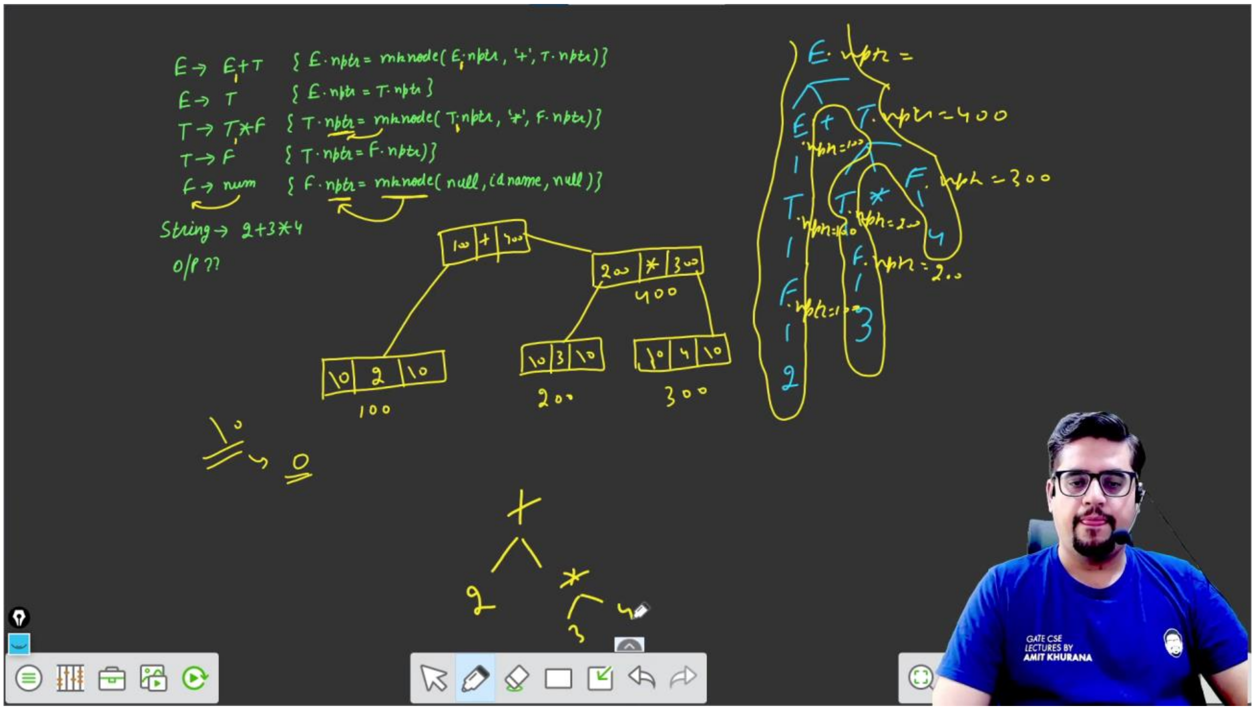
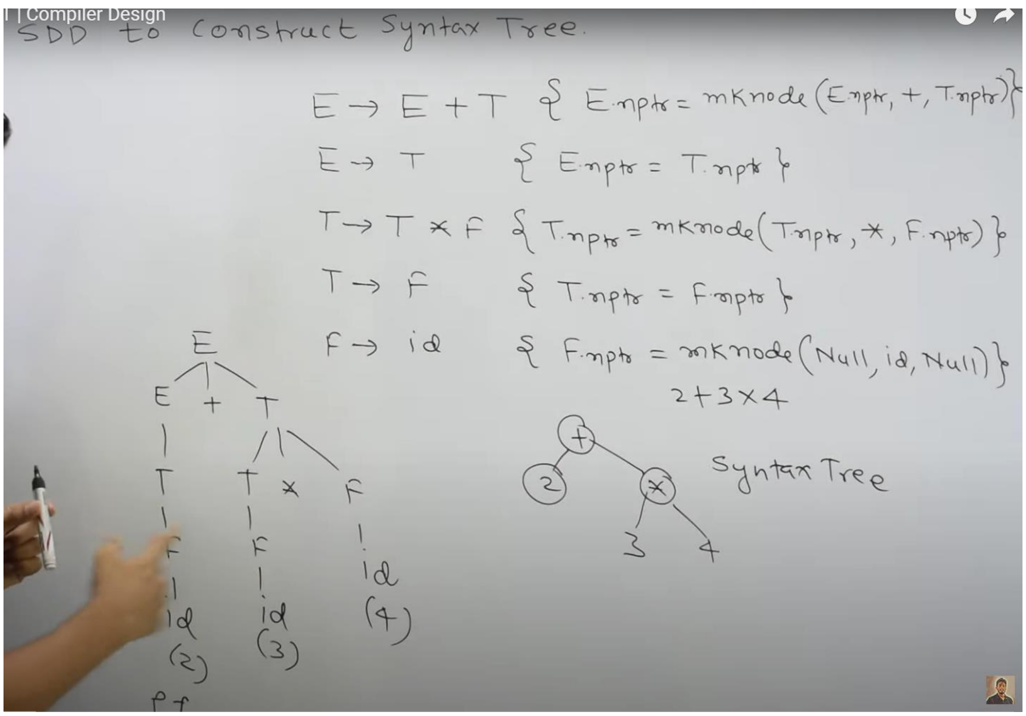


