(E) for some ε-NFA E.**
- The proof is a structural induction on the expression R. We start by showing how to construct automata for the basis expressions single symbols, **ε,** ∅.
- We then show how to combine these automata into larger automata that accept the union, concatenation, or closure of the language accepted by smaller automata.

RE to ϵ-NFA: Basis

- Symbol **ε**:



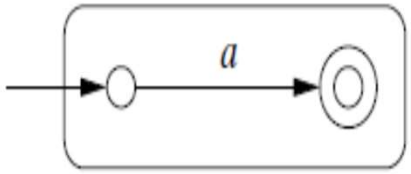
(a)

- ∅:

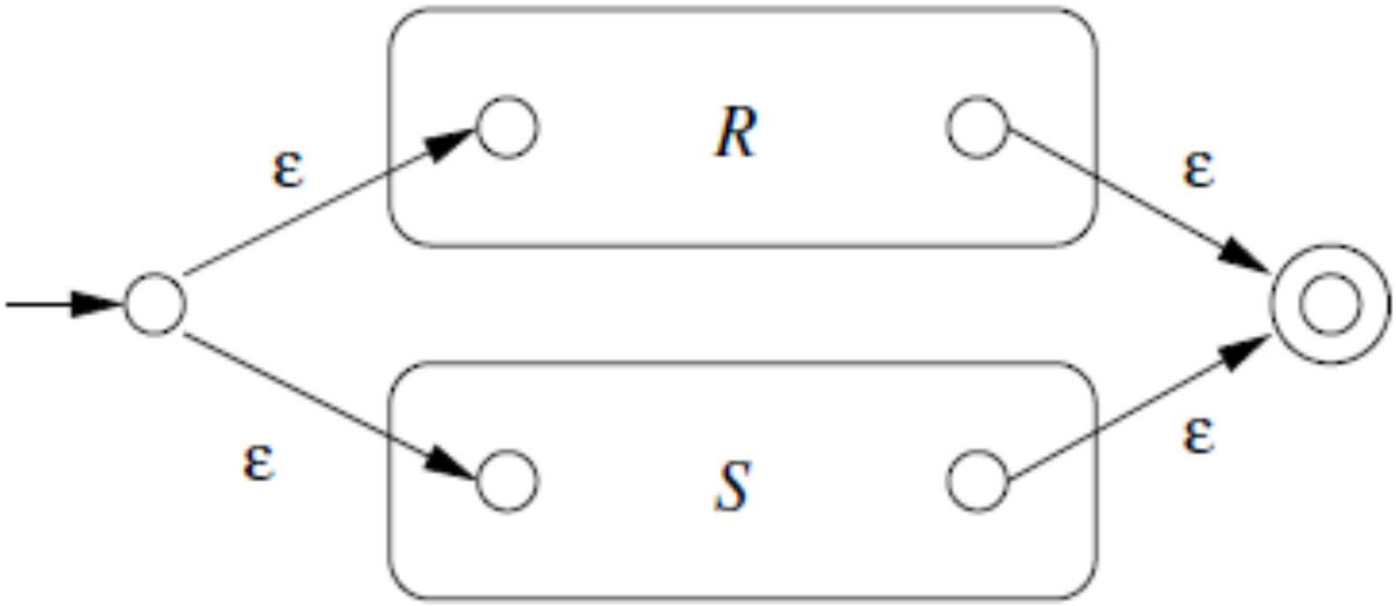


(b)

- a:

