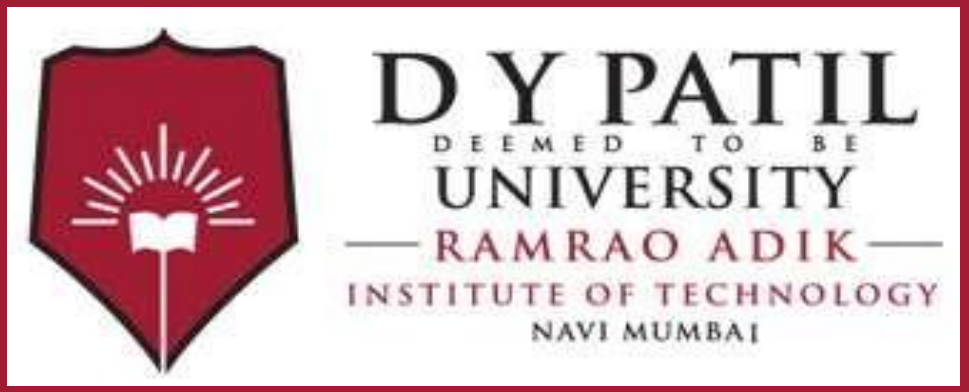


Theoretical Computer Science

Unit 3: Grammars

Faculty Name : Ms. Namita D. Pulgam

Index

Lecture No 22: Introduction to Grammar and Derivation of a Grammar

Introduction to Grammar

- Finite Automata M accepts a language L and regular language can be described by Regular Expression.
- If language is not regular, it can not be represented using finite automata or a regular expression.
- These languages are represented using set of equations i.e. grammar.
- Grammar for particular language is defined as a finite set of formal rules for generating syntactically correct sentences i.e. Grammar defines syntax of a language.
- The language may be a formal programming language such as C, C++ and Java or natural language such as English or French.

Introduction to Grammar (Cont..)

- Grammar consists of two types of symbols namely: Terminal and Non-terminal
- Terminals are part of generated sentences and non terminals are used to form a sentence or string.
- Finite automata are defined over set of input symbols and have set of states and transitions, whereas grammar defined over set of terminals and have set of variables and productions.
- For example: If we want to generate an English statement ‘dog runs’, we may use the following rules:

<sentence> → <noun> <verb>

<noun> → dog

<verb> → runs

Here dog and runs are the terminal symbols. ‘Sentence’, ’noun’ and ‘verb’ are non terminals.

- This describes how the sentence of the form ‘noun’ followed by ‘verb’ can be generated.

Context Free Grammar

- Grammar is a set of rules which check whether a string belong to a particular language or not.
- It is a set of recursive rewriting rules (productions) used to generate patterns of strings having set of terminal symbols.
- Terminal symbols are the characters of the alphabet that appear in the strings generated by the grammar.
- It is called as context free because any of the production rules in the grammar can be applied regardless of context.
- CFG's are useful in many applications
  - Describing syntax of programming languages
  - Parsing
  - Structure of documents, e.g. XML

Formal Definition of CFG

- There is a finite set of symbols that form the strings, i.e. there is a finite alphabet.
- The alphabet symbols are called **terminals.**
- There is a finite set of **variables**, sometimes called non-terminals or syntactic categories. Each variable represents a language (i.e. a set of strings).
- One of the variables is the **start symbol**. Other variables may exist to help define the language.
- There is a finite set of **productions** or production rules that represent the recursive definition of the language. Each production is defined:
    1. Has a single variable that is being defined to the left of the production
    2. Has the production symbol →
    3. Has a string of zero or more terminals or variables, called the body of the production.

Formal Definition of CFG (Cont..)

- A Context Free Grammar G is denoted by a quadruple (V, T, S, P) where −

V : Finite set of non terminals

T : Finite set of terminal

S : S is a non terminal, usually denotes starting symbol

P : Finite set of Productions rules

Grammar Notations

- Non-terminals are usually denoted by capital letters: A,B, C, D etc.
- Terminals are denoted by small letters : a, b, c etc.
- Strings of terminals are usually denoted by u, v, w, x, y, z etc.
- The arrow symbol '→' stands for production. **For example :** A production denoted by 'A → b' indicates 'A produces b'
- If the same non terminal has multiple productions i.e. there is more than one rule for the same non terminal we use '|' (or) to represent the grammar in a simplified form.

Example :

- Consider G = { (E), (+, *, a) , P, E}
- Where P consists of the following productions :

P = { E → E+E

E → E*E

E → a }

- All three productions mentioned are for the non-terminal E.
- All three productions have the same left hand side.
- We can combine these with the help of '|' operator

P = { E → E+E | E*E | a }

Derivation of a Grammar

- In a given grammar how any string can be derived from a start symbol is represented using derivations.
- There are two types of derivations :
    1. Left most Derivation
    2. Right most Derivation

Leftmost Derivation

- If at each step in a derivation a production rule is applied to the leftmost variable or non-terminal then the derivation is called leftmost derivation.
- It generates any string from left to right. Hence it is mainly applicable for top-down parsing.

Rightmost Derivation :

- If at each step in a derivation a production rule is applied to the rightmost variable or non-terminal then the derivation is called rightmost derivation.

Example 1:

Q. Consider the following CFG : G = { (S, A), (a, b), P, S }

where P consists of : S → aAS | a

A → SbA | SS | ba

Derive string 'aabbaa' using Leftmost derivation and Rightmost derivation.

Solution :

In given example S is the start symbol.

Let us number the productions as:

1. S → aAS
2. S → a
3. A → SbA
4. A → SS
5. A → ba

Example 1: (Cont..)

Leftmost Derivation for String ‘aabbaa’ :

$S \Rightarrow aAS$	(Using Rule 1 i.e. S → aAS)
$S \Rightarrow aSbAS$	(Using Rule 3 i.e. A → SbA)
$S \Rightarrow aabAS$	(Using Rule 2 i.e. S → a)
$S \Rightarrow aabbaS$	(Using Rule 5 i.e. A → ba)
$S \Rightarrow aabbaa$	(Using Rule 2 i.e. S → a)

Example 1: (Cont..)

Rightmost Derivation for String ‘aabbaa’ :

$S \Rightarrow aAS$	(Using Rule 1 i.e. S → aAS)
$S \Rightarrow aAa$	(Using Rule 2 i.e. S → a)
$S \Rightarrow aSbAa$	(Using Rule 3 i.e. A → SbA)
$S \Rightarrow aSbbaa$	(Using Rule 5 i.e. A → ba)
$S \Rightarrow aabbaa$	(Using Rule 2 i.e. S → a)

Example 2:

Q. Write LMD and RMD for the string 'bbaaba' for the grammar G which is defined as:

S → aB | bA

A → a | aS | bAA

B → b | bS | aBB where S is the starting symbol.