AST: Abstract syntax tree

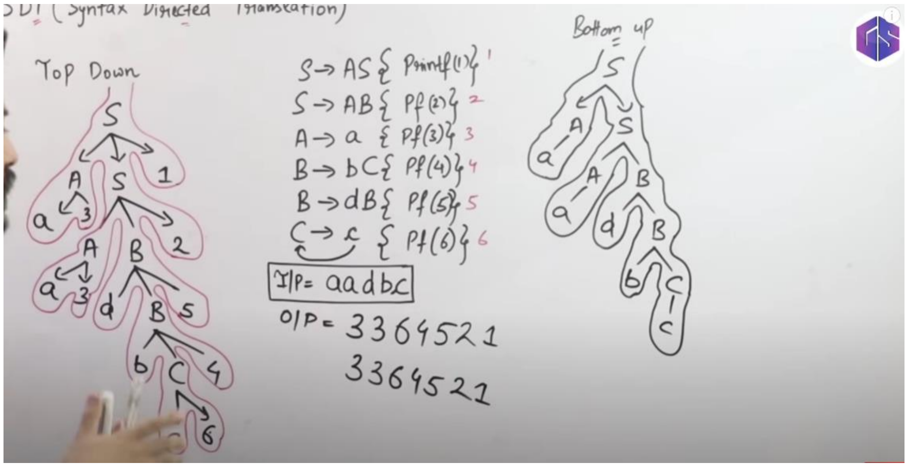
CPT: concrete Parse tree

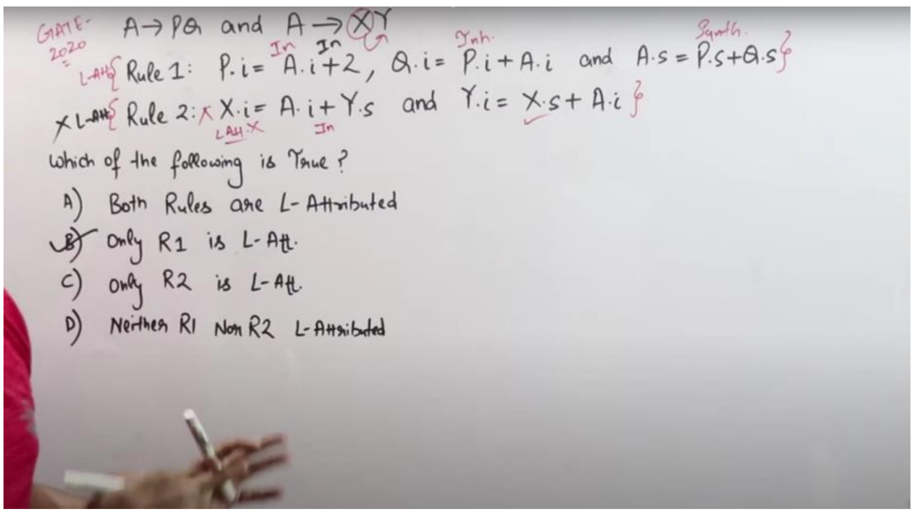


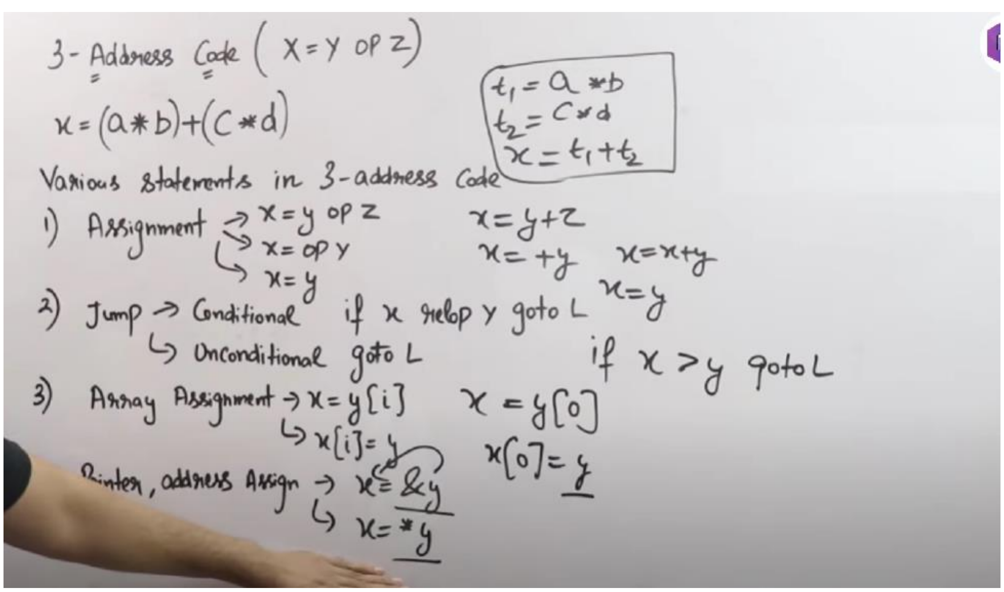
ST: syntax tree

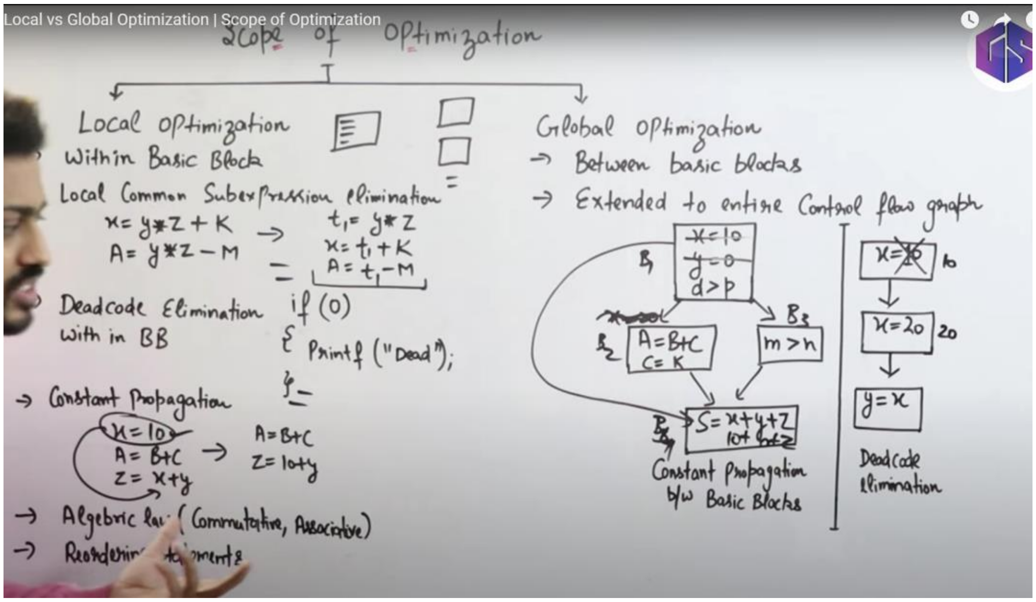