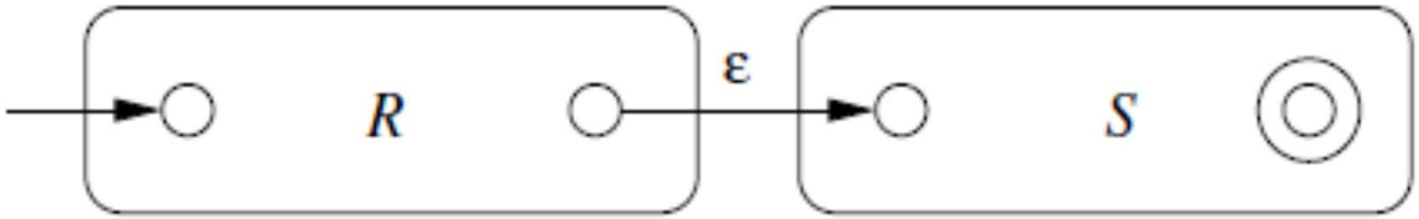


RE to ϵ-NFA: Induction 1 – Union



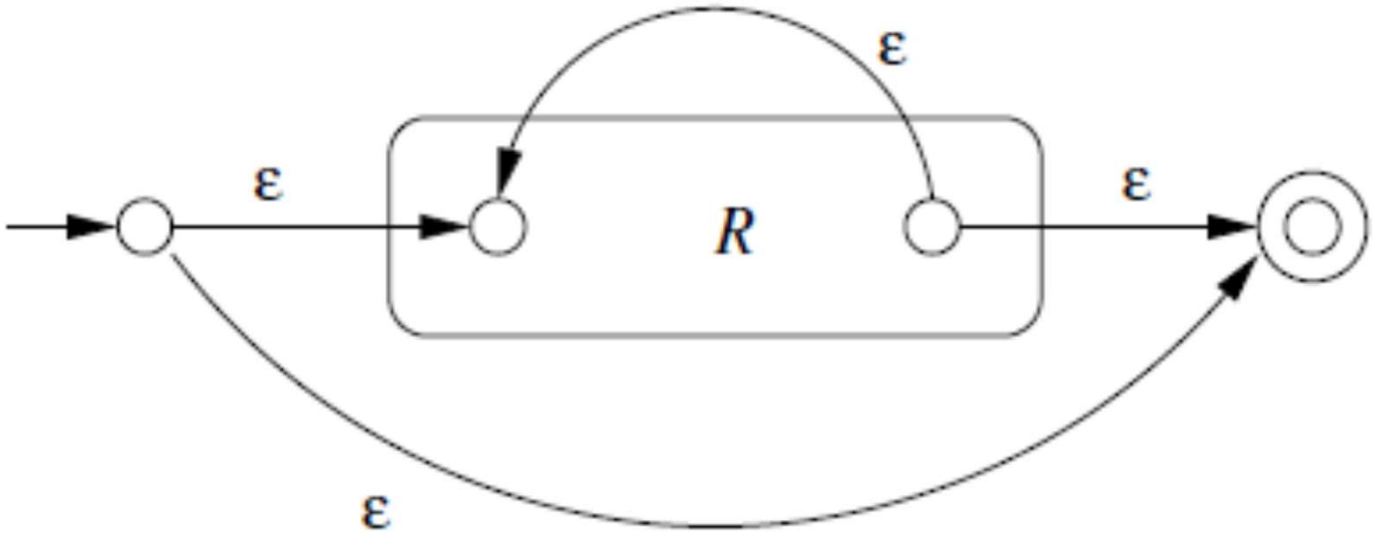
R+S

RE to ϵ-NFA: Induction 2 – Concatenation



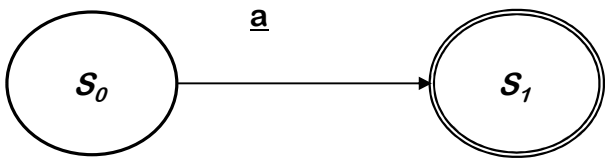
R.S

RE to ϵ-NFA: Induction 3 – Closure

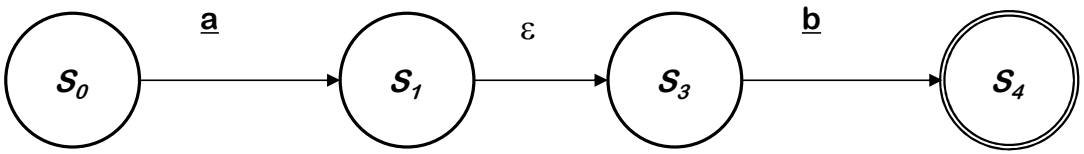


R*

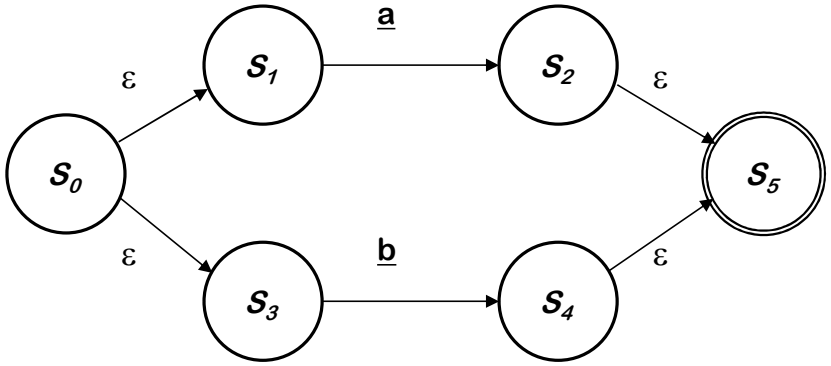
RE →ε-NFA using Thompson's Construction



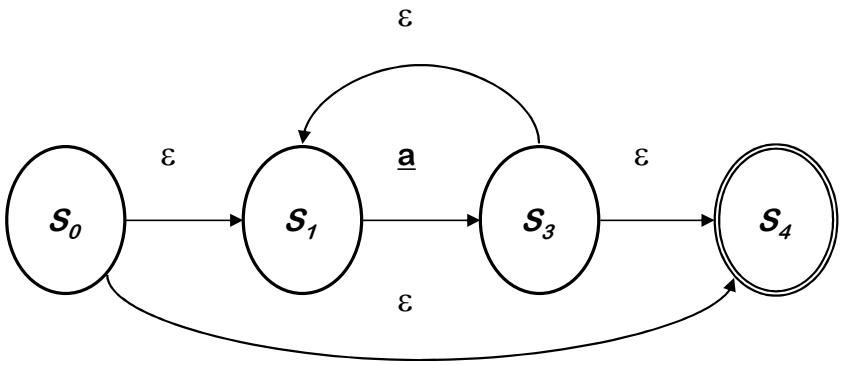
NFA for a



NFA for ab

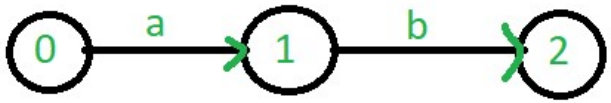


NFA for a + b

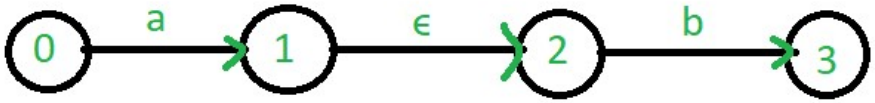


NFA for a*

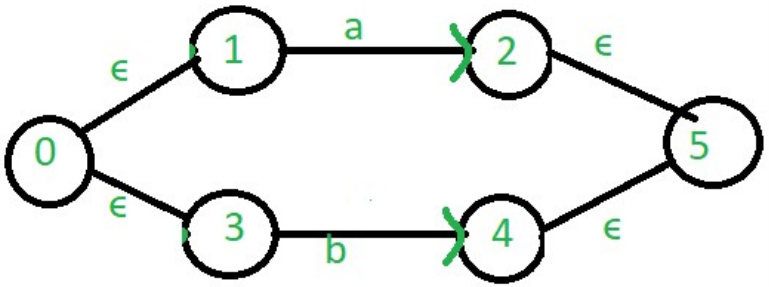
1 ab



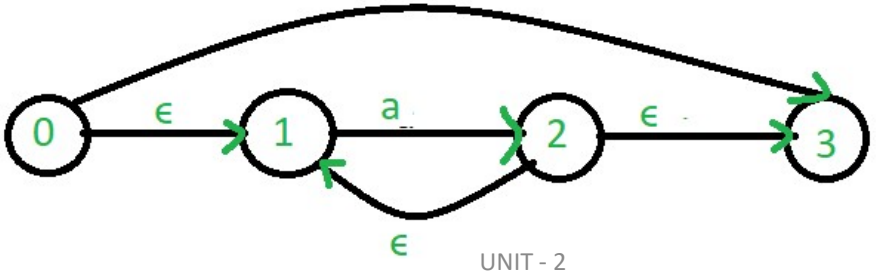
OR



2. a+b

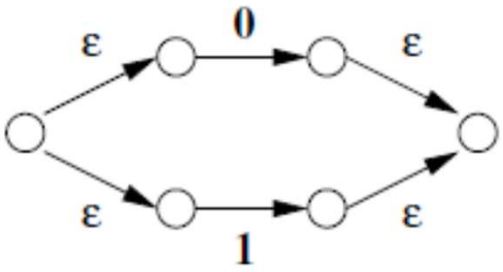


3. a*

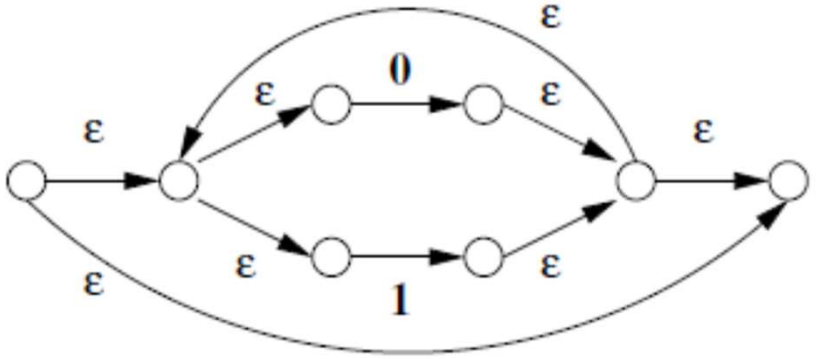


RE $\rightarrow$ε-NFA using Thompson's Construction

- **Convert the RE: (0 + 1)* 1(0+1) to ϵ-NFA**
  - (0 + 1)