Solution :

Let us number the productions as:

1. S → aB
2. S → bA
3. A → a
4. A → aS
5. A → bAA
6. B → b
7. B → bS
8. B → aBB

Example 2: (Cont..)

Leftmost Derivation for String ‘bbaaba’ :

$S \Rightarrow bA$	(Using Rule 2 i.e. S → bA)
$S \Rightarrow bbAA$	(Using Rule 5 i.e. A → bAA)
$S \Rightarrow bbaSA$	(Using Rule 4 i.e. A → aS)
$S \Rightarrow bbaaBA$	(Using Rule 1 i.e. S → aB)
$S \Rightarrow bbaabA$	(Using Rule 6 i.e. B → b)
$S \Rightarrow bbaaba$	(Using Rule 3 i.e. A → a)

Example 2: (Cont..)

Rightmost Derivation for String ‘bbaaba’ :

$S \Rightarrow bA$	(Using Rule 2 i.e. S → bA)
$S \Rightarrow bbAA$	(Using Rule 5 i.e. A → bAA)
$S \Rightarrow bbAa$	(Using Rule 3 i.e. A → a)
$S \Rightarrow bbaSa$	(Using Rule 4 i.e. A → aS)
$S \Rightarrow bbaaBa$	(Using Rule 1 i.e. S → aB)
$S \Rightarrow bbaaba$	(Using Rule 6 i.e. B → b)

Derivation/ Parse Tree

- Derivation tree is a graphical representation or description of how the sentence (or string) has been derived or generated using a grammar.

- There exists a derivation tree for every string derivable from the start symbol.

- It is also known as parse tree, syntax tree or rule tree.

Example 3:

Q. Consider the following CFG : G = { (S, X), (a, b), P, S }

where productions are : S → aSX | b

X → Xb | a

Find i) LMD, ii) RMD and iii) Parse Tree for 'aababa'.

Solution :

Let us number the productions as:

1. S → aSX
2. S → b
3. X → Xb
4. X → a

Example 3: (Cont..)

Leftmost Derivation for String ‘aababa’ :

$S \Rightarrow aSX$	(Using Rule 1 i.e. S → aSX)
$S \Rightarrow aaSXX$	(Using Rule 1 i.e. S → aSX)
$S \Rightarrow aabXX$	(Using Rule 2 i.e. S → b)
$S \Rightarrow aabXbX$	(Using Rule 3 i.e. X → Xb)
$S \Rightarrow aababX$	(Using Rule 4 i.e. X → a)
$S \Rightarrow aababa$	(Using Rule 4 i.e. X → a)

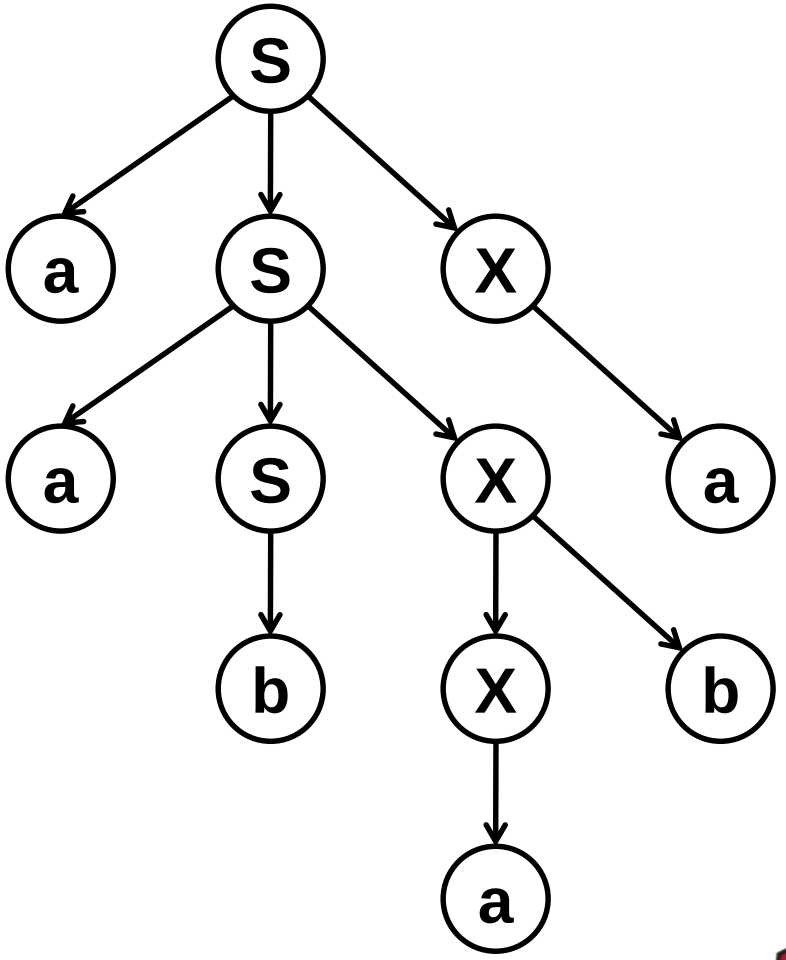
Example 3: (Cont..)

Rightmost Derivation for String ‘aababa’ :

$S \Rightarrow aSX$	(Using Rule 1 i.e. S → aSX)
$S \Rightarrow aSa$	(Using Rule 4 i.e. X → a)
$S \Rightarrow aaSXa$	(Using Rule 1 i.e. S → aSX)
$S \Rightarrow aaSXba$	(Using Rule 3 i.e. X → Xb)
$S \Rightarrow aaSaba$	(Using Rule 4 i.e. X → a)
$S \Rightarrow aababa$	(Using Rule 2 i.e. S → b)

Example 3 : (Cont..)

Parse Tree :



Example 4:

Q. Let G be the grammar, Find i) LMD, ii) RMD and iii) Parse Tree for the string '00110101'.

where the productions are : S → 0B | 1A

A → 0 | 0S | 1AA

B → 1 | 1S | 0BB

Solution :

Let us number the productions as:

1. S → 0B
2. S → 1A
3. A → 0
4. A → 0S
5. A → 1AA
6. B → 1
7. B → 1S
8. B → 0BB

Example 4: (Cont..)