CST: concrete syntax tree

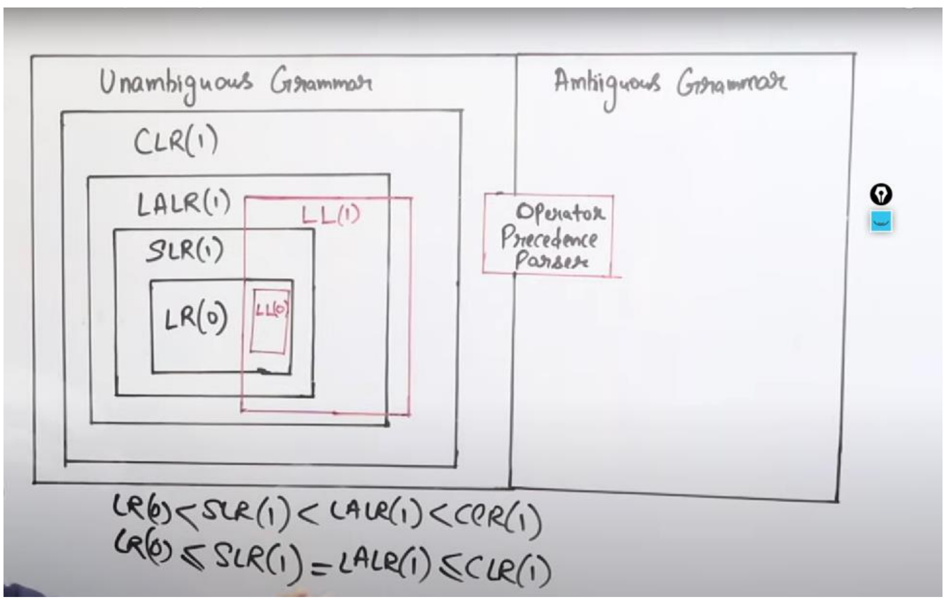
AST: abstract syntax tree -→ same as concrete syntax tree but some information is hidden as above

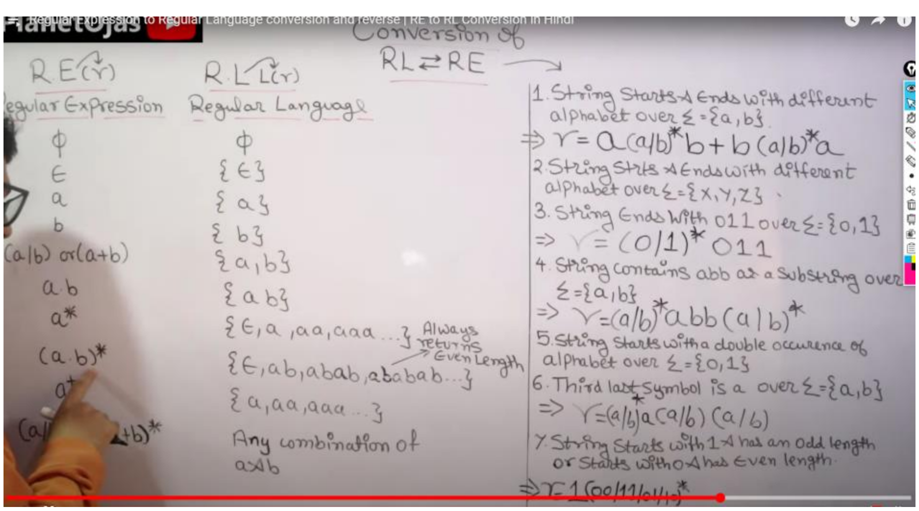

SDT (Syntax Directed Translation)
Top Down
Bottom up
S → AS { Printf(1) } 1
S → AB { Pf(2) } 2
A → a { Pf(3) } 3
B → bC { Pf(4) } 4
B → dB { Pf(5) } 5
C → c { Pf(6) } 6
I/P = aadbc
O/P = 3364521
3364521