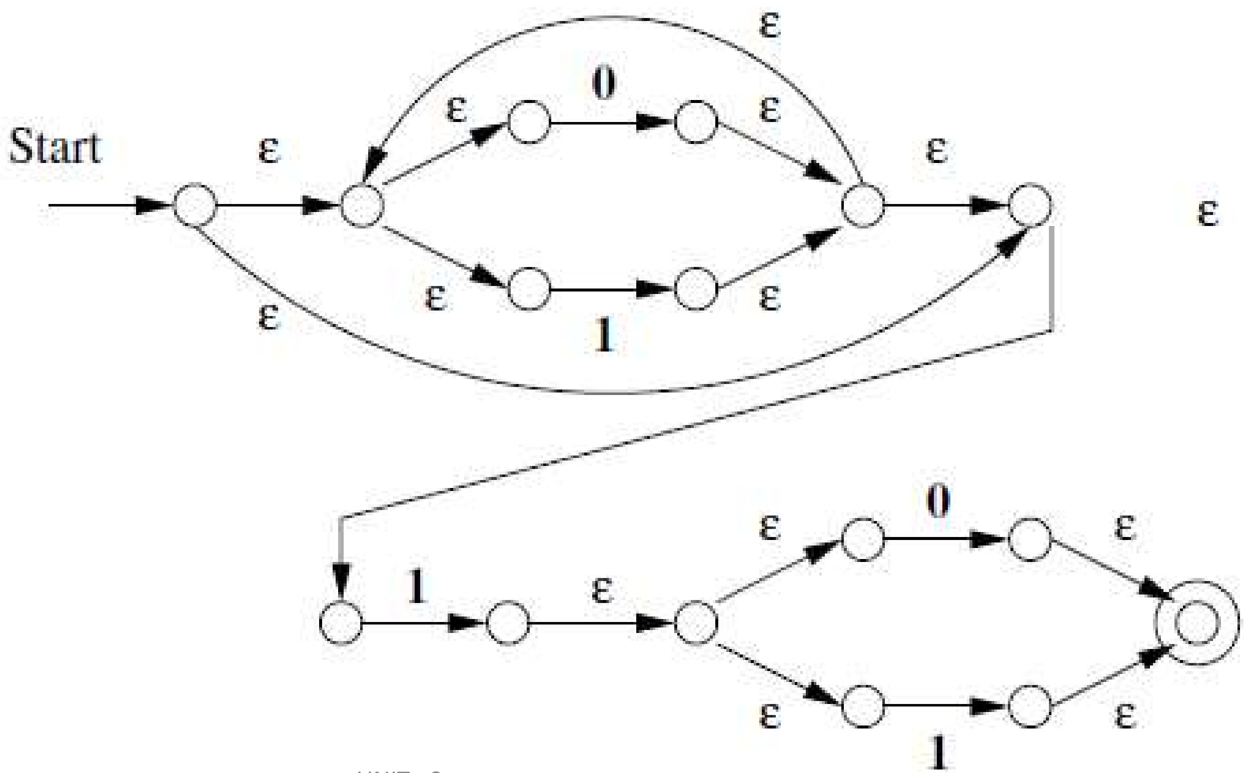


  - (0 + 1)*



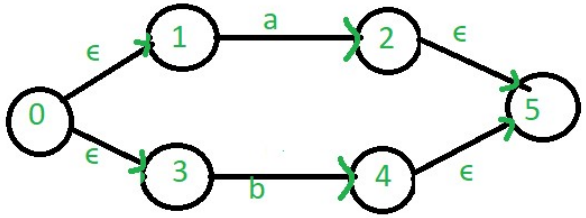
RE →ε-NFA

- (0 + 1)*1(0+1)

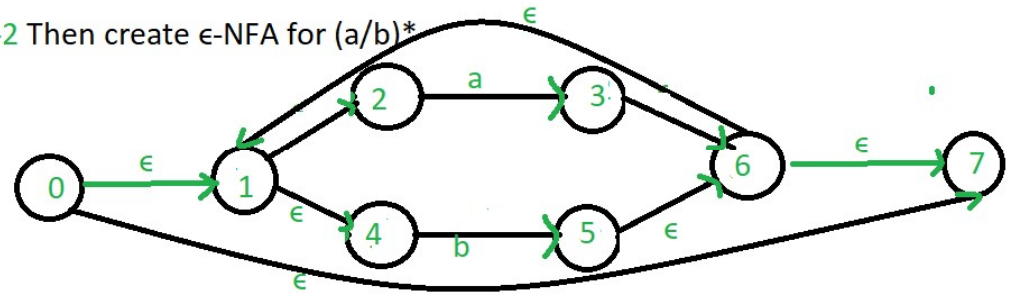


Create a ∈-NFA for regular expression: (a/b)*a

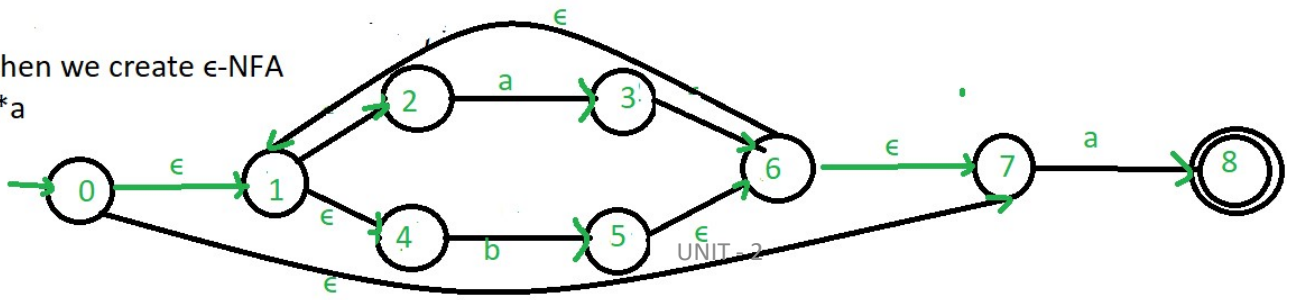
Step-1 First we create ϵ-NFA for (a/b)



Step-2 Then create ϵ-NFA for (a/b)*



Step-3 Then we create ϵ-NFA for(a/b)*a



Home Work

Convert the following regular expressions to NFA's with ϵ transitions

01*

(0+1)01

00(0+1)*

(01+10)*101

10+(0+11)0*1

- The **theory of computation** is the branch that deals with how efficiently problems can be solved on a model of computation, using an algorithm.
- It uses the elements of automata theory, computability theory and complexity theory to understand the **nature of computing problems and how computing operations** are performed.
- **Automata theory** is the study of **abstract computational devices.**
- **Computability theory** deals with the **problem is solvable** on a computer.
- **Complexity theory** considers **how efficiently** the problem can be solved.

- **Finite automata** are used to recognize patterns.
- It takes the **string of symbol as input** and changes its state accordingly. When the **desired symbol is found, then the transition occurs.**
- At the time of transition, the automata can either move to the next state or stay in the same state.
- Finite automata have two states, **Accept state or Reject state**. When the input string is processed successfully, and the automata reached its final state, then it will accept.
- A finite automaton is a collection of **5-tuple (Q, ∑, δ, q0, F)**

- **DFA** refers to **deterministic finite automata.** Deterministic refers to the uniqueness of the computation. In the DFA, the machine goes to **one state only for a particular input** character. DFA does **not accept the null move.**
- **NFA** stands for **non-deterministic finite automata**. It is used to **transmit any number of states for a particular input**. It can **accept the null move.**
- A **regular language** is a language that can be expressed with a r**egular expression or a deterministic or non-deterministic finite automata** or state machine.
- The language accepted by finite automata can be **easily described by simple expressions** called **Regular Expressions.**

- A **regular grammar** is a mathematical object, G, with four components, **G = (N, Σ, P, S)**, where

N is a nonempty, finite set of nonterminal symbols,

Σ is a finite set of terminal symbols , or alphabet, symbols,

P is a set of grammar rules, each of one having one of the forms

$A \rightarrow aB$

$A \rightarrow a$

$A \rightarrow \varepsilon$, for $A, B \in N$, $a \in \Sigma$, and ε the empty string, and

$S \in N$ is the start symbol

Unit III Context Free Grammar and Languages

Syllabus

- Context Free Grammar (CFG)
- Parse Trees
- Ambiguity in Grammar and Languages
- Definition of Push Down Automata
- Equivalance of PDA and CFG
- Deterministic Push Down Automata

GRAMMAR

- Every language is generated by some grammar.
- It produces strings of a language by means of replacing symbols.
- In general, **a grammar is a set of production rules for generating strings in a language.**
- A grammar can also be called as formal grammar, which generates strings in a formal language.

GRAMMAR

A grammar generates a language with the aid of four elements, **G=(V,T,P,S)** such as,

- V - Non Terminal symbol
- T - Terminal symbol
- S - Start symbol
- P - Production rules

GRAMMAR

- **Terminal** refers to the **symbols of the alphabet** being used.
  - There are several types of terminals namely –
    - Binary alphabet 0 and 1
    - Lowercase alphabet a, b, … , z
    - Decimal alphabets 0, 1, 2, … , 9 etc.
  - Most commonly used terminals are **binary and lowercase types**.
- The **non-terminal** symbols are those **variables** that denote/produce some terminals in-turn.
  - Non-terminals are equivalent to the states of automation.
  - They are usually represented by **uppercase alphabets** like. A, B, … , Z.
- The **start symbol** is a special **non-terminal** that works similar to the start state of a finite automation.
  - It is represented by an element in variable set.
- The **production rules** are the translation functions/replacement rules of the variables.