Leftmost Derivation for String '00110101' :

$S \Rightarrow 0B$	(Using Rule 1 i.e. S → 0B)
$S \Rightarrow 00BB$	(Using Rule 8 i.e. B → 0BB)
$S \Rightarrow 001SB$	(Using Rule 7 i.e. B → 1S)
$S \Rightarrow 0011AB$	(Using Rule 2 i.e. S → 1A)
$S \Rightarrow 00110SB$	(Using Rule 4 i.e. A → 0S)
$S \Rightarrow 001101AB$	(Using Rule 2 i.e. S → 1A)
$S \Rightarrow 0011010B$	(Using Rule 3 i.e. A → 0)
$S \Rightarrow 00110101$	(Using Rule 6 i.e. B → 1)

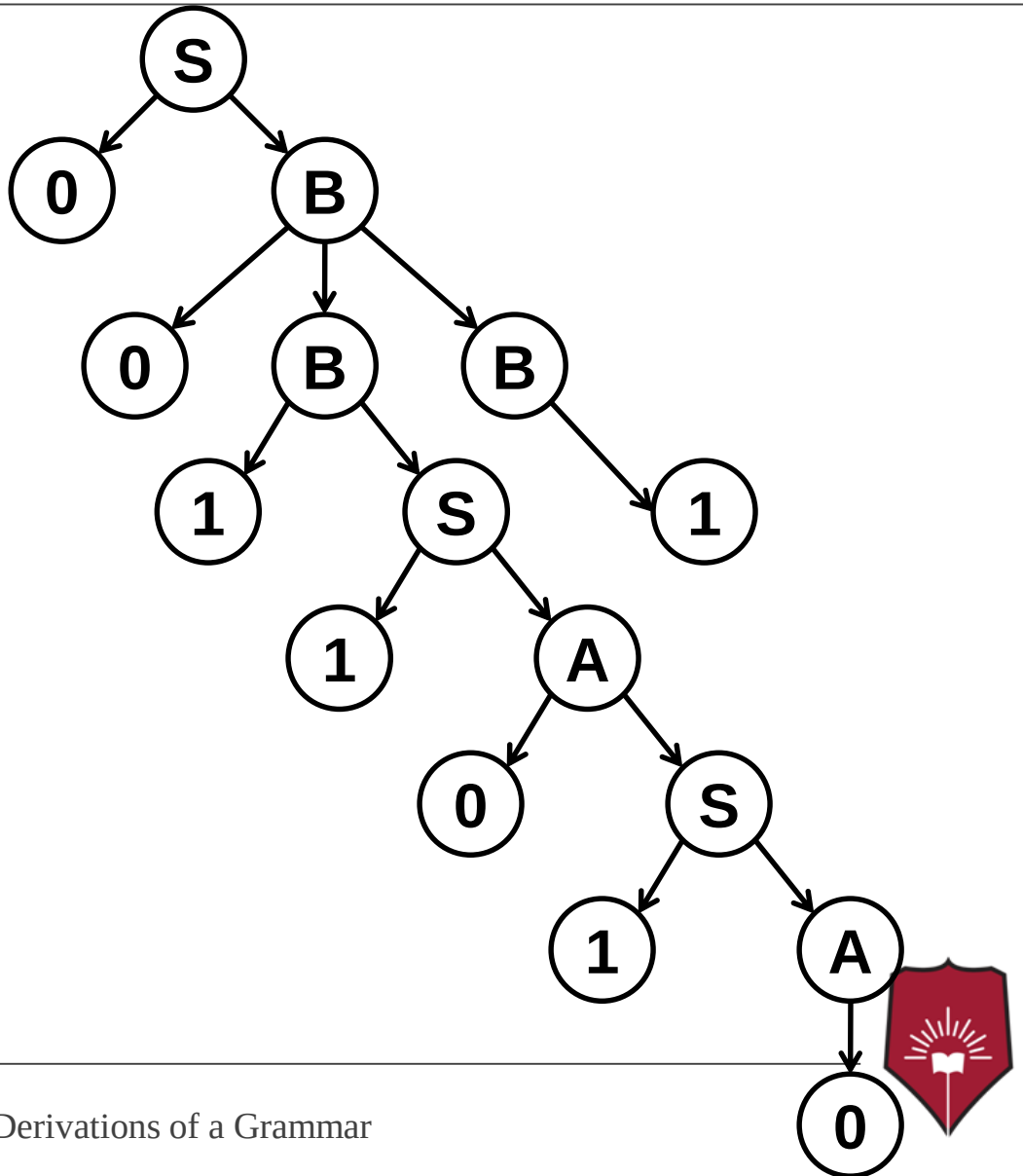
Example 4: (Cont..)

Rightmost Derivation for String '00110101' :

$S \Rightarrow 0B$	(Using Rule 1 i.e. S → 0B)
$S \Rightarrow 00BB$	(Using Rule 8 i.e. B → 0BB)
$S \Rightarrow 00B1$	(Using Rule 6 i.e. B → 1)
$S \Rightarrow 001S1$	(Using Rule 7 i.e. B → 1S)
$S \Rightarrow 0011A1$	(Using Rule 2 i.e. S → 1A)
$S \Rightarrow 00110S1$	(Using Rule 4 i.e. A → 0S)
$S \Rightarrow 001101A1$	(Using Rule 2 i.e. S → 1A)
$S \Rightarrow 00110101$	(Using Rule 3 i.e. A → 0)

Example 4 : (Cont..)

Parse Tree :



Example 5:

Q. Let G be the grammar, Find i) LMD, ii) RMD and iii) Derivation Tree for the expression 'a*b+a*b'.

where the productions are : S → S+S | S*S

S → a | b

Solution :

Let us number the productions as:

1. S → S+S
2. S → S*S
3. S → a
4. S → b

Example 5: (Cont..)

Leftmost Derivation for String ‘a*b+a*b’ :

$S \Rightarrow S+S$	(Using Rule 1 i.e. S → S+S)
$S \Rightarrow S*S+S$	(Using Rule 2 i.e. S → S*S)
$S \Rightarrow a*S+S$	(Using Rule 3 i.e. S → a)
$S \Rightarrow a*b+S$	(Using Rule 4 i.e. S → b)
$S \Rightarrow a*b+S*S$	(Using Rule 2 i.e. S → S*S)
$S \Rightarrow a*b+a*S$	(Using Rule 3 i.e. S → a)
$S \Rightarrow a*b+a*b$	(Using Rule 4 i.e. S → b)

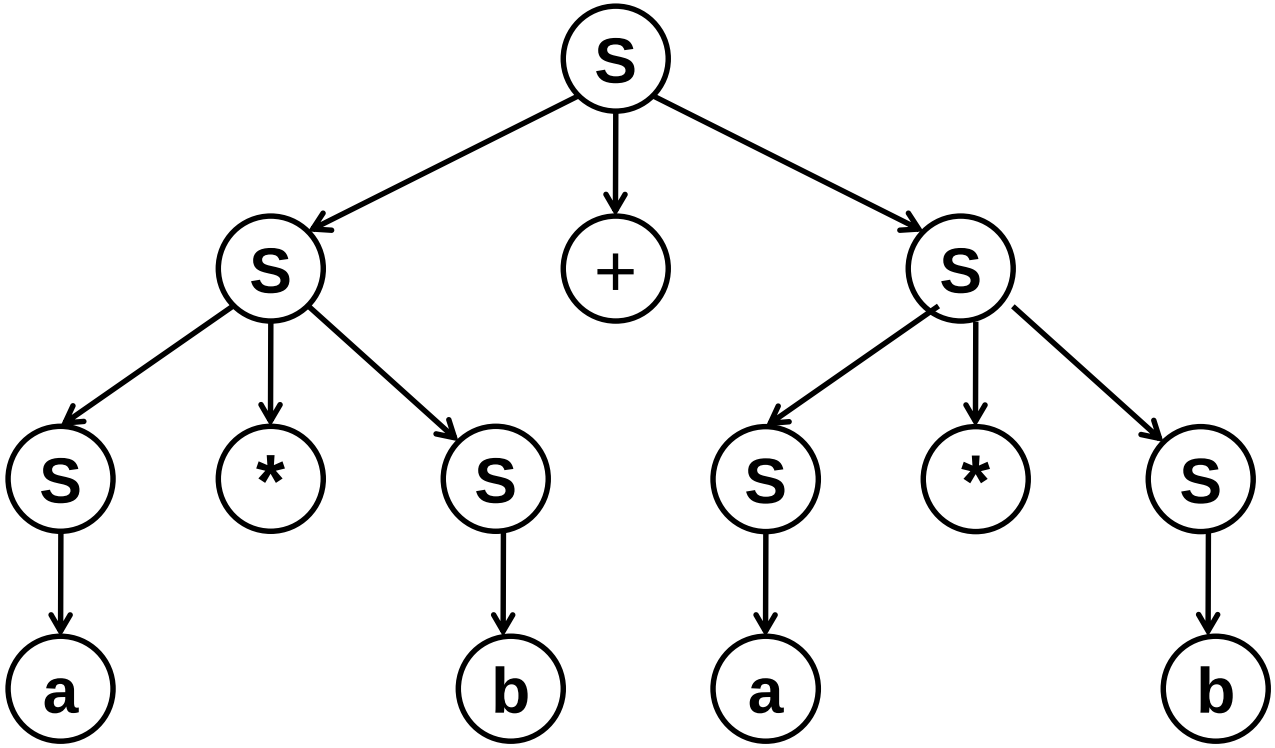
Example 5: (Cont..)

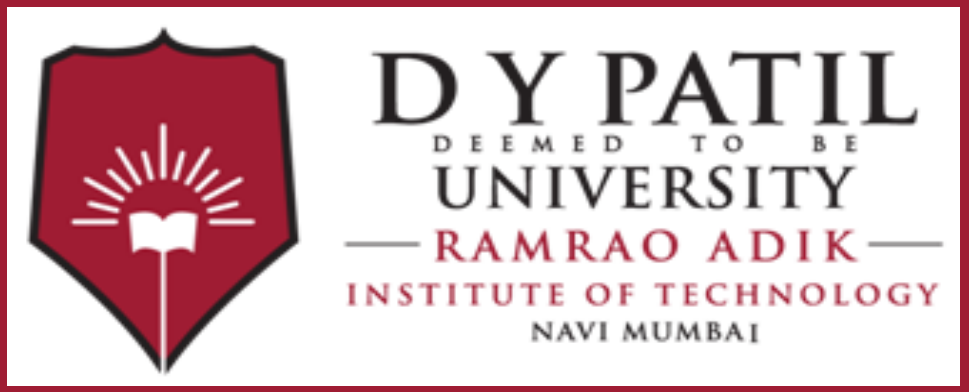
Rightmost Derivation for String ‘a*b+a*b’ :

$S \Rightarrow S+S$	(Using Rule 1 i.e. S → S+S)
$S \Rightarrow S+S*S$	(Using Rule 2 i.e. S → S*S)
$S \Rightarrow S+S*b$	(Using Rule 4 i.e. S → b)
$S \Rightarrow S+a*b$	(Using Rule 3 i.e. S → a)
$S \Rightarrow S*S+a*b$	(Using Rule 2 i.e. S → S*S)
$S \Rightarrow S*b+a*b$	(Using Rule 4 i.e. S → b)
$S \Rightarrow a*b+a*b$	(Using Rule 3 i.e. S → a)

Example 5 : (Cont..)

Derivation Tree :




D Y PATIL
DEEMED TO BE
UNIVERSITY
RAMRAO ADIK
INSTITUTE OF TECHNOLOGY
NAVI MUMBAI

Thank You

Lecture No 23: Ambiguous Context Free Grammar

Ambiguous Context Free Grammar

- A CFG is ambiguous if one or more terminal strings have multiple leftmost derivations from the start symbol.

- Equivalently: multiple rightmost derivations, or multiple parse trees.

Example 6:

Q. Consider the following grammar:

E → E+E | E*E | id

Find :

i) LMD

ii) RMD

iii) Derivation Tree for the expression 'id+id*id'

iv) Check grammar is ambiguous or not.

Solution :

Let us number the productions as:

1. E → E+E
2. E → E*E
3. E → id

Example 6: (Cont..)

i) Leftmost Derivation for String ‘id+id*id’ :

$E \Rightarrow E+E$	(Using Rule 1 i.e. E → E+E)
$E \Rightarrow id+E$	(Using Rule 3 i.e. E → id)
$E \Rightarrow id+E*E$	(Using Rule 2 i.e. E → E*E)
$E \Rightarrow id+id*E$	(Using Rule 3 i.e. E → id)
$E \Rightarrow id+id*id$	(Using Rule 3 i.e. E → id)