Example: (Informal)

- Consider the string: **'sentence**'.
- A sentence is formed by 'subject', 'verb', 'object'.

  So,

  Start symbol: <sentence>

  and,

  <sentence> → <subject> <verb> <object>
- Now the above statement forms a production rule, which requires some strings to be substituted /replaced for subject, verb and object.
- Let
  - <subject> fruits
  - <verb> are
  - <object> fresh
- Therefore,

  <sentence> →fruits are fresh ------ one string in the language

  where
  - **Terminals:** fruits, are, fresh
  - **Nonterminals:** <sentence>, <subject>, <verb>, <object>
  - **Start symbol:** <sentence>
  - **Production rules**:
    - **<sentence> → <subject> <verb> <object>**
    - **<subject>→ fruits**
    - **<verb> →are**
    - **<object>→ fresh**

Example: (formal)

- A grammar thus generates a formal language, **which is a set of strings formed by applying production rules that is derived from the start symbol.**
- Consider, the **terminals a and b**, the **start symbol S,** and the **production rules:**

  S → aSb

  S → ab

- This may generate,

  S → a**S**b ---- by applying rule 1.

  → a**aSb**b ----by applying rule 1 on S.

  →aa**ab**bb ---- by applying rule 2 on S.

- Thus the above production rules may generate,

  L = {ab, aabb, aaabbb, a4b4, … }

- Therefore the production rules generate a language containing equal number of a`s and b`s and is defined as

  $L = \{a^n b^n \mid n \geq 1\}$

PROBLEMS ON GRAMMARS

Construct the grammar representing the set of palindromes over (0 + 1)*.

Solution:

- The first possibility is having a string of single / zero length.

  $S \rightarrow 0 \mid 1 \mid \varepsilon$

- For other strings,

  $S \rightarrow 0S0$ first and last data are equal

  $S \rightarrow 1S1$ first and last data are equal

- Thus the grammar is given by

  $S \rightarrow 0 \mid 1 \mid \varepsilon$

  $S \rightarrow 0S0$

  $S \rightarrow 1S1$

PROBLEMS ON GRAMMARS

Construct a grammar for L = {a^n | n is odd}.

Solution:

When n = 1 S = a1 = a [S⟶ a]
When n = 3 S = a3 = aaa [S ⟶ aSa]
When n = 5 S = a5 = aaaaa [S ⟶ aSa]

Therefore the production shall be defined as,

S ⟶ a
S ⟶ aSa

The grammar G = (V, T, P, S) is given by

V = {S}
T = {a}
P = {S ⟶ aSa | a}
S = {S}

TYPES OF GRAMMARS

- **Noam Chomsky** formalized grammars in 1956.
- He classified them into four types which are collectively called as 'the Chomsky hierarchy'.
- Chomsky hierarchy consists of the following four levels, namely
  - **Type 0 grammars / unrestricted grammars**
  - **Type 1 grammars / context sensitive grammars**
  - **Type 2 grammars / context free grammars**
  - **Type 3 grammars / regular grammars**

Type 0 grammars

- These are free / unrestricted grammars that have no restrictions on the production rules.
- The production rule is of the form,

$$\alpha \rightarrow \beta \mid \alpha \neq \varepsilon$$

where α, β can be strings composed of terminals and non-terminals.

- The most important property is that the left hand side () should contain at least one non-terminal and cannot be empty.
- This type of grammar generates **Recursively Enumerable Language**
- This grammar can be modeled using **Turing Machine.**

Type 1 grammar

- The languages that follow type 1 grammars are called as **context-sensitive grammars (CSG).**
- A **language is said to be context sensitive** if the production rules are of the form,

$$\alpha A \beta \Rightarrow \alpha \gamma \beta$$

$\gamma \rightarrow$ non-empty
$\alpha, \beta \rightarrow$ can be empty
where,
A is a non-terminal symbol
α, β, γ are any combination of terminals and non-terminals.

- Thus CSG possess two properties
  - At least one symbol on the left hand side should be a non terminal.
  - The number of symbols on the left hand side does not exceed the number of symbols on the right hand side.
- The machine/automaton that recognizes a CSG is called **linear-bounded machine/automaton** which is a finite automaton with memory to store symbols.

Type 2 grammars

- Type 2 grammars are **context free grammars.**
- A grammars, G = (V, T, P, S) is said to be context free if the production rule is of the form,

$$A \rightarrow \alpha$$

The transition allows

$$A \rightarrow \varepsilon \text{ [i.e., } \alpha \rightarrow \varepsilon]$$

where,

A is a non terminal symbol

α is any terminal or non-terminal symbol

- Here, the left hand side of the transition rule consists of only one non-terminal.
- Type 2 grammars are the basis of the syntax of most of the programming languages such as XML.
- CFG can be handled using the **pushdown automaton** that uses stack memory to store the symbols.

Type 3 grammars

- Type 3 grammars are **regular grammars** that describe regular / formal languages.
- This grammar can be modeled using **finite automata.**
- These grammars contain production rules consisting of
  - only one non-terminal at the left hand side
  - the right hand side having a single terminal and may or may not be followed by non terminals
- Example

$$A \rightarrow \varepsilon$$

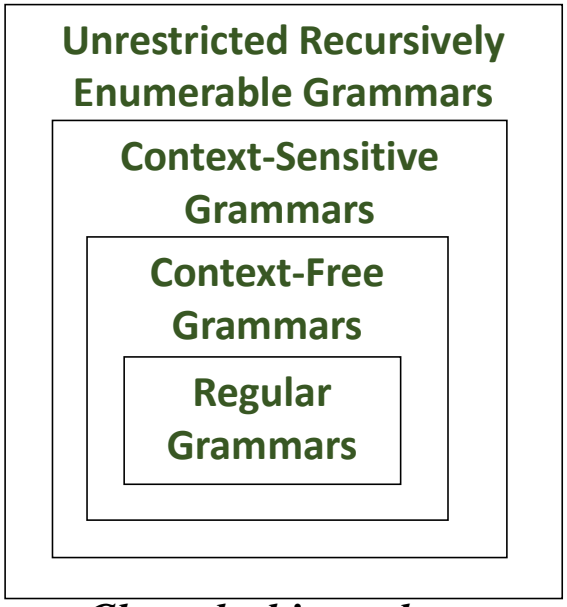
$$A \rightarrow a$$

$$A \rightarrow b$$

$$A \rightarrow aA$$

Chomsky's Hierarchy

- **Type 3 grammar ⊆ type 2 grammar ⊆ type 1 grammar⊆ type 0 grammar**
- **Regular grammars ⊆ context free grammars ⊆ context sensitive grammars ⊆ unrestricted grammars**



Chomsky hierarchy

Grammar Type	Grammar Accepted	Language Accepted	Automaton
Type 0	Unrestricted grammar	Recursively enumerable language	Turing machine
Type 1	Context-sensitive grammar	Context-sensitive language	Linear-bounded automaton
Type 2	Context-free grammar	Context-free language	Pushdown automaton
Type 3	Regular grammar	Regular language	Finite state automaton

CONTEXT-FREE GRAMMARS

Formal definition

A context free grammar (CFG) is described by **four tuples** (quadruple) as,

G = (V, T, P, S)

where,

- V is a finite set of variables / **non terminals**
- T is a finite set of terminals / **terminal symbols**
- P is a finite set of transitions / **production rules**
- S is the **start symbol**, where S∈ V.