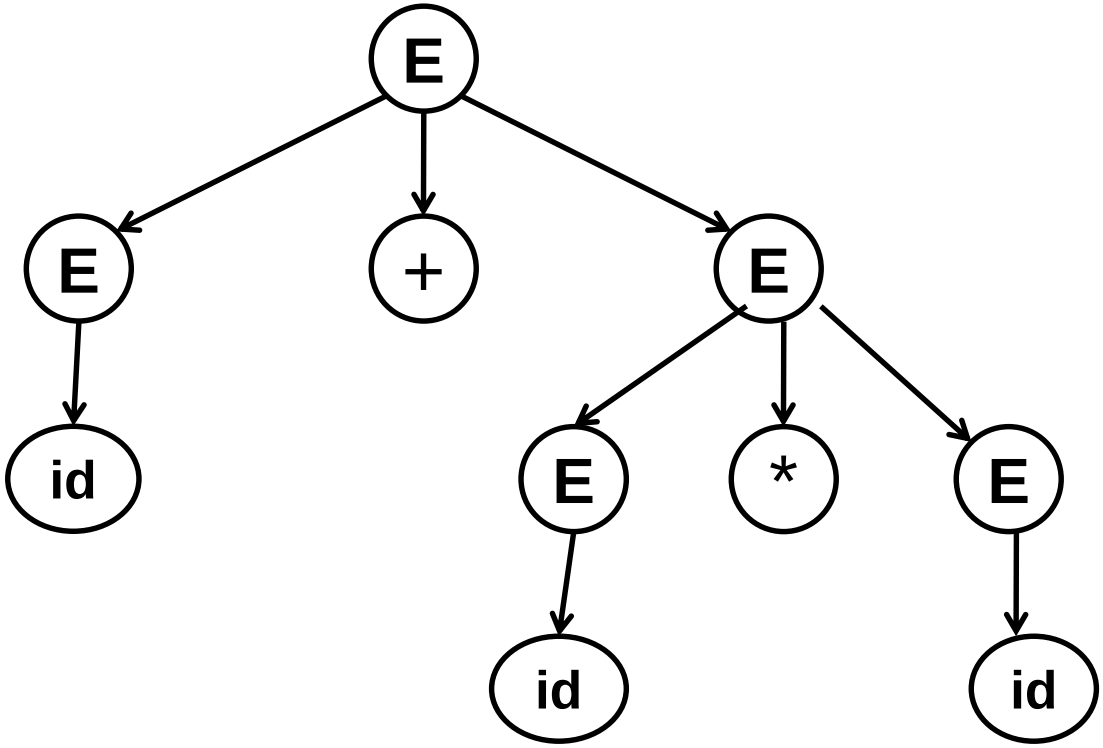
Example 6: (Cont..)

ii) Rightmost Derivation for String ‘id+id*id’ :

$E \Rightarrow E+E$	(Using Rule 1 i.e. E → E+E)
$E \Rightarrow E+E*E$	(Using Rule 2 i.e. E → E*E)
$E \Rightarrow E+E*id$	(Using Rule 3 i.e. E → id)
$E \Rightarrow E+id*id$	(Using Rule 3 i.e. E → id)
$E \Rightarrow id+id*id$	(Using Rule 3 i.e. E → id)

Example 6 : (Cont..)

iii) Derivation Tree :



Example 6: (Cont..)

iv) Ambiguity Check :

Find other way of deriving Leftmost or Rightmost derivation.

Leftmost Derivation :

$E \Rightarrow E*E$	(Using Rule 2 i.e. E → E*E)
$E \Rightarrow E+E*E$	(Using Rule 1 i.e. E → E+E)
$E \Rightarrow id+E*E$	(Using Rule 3 i.e. E → id)
$E \Rightarrow id+id*E$	(Using Rule 3 i.e. E → id)
$E \Rightarrow id+id*id$	(Using Rule 3 i.e. E → id)

Example 6: (Cont..)

iv) Ambiguity Check :

Find other way of deriving Leftmost or Rightmost derivation.

Rightmost Derivation :

$E \Rightarrow E*E$	(Using Rule 2 i.e. E → E*E)
$E \Rightarrow E*id$	(Using Rule 3 i.e. E → id)
$E \Rightarrow E+E*id$	(Using Rule 1 i.e. E → E+E)
$E \Rightarrow E+id*id$	(Using Rule 3 i.e. E → id)
$E \Rightarrow id+id*id$	(Using Rule 3 i.e. E → id)

We can also draw different parse tree / derivation tree to depict the same string.

There are two different ways of deriving string "id+id*id".

Hence the given grammar is ambiguous.

Example 7:

Q. Let G be the grammar. Find the leftmost derivation, rightmost derivation and parse tree for the string 'bbaaabbaba'.

S → aB | bA

A→ a | aS | bAA

B → b | bS | aBB

check grammar is ambiguous or not.

Solution :

Let us number the productions as:

1. S → aB
2. S → bA
3. A → a
4. A → aS
5. A → bAA
6. B → b
7. B → bS
8. B → aBB

Example 7: (Cont..)

i) Leftmost Derivation for String ‘bbaaabbaba’ :

$S \Rightarrow bA$	(Using Rule 2 i.e. S → bA)
$S \Rightarrow bbAA$	(Using Rule 5 i.e. A → bAA)
$S \Rightarrow bbaSA$	(Using Rule 4 i.e. A → aS)
$S \Rightarrow bbaaBA$	(Using Rule 1 i.e. S → aB)
$S \Rightarrow bbaaaBBA$	(Using Rule 8 i.e. B → aBB)
$S \Rightarrow bbaaabSBA$	(Using Rule 7 i.e. B → bS)
$S \Rightarrow bbaaabbABA$	(Using Rule 2 i.e. S → bA)
$S \Rightarrow bbaaabbaBA$	(Using Rule 3 i.e. A → a)
$S \Rightarrow bbaaabbabA$	(Using Rule 6 i.e. B → b)
$S \Rightarrow bbaaabbaba$	(Using Rule 3 i.e. A → a)