CONTEXT-FREE GRAMMARS

- There are **four important components** in a grammatical description of a language:
  - There is a **set of symbols that form the strings of the language** being defined. They are called **terminal symbols**, represented by **T**.
  - There is a **finite set of variables**, called **non-terminals.** These are represented by **V.**
  - **One of the variable** represents the language being defined; it is called the **start symbol**. It is represent by **S**.
  - There are **finite set of rules** called **productions** that represent the recursive definition of a language. Each production **P** consists of:
    - A **variable** that is being defined by the production. This variable is often called the **head of the production**.
    - The **production symbol ->**.
    - A **string of zero or more terminals and variable.**

CONTEXT-FREE GRAMMARS (CFG)

- The production rules are the recursive definitions of the languages.
- The production is stated as,

  Head of the production → body of the production

  where,
  - Head of the production is any **variable defined in V**.
  - Body of the production is set of zero or more terminals and/or non terminals.

Thus the production is of the form,

$$V_i \rightarrow \alpha_i ,$$

where,

$V_i \varepsilon V$

$\alpha_i \varepsilon \{VUT\}$

Example

Generation of palindromes over the terminals {a, b}

Basis: {ε, a, b}

Induction: {aa, bb, aba, bab, aaa, bbb, aaaa, abba, bbbb, baab, ….}

To generate the basics form of strings, we get

$S \rightarrow \varepsilon$

$S \rightarrow a$

$S \rightarrow b$

Other strings, specified by the induction are given by

$S \rightarrow aSa$ generates aa, aaa, aba, aaaa, abba etc.,

$S \rightarrow bSb$ generates bb, bbb, bab, bbbb, baab, etc,

Thus the grammar, G = (V, T, P, S) is given by G = ({S}, {a,b}, P, {S}) where P is given by

$$S \rightarrow \varepsilon \mid a \mid b \mid aSa \mid bSb$$

Problem-1

Construct the CFG for the language having any number of a's over the set ∑= {a}.

Solution:

As we know the regular expression for the above language is

r.e. = a*

Production rule for the Regular expression is as follows:

S → aS rule 1

S → ε rule 2

The r.e. = a* can generate a set of string **{ε, a, aa, aaa,.....}.**

We can have a null string because S is a start symbol and rule 2 gives S → ε.

Now if we want to derive a string "aaaaaa", we can start with start symbols.

S
aS
aaS rule 1
aaaS rule 1
aaaaS rule 1
aaaaaS rule 1
aaaaaaS rule 1
aaaaaaε rule 2
aaaaaa

Problem-2

Construct a CFG for the regular expression (0+1)*

Solution:

The CFG can be given by,

Production rule (P):

S → 0S | 1S

S → ε

The rules are in the combination of 0's and 1's with the start symbol. Since (0+1)* indicates {ε, 0, 1, 01, 10, 00, 11, ...}. In this set, ε is a string, so in the rule, we can set the rule S → ε.

Problem-3

Construct a CFG for a language L = {wcw^R | where w € (a, b)*}.

Solution:

The string that can be generated for a given language is {aacaa, bcb, abcba, bacab, abbcbba, ...}

The grammar could be:

S → aSa rule 1

S → bSb rule 2

S → c rule 3

Now if we want to derive a string "abbcbba", we can start with start symbols.

S → aSa
S → abSba from rule 2
S → abbSbba from rule 2
S → abbcbba from rule 3

Thus any of this kind of string can be derived from the given production rules.

Problem-4

Construct a CFG for the language L = {$a^n b^{2n}$ where n>=1}

- **Solution:**
- The string that can be generated for a given language is {abb, aabbbb, aaabbbbbb....}.

The grammar could be:

S → aSbb | abb

Now if we want to derive a string "aabbbb", we can start with start symbols.

S → aSbb
S → aabbbb

Problem-5

Construct a CFG for L = $\{a^n b^n \mid n \geq 0\}$

Solution

The possible productions are {ε, ab, aabb, aaabbb, $a^4 b^4$, etc}

So the productions are,

$S \rightarrow \varepsilon$

$S \rightarrow aSb$

Thus, the CFG, G = (V, T, P, S) is given by

V = {S}

T = {a, b}

P = { $S \rightarrow aSb \mid \varepsilon$}

S = {S}

Problem-6

Construct a CFG representing the set of palindromes over (0+1)*.

Solution

The possible productions are {ε, 0, 1, 00, 11, 000, 111, etc}

So the productions are

$P \Rightarrow \quad S \rightarrow \varepsilon$

$S \rightarrow 0$

$S \rightarrow 1$

$S \rightarrow 0S0$

$S \rightarrow 1S1$

Thus the CFG is given by G = (V, T, P, S) where

$V = \{S\}$

$T = \{0, 1\}$

$P = \{ S \rightarrow 0S0 \mid 1S1 \mid 0 \mid 1 \mid \varepsilon\}$

$S = \{S\}$

Problem-7

Construct a CFG for the language, L = {w C w^R | w is a string in (a+b)*}.

Solution

The intermediate symbol is 'C'. The string written on the left of 'C' is reversed at the right side.

The possible strings generated from L can be {c, aca, bcb, aacaa, bbcbb, abcba, bacab, abacaba, babcbab, aaacaaa, etc.}

∴ The production rules are defined to be,

$S \rightarrow c$

$S \rightarrow aSa$

$S \rightarrow bSb$

Thus the grammar, G = (V, T, P, S) where

$V = \{S\}$

$T = \{a, b, c\}$

$P = \{S \rightarrow aSa \mid bSb \mid c\}$

$S = \{S\}$

The language of the CFG

- The language of a grammar **G = (V, T, P, S) is denoted as L(G).**
- Every CFG generates its corresponding language.
- Every string of a language is generated by applying the production rules, a finite number of times.
- Each string is **derived from the start symbol** and ends with a final string, consisting of a set of terminals.

The language of the CFG
Formal definition

- The language L(G) of a grammar G, is a set of strings derived from the start symbol.

It is denoted as

$$L(G) = \left\{ w \mid w \in T^* \text{ and } S \overset{*}{\underset{G}{\Rightarrow}} w \right\}$$

'*' denotes that the grammar, G is applied over S several times.

If the grammar is applied once, it is denoted as $S \underset{G}{\Rightarrow} w$

- The language generated from a context - free grammar is called a context - free language.

Example

Design the language for the CFG given. The grammar, G = ({S}, {0,1}, P, {S}) where

P: S → 0S1

S → 01

Solution

For the given grammar, we can generate the strings as

$S \rightarrow 01$

$S \rightarrow 0\underline{S}1 \rightarrow 0011$

$S \rightarrow 0\underline{S}1 \rightarrow 00\underline{S}11 \rightarrow 000111$

$S \rightarrow 0\underline{S}1 \rightarrow 00\underline{S}11 \rightarrow 000\underline{S}111 \rightarrow 00001111$ ……

Thus the strings are $\{01, 0011, 000111, 0^4 1^4, 0^5 1^5, \ldots\}$

The language, $L = \{0^n 1^n \mid n \geq 1\}$ generates equal number of zeroes and ones.

Problem-1

Let G = {{S, c}, {a,b}, P, S} where S → aCa, C → aCa|b. Find L(G).

Solution

1. $S \rightarrow a\underline{C}a \rightarrow aba$
2. $S \rightarrow a\underline{C}a \rightarrow aa\underline{C}aa \rightarrow aabaa$
3. $S \rightarrow \mathbf{aCa} \rightarrow aa\underline{C}aa \rightarrow aaa\underline{C}aaa \rightarrow aaabaaa$

.

.

.

$\therefore L(G) = \{a^n b\, a^n \mid n \geq 1\}$

Problem-2

Find out the CFL for,

$S \rightarrow aSb \mid aAb$
$A \rightarrow bAa$
$A \rightarrow ba$

Solution

1. $S \rightarrow aSb$
   $\rightarrow aaAbb$ [using $S \rightarrow aAb$]
   $\rightarrow aababb$ [using $A \rightarrow ba$]
2. $S \rightarrow aSb$
   $\rightarrow aaAbb$ [using $S \rightarrow aAb$]
   $\rightarrow aabAabb$ [using $A \rightarrow bAa$]
   $\rightarrow aabbaabb$ [using $A \rightarrow ba$]
3. $S \rightarrow aAb$
   $\rightarrow abab$ [using $A \rightarrow ba$]
4. $S \rightarrow aAb$
   $\rightarrow abAab$ [using $A \rightarrow bAa$]
   $\rightarrow abbaab$ [using $A \rightarrow ba$]

From the above strings, the CFL is given as,

$$L(G) = \{a^n b^m a^m b^n \mid m, n \geq 1\}$$

Applications of CFLs & CFGs

1. Grammars are used to describe programming languages
   - Compilers use parsers for syntactic checking
   - Parsers can be expressed as CFGs
     1. Balancing paranthesis
     2. If-then-else
     3. C paranthesis matching { … }
     4. Pascal *begin-end* matching
     5. YACC (Yet Another Compiler-Compiler)
2. Markup languages
3. XML and Document Type Definition

Derivations

- While inferring whether the given input string belongs to the given CFG, we can have two approaches.
  - **Recursive inference** -- using the rules from **body to head**
  - **Derivation** --using the rules from **head to body**
- Recursive inference
  - Here we take strings from each variables, **concatenate them in proper order and infer that the resulting strings** in the language of the variable in the head.
- Derivation
  - Here we use the productions from head to body (i.e) **from start symbol expanding till reaches the given string.**

Derivations

- Derivations are **the set of strings that are derived from the start symbol, after applying the production rules, a finite number of times.**

$$S \overset{*}{\underset{G}{\Rightarrow}} \omega \mid \omega\ \varepsilon\ T^*$$

- Derivation is a sequence of production rules. It is used to get the input string through these production rules. During parsing, we have to take two decisions. These are as follows:
  - We have to **decide the non-terminal** which is to be replaced.
  - We have to **decide the production rule by which the non-terminal** will be replaced.
- Types of derivations
  - There are two types of derivations,
    - **Leftmost derivations**
    - **Rightmost derivations**

Representation of derivations

- Derivations are represented in either of the two forms.
  - They are,
    - **Sentential form**
    - **Parse tree form**
- Sentential form

  Sentential form is the **derivation derived from the start symbol, by applying the rules/productions.**
  - Let the G = (V, T, P, S) be a CFG then

$$S \overset{*}{\Rightarrow} \alpha \quad \alpha \in (TUV)^*$$

  is in sentential form, where

Example

Let G = (V, T, P, S) be given by G = ({S, A, B}, {0, 1}, P, {S}).

P: $S \rightarrow A1B$

$A \rightarrow 0A \mid \varepsilon$

$B \rightarrow 0B \mid 1B \mid \varepsilon$

Solution

Consider the string 00101, which is to be generated by the given grammar using sentential form.

$S \Rightarrow \underline{A}1B$ [Using $S \rightarrow A1B$]

$\Rightarrow 0\underline{A}1B$ [Using $A \rightarrow 0A$]

$\Rightarrow 00\underline{A}1B$ [Using $A \rightarrow 0A$]

$\Rightarrow 001\underline{B}$ [Using $A \rightarrow \varepsilon$]

$\Rightarrow 0010\underline{B}$ [Using $B \rightarrow 0B$]

$\Rightarrow 00101\underline{B}$ [Using $B \rightarrow 1B$]

$\Rightarrow 00101$ [Using $B \rightarrow \varepsilon$]

Thus the string 00101 ϵ L(G).

Types of sentential form

- There are two types of sentential forms namely
  - **Left sentential forms**
  - **Right sentential forms**
- **Left sentential form**
  - When the derivations are generated by **expanding the leftmost symbol** is called left sentential form.
  - It is denoted as, $S \underset{\ell m}{\overset{*}{\Rightarrow}} \alpha$
- **Right sentential form**
  - The derivation generated by performing **substitution on the right most variable** is called right sentential form.
  - It is denoted as $S \underset{r m}{\overset{*}{\Rightarrow}} \alpha$

Example

Consider the grammar given below

$S \rightarrow A1B$

$A \rightarrow 0A \mid \varepsilon$

$B \rightarrow 0B \mid 1B \mid \varepsilon$

Give the leftmost derivation for the string 1001.

Solution

$S \underset{lm}{\Rightarrow} \underline{A}1B$ [Using $S \rightarrow A1B$]

$\underset{lm}{\Rightarrow} 1\underline{B}$ [Using $A \rightarrow \varepsilon$]

$\underset{lm}{\Rightarrow} 10\underline{B}$ [Using $B \rightarrow 0B$]

$\underset{lm}{\Rightarrow} 100\underline{B}$ [Using $B \rightarrow 0B$]

$\underset{lm}{\Rightarrow} 1001\underline{B}$ [Using $B \rightarrow 1B$]

$\underset{lm}{\Rightarrow} 1001$ [Using $B \rightarrow \varepsilon$]

- Thus the string '1001' is thus derived using left sentential form.
- At each step, the left most variable is expanded.

Consider

$S \rightarrow A1B$

$A \rightarrow 0A|\varepsilon$

$B \rightarrow 0B|1B|\varepsilon$

Generate the right-most derivation for the string '1001'.

Solution

$S \underset{rm}{\Rightarrow} A1\underline{B}$ [Using $S \rightarrow A1B$]

$\underset{rm}{\Rightarrow} A10\underline{B}$ [Using $B \rightarrow 0B$]

$\underset{rm}{\Rightarrow} A100\underline{B}$ [Using $B \rightarrow 0B$]

$\underset{rm}{\Rightarrow} A1001\underline{B}$ [Using $B \rightarrow 1B$]

$\underset{rm}{\Rightarrow} \underline{A}1001$ [Using $B \rightarrow \varepsilon$]

$\underset{rm}{\Rightarrow} 1001$ [Using $A \rightarrow \varepsilon$]

- Thus the string '1001' is derived from the start symbol using right most derivation.
- The right most variable is expanded at each step.

Types of derivations

- There are two types of derivations,
    - **Leftmost derivations**
    - **Rightmost derivations**
- **Leftmost Derivations (LMD)**
    - If at each step in a derivation, a production is applied to the **leftmost variable**, then it is called leftmost derivation.
- **Rightmost Derivations (RMD)**
    - If at each step in a derivation, a production is applied to the **righrmost variable,** then it is called rightmost derivation.
- The **left most and right most derivations are equivalent to the left and right sentential forms.**

Example

Let G be the grammar. P = {S → aB | bA, A → a | aS | bAA, B → b | bs | aBB}. For the string aabbbbaa, find LMD and RMD.