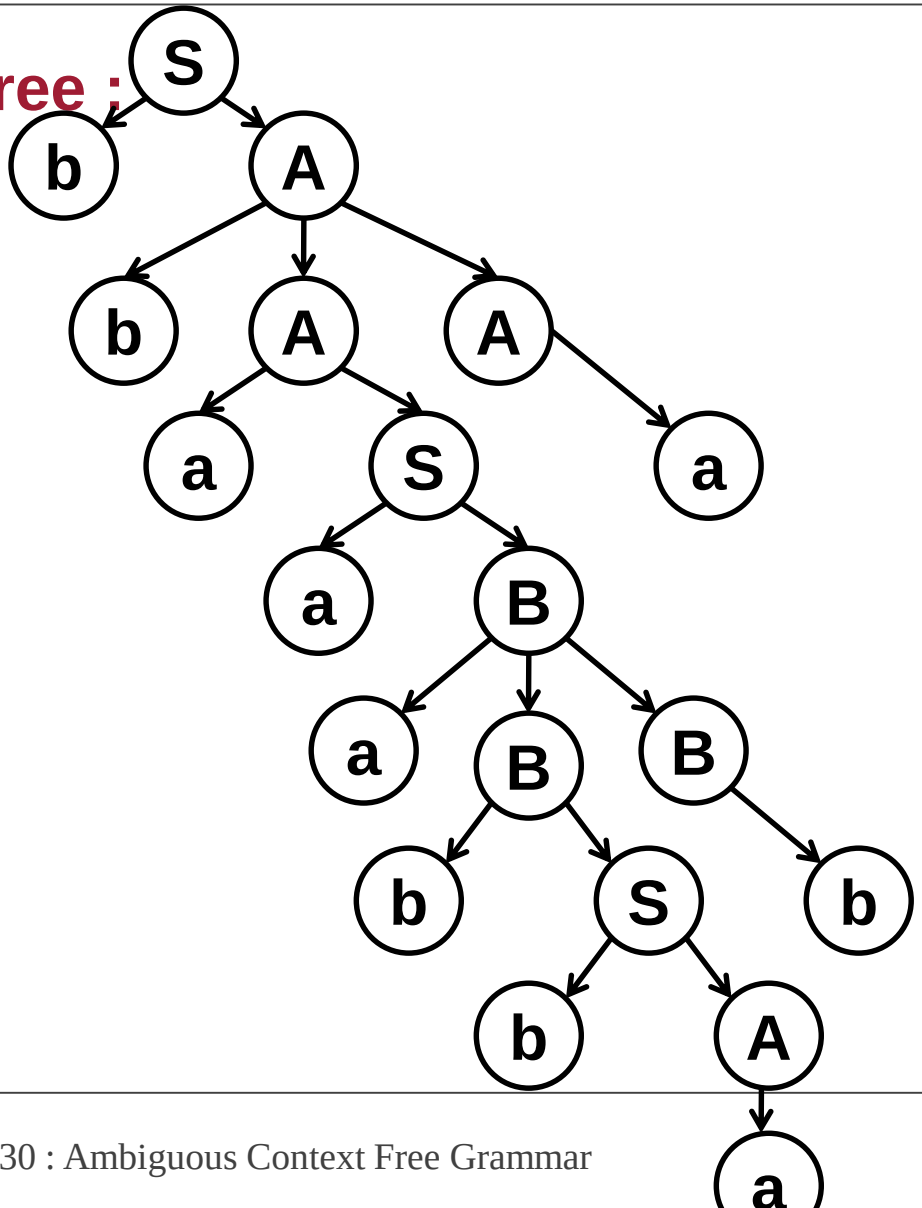
Example 7: (Cont..)

ii) Rightmost Derivation for String 'bbaaabbaba' :

$S \Rightarrow bA$	(Using Rule 2 i.e. S → bA)
$S \Rightarrow bbAA$	(Using Rule 5 i.e. A → bAA)
$S \Rightarrow bbAa$	(Using Rule 3 i.e. A → a)
$S \Rightarrow bbaSa$	(Using Rule 4 i.e. A → aS)
$S \Rightarrow bbaaBa$	(Using Rule 1 i.e. S → aB)
$S \Rightarrow bbaaaBBa$	(Using Rule 8 i.e. B → aBB)
$S \Rightarrow bbaaaBba$	(Using Rule 6 i.e. B → b)
$S \Rightarrow bbaaabSba$	(Using Rule 7 i.e. B → bS)
$S \Rightarrow bbaaabbAba$	(Using Rule 2 i.e. S → bA)
$S \Rightarrow bbaaabbaba$	(Using Rule 3 i.e. A → a)

Example 7 : (Cont..)

iii) Parse Tree :



Example 7: (Cont..)

iv) Ambiguity Check : Find other way of deriving Leftmost or Rightmost derivation.

Leftmost Derivation :

$S \Rightarrow bA$	(Using Rule 2 i.e. S → bA)
$S \Rightarrow bbAA$	(Using Rule 5 i.e. A → bAA)
$S \Rightarrow bbaA$	(Using Rule 3 i.e. A → a)
$S \Rightarrow bbaaS$	(Using Rule 4 i.e. A → aS)
$S \Rightarrow bbaaaB$	(Using Rule 1 i.e. S → aB)
$S \Rightarrow bbaaabS$	(Using Rule 7 i.e. B → bS)
$S \Rightarrow bbaaabbA$	(Using Rule 2 i.e. S → bA)
$S \Rightarrow bbaaabbaS$	(Using Rule 4 i.e. A → aS)
$S \Rightarrow bbaaabbabA$	(Using Rule 2 i.e. S → bA)
$S \Rightarrow bbaaabbaba$	(Using Rule 3 i.e. A → a)

- We are able to derive one more leftmost derivation.
- **Hence given grammar is ambiguous.**

Example 8:

Q. Consider the following grammar:

S → iCtS | iCtSeS | a

C → b

For the string 'ibtibtaea' find following: i) leftmost derivation, ii) rightmost derivation, iii) parse tree and iv) check grammar is ambiguous or not.

Solution :

Let us number the productions as:

1. S → iCtS
2. S → iCtSeS
3. S → a
4. C → b

Example 8: (Cont..)

i) Leftmost Derivation for String ‘ibtibtaea’ :

$S \Rightarrow iCtSeS$	(Using Rule 2 i.e. S → iCtSeS)
$S \Rightarrow ibtSeS$	(Using Rule 4 i.e. C → b)
$S \Rightarrow ibtiCtSeS$	(Using Rule 1 i.e. S → iCtS)
$S \Rightarrow ibtibtSeS$	(Using Rule 4 i.e. C → b)
$S \Rightarrow ibtibtaeS$	(Using Rule 3 i.e. S → a)
$S \Rightarrow ibtibtaea$	(Using Rule 3 i.e. S → a)