Solution

Leftmost derivation

$S \underset{lm}{\Rightarrow} aB$

$\underset{lm}{\Rightarrow} aaBB$

$\underset{lm}{\Rightarrow} aabB$

$\underset{lm}{\Rightarrow} aabbS$

$\underset{lm}{\Rightarrow} aabbbA$

$\underset{lm}{\Rightarrow} aabbbbAA$

$\underset{lm}{\Rightarrow} aabbbbaA$

$\underset{lm}{\Rightarrow} aabbbbaa$

Rightmost derivation

$S \underset{rm}{\Rightarrow} aB$

$\underset{rm}{\Rightarrow} aaBB$

$\underset{rm}{\Rightarrow} aaBbS$

$\underset{rm}{\Rightarrow} aaBbbA$

$\underset{rm}{\Rightarrow} aaBbbbAA$

$\underset{rm}{\Rightarrow} aaBbbbAa$

$\underset{rm}{\Rightarrow} aaBbbbaa$

$\underset{rm}{\Rightarrow} aabbbbaa$

Home Work

- Derive the string "abb" for leftmost derivation and rightmost derivation using a CFG given by,

  S → AB | ε

  A → aB

  B → Sb

- Derive the string "aabbabba" for leftmost derivation and rightmost derivation using a CFG given by,

  S → aB | bA

  S → a | aS | bAA

  S → b | aS | aBB

- Derive the string "00101" for leftmost derivation and rightmost derivation using a CFG given by,

  S → A1B

  A → 0A | ε

  B → 0B | 1B | ε

Parser

- Parser is a compiler that is used to break the data into smaller elements coming from lexical analysis phase.
- A parser takes input in the form of sequence of tokens and produces output in the form of parse tree.
- Parser is that phase of compiler **which takes token string as input and with the help of existing grammar, converts it into the corresponding parse tree**. Parser is also known as Syntax Analyzer.
- CFG is a helpful tool in describing the syntax of programming languages.

Parse Tree

- Derivation tree is a **graphical representation for the derivation of the given production rules** for a given CFG.
- It is the simple way to show **how the derivation can be done to obtain some string from a given set of production rules**. The **derivation tree is also called a parse tree.**
- The tree representation provides a clear view of the expansions / substitutions done on the variables.
- The tree representation also facilitates easy conversion / translation of the source code in to executable code, when used in compilers.

Parse Tree

- A **set of derivations substituted / applied to generate a string can be represented using a tree**, called **parse tree.**
- Parse tree provides clear understanding of recursive function, grouping of symbols especially parenthesis balancing etc.
- The parse trees can be constructed by using the following **three conditions**.

  1. The **root node** of the tree being the **start symbol** and the **interior nodes** [non – leaf nodes] is represented by the **variable V**, of grammar G.

  2. The **leaf nodes** are labeled by either the **terminals T** or by ε . However if the leaf node is labeled ε, *then it must be the only child of its parent.*

  3. If the **interior node is labeled V1 where V1 ∈ V,** having its children, X1, X2, … , Xn are the production of V1 which are represented from **left to right**.

Yield of a parse tree

- The **terminals present at the leaf nodes are concatenated from left to right generate a string,** derived from the grammar.
- The **generated string** is called the **yield of a parse tree.**

Example

For the grammar S → 0S1|01, generate the derivation of the string '000111'.

Solution

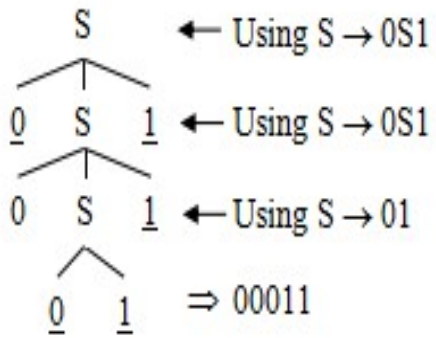
The derivation is given by

$S \Rightarrow 0S1$ [Using $S \rightarrow 0S1$]

$S \Rightarrow 00S11$ [Using $S \rightarrow 0S1$]

$S \Rightarrow 000111$ [Using $S \rightarrow 01$]

The parse tree is given by



- In the mentioned parse tree, the terminal symbols in the leaf nodes, taken from left to right generate the string, '**000111**'.
- Thus **000111 is the yield of the parse tree.**

Types of parse tree

- There are two types of parse tree. They are,
  - **Leftmost derivation tree / left derivation tree**
  - **Rightmost derivation tree / right derivation tree**
- **Leftmost derivation tree**
  - A derivation tree representing $A \Rightarrow \alpha$ is called a leftmost derivation tree if the production rule is applied only to the **leftmost variable** at every step.
- **Rightmost derivation tree**
  - A derivation / parse tree representing $A \Rightarrow \alpha$ is said to be a rightmost derivation tree if the production rule is applied only to the **rightmost variable** at every step.

For the grammar given below give the parse tree for leftmost and rightmost derivation of the string '1001'.

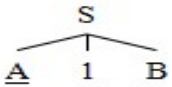
$S \rightarrow A1B$

$A \rightarrow 0A \mid \varepsilon$

$B \rightarrow 0B \mid 1B \mid \varepsilon$

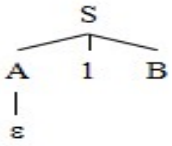
Left Most Derivation tree

Step 1



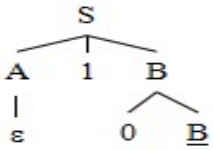
Using $S \rightarrow A1B$

Step 2



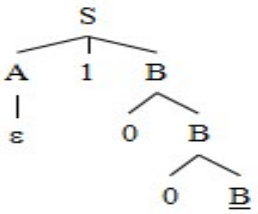
Using $A \rightarrow \varepsilon$

Step 3



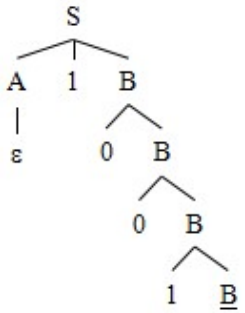
Using $B \rightarrow 0B$

Step 4



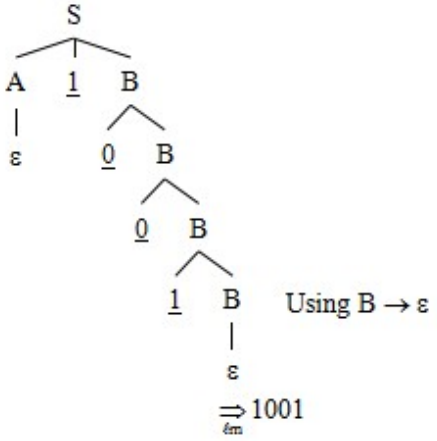
Using $B \rightarrow 0B$

Step 5



Using $B \rightarrow 1B$

Step 6

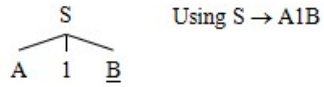


Using $B \rightarrow \varepsilon$

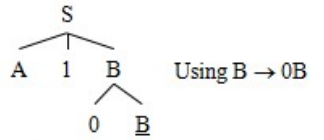
$\Rightarrow_{lm} 1001$

Rightmost Derivation Tree

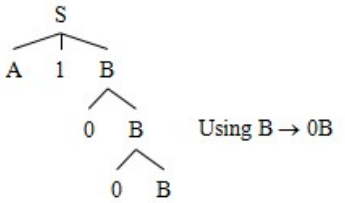
Step 1



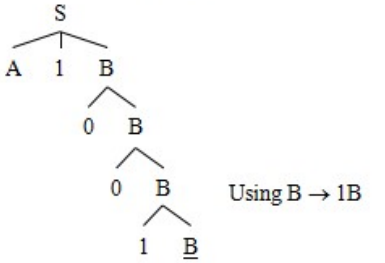
Step 2



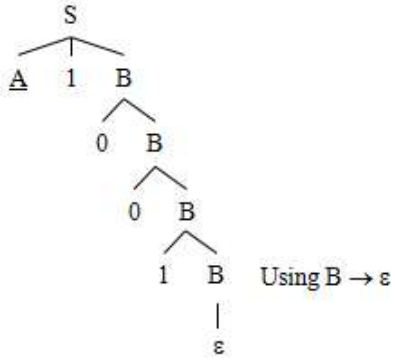
Step 3



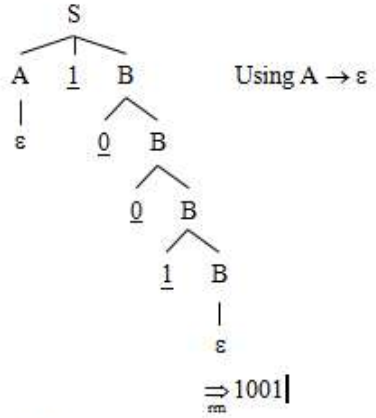
Step 4



Step 5



Step 6



Home Work

- Draw a derivation tree for the string "bab" from the CFG given by

  $S \rightarrow bSb \mid a \mid b$

- Construct a derivation tree for the string “aabbabba” for the CFG given by,

  $S \rightarrow aB \mid bA$

  $A \rightarrow a \mid aS \mid bAA$

  $B \rightarrow b \mid bS \mid aBB$

- Show the derivation tree for string "aabbbb" with the following grammar.