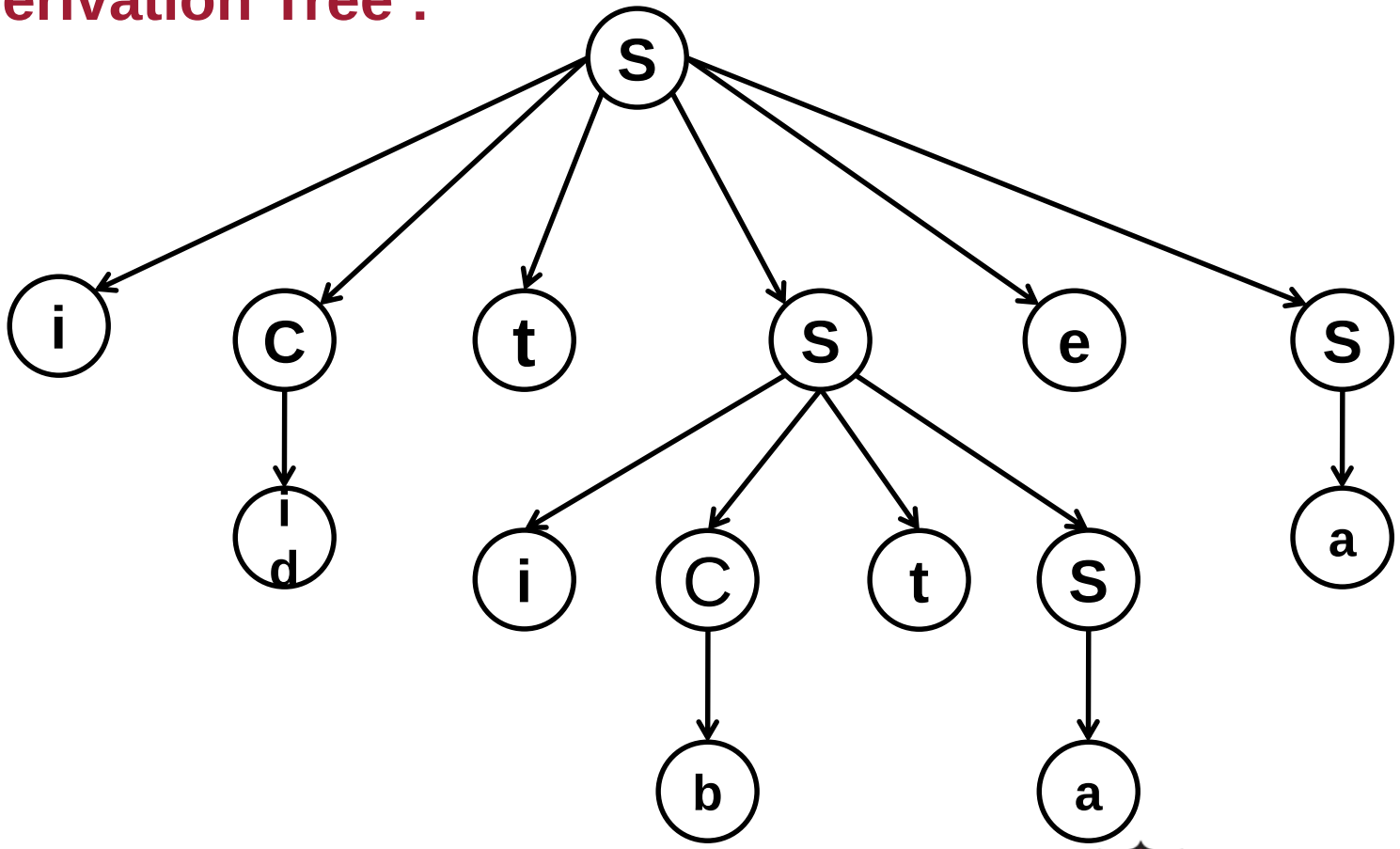
Example 8: (Cont..)

ii) Righttmost Derivation for String ‘ibtibtaea’ :

$S \Rightarrow iCtSeS$	(Using Rule 2 i.e. S → iCtSeS)
$S \Rightarrow iCtSea$	(Using Rule 3 i.e. S → a)
$S \Rightarrow iCtiCtSea$	(Using Rule 1 i.e. S → iCtS)
$S \Rightarrow iCtiCtaea$	(Using Rule 3 i.e. S → a)
$S \Rightarrow iCtibtaea$	(Using Rule 4 i.e. C → b)
$S \Rightarrow ibtibtaea$	(Using Rule 4 i.e. C → b)

Example 8 : (Cont..)

iii) Derivation Tree :



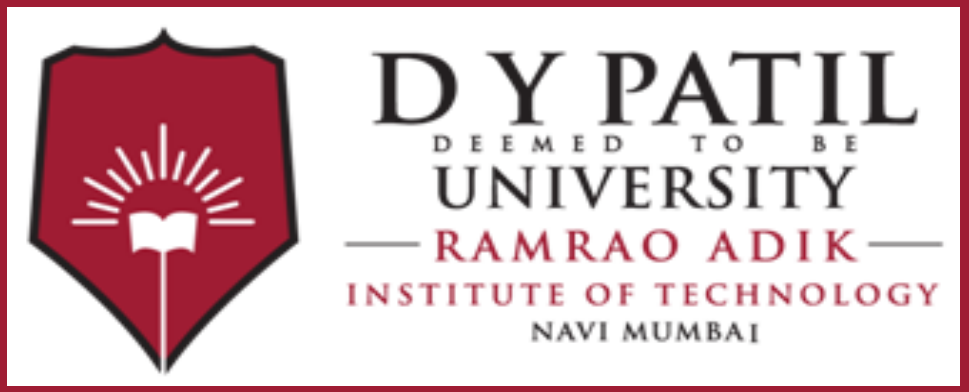
Example 8: (Cont..)

iv) Ambiguity Check : Find other way of deriving Leftmost or Rightmost derivation for the given string 'ibtibtaea'.

Leftmost Derivation :

$S \Rightarrow iCtS$	(Using Rule 1 i.e. S → iCtS)
$S \Rightarrow ibtS$	(Using Rule 4 i.e. C → b)
$S \Rightarrow ibtiCtSeS$	(Using Rule 2 i.e. S → iCtSeS)
$S \Rightarrow ibtibtSeS$	(Using Rule 4 i.e. C → b)
$S \Rightarrow ibtibtaeS$	(Using Rule 3 i.e. S → a)
$S \Rightarrow ibtibtaea$	(Using Rule 3 i.e. S → a)

- We are able to derive one more leftmost derivation for the given string **'ibtibtaea'**.
- **Hence given grammar is ambiguous.**


D Y PATIL
DEEMED TO BE
UNIVERSITY
RAMRAO ADIK
INSTITUTE OF TECHNOLOGY
NAVI MUMBAI

Thank You

Lecture No 24:
Simplification of CFG

Simplification of CFG

- In a CFG, it may happen that all the production rules and symbols are not needed for the derivation of strings.
- There may be some null productions and unit productions and eliminating these productions and symbols is called simplification of CFGs.
- Grammar can be simplified by :
  - Eliminating useless symbols
  - Eliminating unit productions
  - Eliminating null productions

Removal of Useless Symbols

- A variable is useless if it is not deriving any string of terminals or it is not present in sentential form that means it is not reachable from start symbol.

- If there are useless symbols in a grammar then all the productions in which these variables are present should be detected.

Steps to Eliminate Useless Symbols

1. Identify all variables which are not deriving any string of terminals.
2. Delete all the productions in which these variables are present.
3. Identify the variables which are not present in sentential form or not reachable from start symbol.
4. Delete all the productions in which these variables are present.