  $S \rightarrow AB \mid \varepsilon$

  $A \rightarrow aB$

  $B \rightarrow Sb$

Home Work

1) For the grammar, $S \rightarrow 0S1 \mid 01$, give the derivation of 000111.

2) Find whether S = aabbb is in L = L(G) where G is given by $S \rightarrow XY$; $X \rightarrow YY \mid a$; $Y \rightarrow XY \mid b$.

3) For the below given grammar, generate the derivation of (a+b)*a+b.

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid a \mid b$

4) Construct a parse tree for $S \Rightarrow$ aabbaa, given the productions: $S \rightarrow aAS|a$; $A \rightarrow SbA|SS|ba$.

5) Let G be given as, $P = \{S \rightarrow 0B|1A,\ A \rightarrow 0|0S|1AA,\ B \rightarrow 1|1S|0BB\}$. Find the rightmost derivation for '0110'.

Home Work

6) **Find the leftmost derivation for S = aaabbabbba.**
Given the productions

$S \rightarrow aB|bA$

$A \rightarrow aS|bAA|a$

$B \rightarrow bS|aBB|b$

7) **Let G be the grammar, S → 0B|1A, A → 0|0S|1AA, B → 1|1S|0BB. For S=00110101, find the leftmost derivation.**

8) **Find the derivation tree of a*b + a*b where G is given by S → S + S | S * S | a | b.**

Ambiguity in Grammar

- A grammar is said to be ambiguous if there exists **more than one leftmost derivation or more than one rightmost derivation or more than one parse tree for the given input string.**
- **A grammar, G is said to be ambiguous if there exists two different parse trees for at least one string w where w ∈ T*, each parse tree with the same symbol.**
- **If each string w has at most one parse tree in the grammar, then it is unambiguous grammar.**

Ambiguity in Grammar

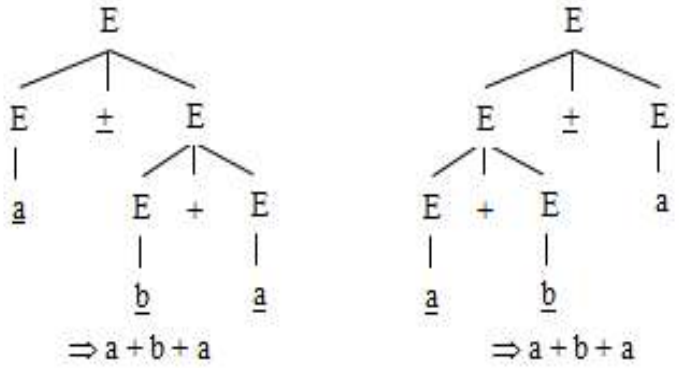
Example: Ambiguous grammar

Consider the grammar given below, prove that it is ambiguous.

$$E \rightarrow E + E \mid a \mid b$$

Solution

Consider a + b + a, then the parse trees are derived as,



- Since there are two different parse trees with same start symbol, leads to the same string, ω ε T*, the grammar is ambiguous.

Test whether the following grammars are ambiguous.

(a) S → 0S1S | 1S0S | ε

(b) S → AA

A → aAb | bAa | ε

Test whether the following grammars are ambiguous.

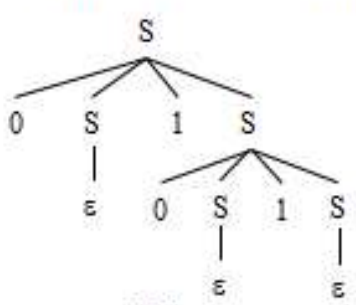
(a) S → 0S1S | 1S0S | ε

(b) S → AA

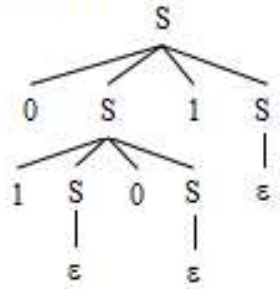
A → aAb | bAa | ε

Solution

(a) Consider S = 0101, the parse tree for LMD and RMD are given as,



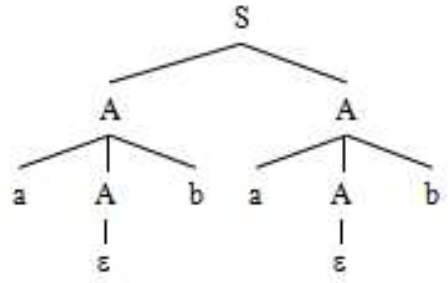
⇒ 0101



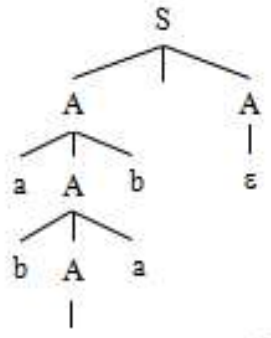
⇒ 0101

Since there are two different parse trees for the same string, the grammar is ambiguous.

(b) Consider S = abab, the parse trees are,



⇒ abab



⇒ abab

There are two different parse trees for the same string S = abab. Hence the grammar is ambiguous.

Home Work

1) **Show that $S \rightarrow a \mid Sa \mid bSS \mid SSb \mid SbS$ is ambiguous.**

2) **Show that the given grammar is ambiguous**

$S \rightarrow a \mid abSb \mid aAb$

$A \rightarrow bS \mid aAAb$

3) **If G is the grammar defined by, $S \rightarrow SbS \mid a$. Show that G is ambiguous.**

Show the derivation steps and construct derivation tree for the string 'ababbb'

by using leftmost derivation with the grammar

$S \rightarrow AB \mid \epsilon$

$A \rightarrow aB$

$B \rightarrow Sb$

Consider the context-free grammar (CFG) given below. Give the leftmost derivation for the string *bbaa* using the grammar.

$S \rightarrow bS \mid aT \mid \in$

$T \rightarrow aT \mid bU \mid \in$

$U \rightarrow aT \mid \in$

Derive a string 'aababa' for the following Context Free Grammar (CFG).

S → aSX / b;

X → Xb / a

Show that the following grammar is ambiguous: $S \rightarrow SbS \mid a$.

Consider the string S = ababa

1. Define Context Free Grammar? Give Example.
2. Give the Chomsky Hierarchy of Grammars.
3. Construct a CFG for L = $\{a^n \mid n \text{ is odd}\}$.
4. Construct the CFG representing the set of palindromes over $(0 + 1)^*$.
5. Construct a CFG for set of strings that contain equal number of a's and b's over the alphabets (a,b).
6. Construct a CFG for a language L = $\{wcw^R \mid \text{where } w \in (a, b)^*\}$.
7. Define CFL. Give Example.
8.

Design the language for the CFG given. The grammar, G = ({S}, {0,1}, P, {S}) where

P: S → 0S1

S → 01

9.What do u meant by derivation?
10.What is Parse tree? Give Example.
11.When do u say CFG is Ambiguous?