Example 1:

Q. Consider the following grammar : G = { { S, A }, { 1, 0 }, P, S }

where P consists of : S → 10 | 0S1 | 1S0 | A | SS

Solution :

- In given example a non terminal A is not deriving any string.
- So A is useless symbol and we should remove it.
- To remove A we should delete the production S → A.
- Simplified or Reduced Grammar is :

S → 10 | 0S1 | 1S0 | SS

Example 2:

Q. Consider the following grammar : G = { { S, A, B }, { a }, P, S }
where P consists of : S → AB | a
A → a

Solution :

- In given example a non terminal B is not deriving any string.
- So B is useless symbol and we should remove it.
- To remove B we should delete the production S → AB and we get S → a
  and A → a
- A is also useless because it is not present in sentential form.
- Hence we should remove production in which A is present.
- Simplified or Reduced Grammar is :

S → a

Removal of Unit Production

- A production of the form, A → B where A and B both are non terminals is called as Unit production.

- **Steps to Eliminate Unit Production :**

- Variables on right hand side of production should be replaced by the right hand side of all its production for every pair of non terminals ‘A’ and ‘B’.

Example 1:

Q. Consider the following grammar : G = { { A, B }, { a, b }, P, A }

where P consists of : A → B

B → a | b

Solution :

- In given example A → B is unit production.
- To remove this we should replace ‘B’ (right hand side of production) by right hand side of all its production i.e. ‘B’ can be replaced by a |b.
- Simplified or Reduced Grammar is :

A → a | b

Example 2:

Q. Reduce the following grammar :

S → A

A → B

B → C

C → a

Solution :

- In given example unit productions are :

  S → A

  A → B

  B → C

- To remove this we should replace ‘A’ by ‘B’ thus we have S → B.
- Further replace ‘B’ by ‘C’ and we have S → C.

Example 2: (Cont..)

- Now replace ‘C’ by ‘a’ and we get S → a.
- Similarly, for unit production A → B replace ‘B’ by ‘C’ and we have A → C.
- Further replace ‘C’ by ‘a’ and we get A → a.
- Next we have unit production B → C replace ‘C’ by ‘a’ thus we have B → a.
- After removing all unit productions we get :

  S → a

  A → a

  B → a

  C → a

- A, B and C are not in sentential form (not reachable from start state) hence remove their productions.
- Simplified or Reduced Grammar is :

  S → a

Removal of Null Production

- A production of the form, A → ϵ is called as null (or ϵ) production, where A is non terminal.
- **Steps to Eliminate Null Production :**

1. Delete all ϵ productions.
2. Identify nullable non-terminals.
3. If there is a production A → α where 'α', consist of at least one nullable non-terminal then add new productions with right hand side formed by deleting all possible subsets of nullable non-terminals.

After step 3 if we get production A → ϵ then do not add this to the final grammar.

Nullable Non Terminal

- In a given CFG if there is a non terminal ‘N’ and a production N → ϵ or ‘N’ derives ϵ in more than one step then ‘N’ is called as Nullable Non Terminal.
- **For Example :**

1. Consider the grammar : A → B | ϵ
   B → a

- In this grammar ‘A’ is nullable non-terminal as it has null production.

2. Consider the grammar : A → B | a
   B → a | ϵ

- As we have production B → ϵ then, ‘B’ is nullable non terminal.
- Also we have A → B so it gives production A → ϵ.
- Hence A is also nullable non terminal.

Example 1:

Q. Eliminate ϵ productions from G, where G consists of following productions:

S → aSa | bSb | ϵ

Solution :

Step 1 : Deleting ϵ-production

we get, S → aSa | bSb

Step 2 : Identify nullable non terminals

S is nullable non terminal as we have production S → ϵ .

Step 3 : Following productions consist of nullable non terminal, S on R.H.S.

S→ aSa | bSb

Example 1: (Cont..)

Step 4 : Add new productions by deleting all possible subsets of nullable non terminals.

S→ aa (Deleting S, since S → ϵ)

S→ bb (Deleting S, since S → ϵ)

Step 5 :

Final Simplified or Reduced Grammar is :

S → aSa | bSb | aa | bb

Example 2:

Q. Eliminate ϵ productions from given grammar G

S → ABA

A → aA | ϵ

B → bB | ϵ

Solution :

Step 1 : Deleting ϵ-production

S → ABA

A → aA

B → bB

Step 2 : Identify nullable non terminals

- A and B are nullable non terminals
- S is also nullable non terminal as we have production :

A → ϵ and B → ϵ , S→ ABA → ϵ (i.e. S → ϵ)

Example 2: (Cont..)

Step 3 : Following productions consist of nullable non terminals on R.H.S. :

S → ABA

A → aA

B → bB

Step 4 : Add new productions by deleting all possible subsets of nullable non terminals.

1. Consider production : S → ABA

S → BA (Deleting first A, since A → ϵ)

S → AB (Deleting last A, since A → ϵ)

S → AA (Deleting B, since B → ϵ)

S → A (Deleting AB, since A → ϵ and B→ ϵ)

S → B (Deleting both first and last A, since A → ϵ)

S → ϵ (Deleting ABA, since A → ϵ and B→ ϵ)

Note : We will not add S → ϵ to the final grammar.

Example 2: (Cont..)

Step 4 : Add new productions by deleting all possible subsets of nullable non terminals.

2. Consider production : A → aA

   A → a (Deleting A, since A → ϵ)

3. Consider production : B → bB

   B → b (Deleting B, since B → ϵ)

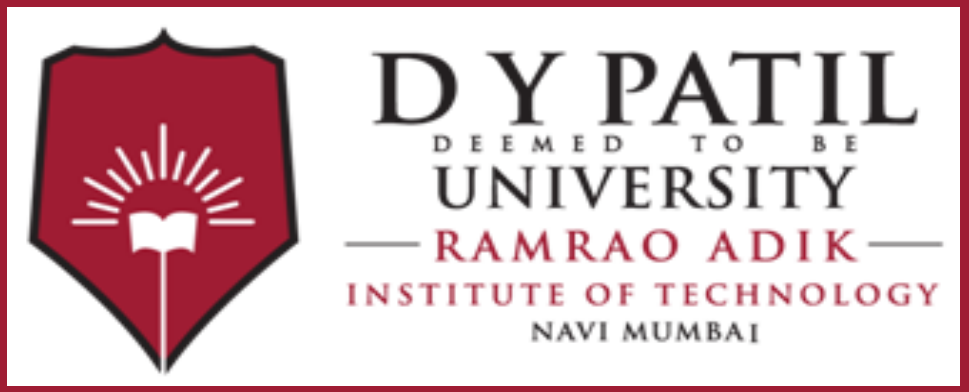
Step 5 :

Final Simplified or Reduced Grammar is :

S → ABA | BA | AB | AA | A | B

A → aA | a

B → bB | b


D Y PATIL
DEEMED TO BE
UNIVERSITY
RAMRAO ADIK
INSTITUTE OF TECHNOLOGY
NAVI MUMBAI

Thank You