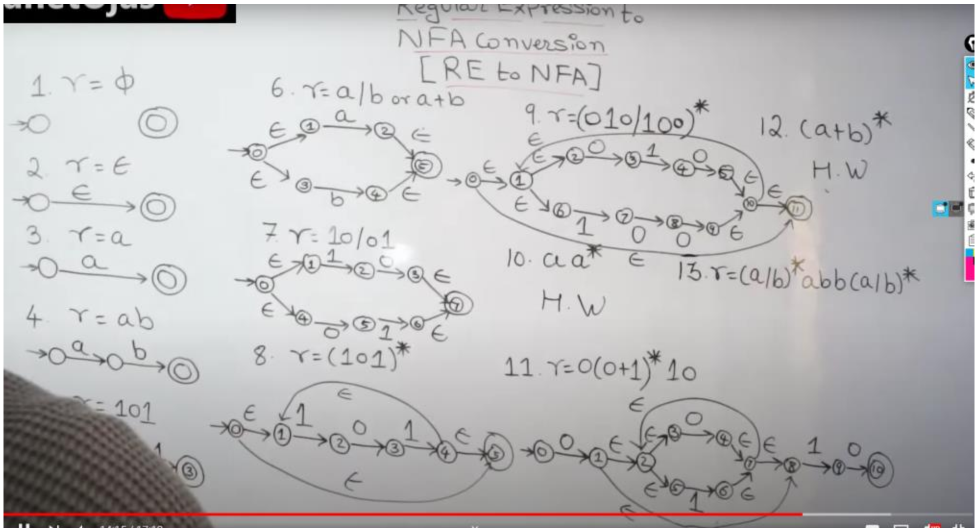

GATE-2020
A → PQ and A → XY
L-Att { Rule 1: P.i = A.i + 2, Q.i = P.i + A.i and A.s = P.s + Q.s }
× L-Att { Rule 2: X.i = A.i + Y.s and Y.i = X.s + A.i }
Which of the following is True?
A) Both Rules are L-Attributed
B) Only R1 is L-Att.
C) Only R2 is L-Att.
D) Neither R1 Nor R2 L-Attributed


3-Address Code (x = y op z)
x = (a*b)+(c*d)
t1 = a*b
t2 = c*d
x = t1 + t2
Various statements in 3-address Code
1) Assignment → x = y op z
x = op y
x = y
x = y+z
x = +y
x=x+y
x=y
2) Jump → Conditional if x relop y goto L
Unconditional goto L
if x > y goto L
3) Array Assignment → x = y[i]
x[i] = y
x = y[0]
x[0] = y
Pointer, address Assign → x = &y
x = *y

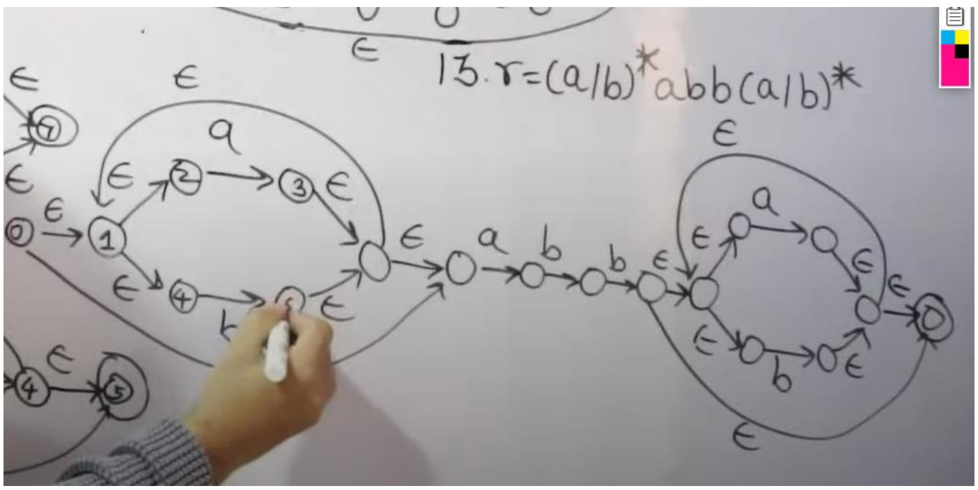

Local vs Global Optimization | Scope of Optimization
Scope of Optimization
Local optimization
Within Basic Block
Local Common Subexpression elimination
x = y*Z + K
A = y*Z − M
t1 = y*Z
x = t1 + K
A = t1 − M
Deadcode Elimination
with in BB
if (0)
{ Printf ("Dead");
}
→ Constant Propagation
x = 10
A = B+C
Z = x+y
A = B+C
Z = 10+y
→ Algebric law (Commutative, Associative)
→ Reordering statements
Global optimization
→ Between basic blocks
→ Extended to entire Control flow graph
x=10
y=0
d>p
A=B+C
C=K
m>n
S=x+y+Z
Constant Propagation
b/w Basic Blocks
x=20
y=x
Deadcode
Elimination


Unambiguous Grammar
Ambiguous Grammar
CLR(1)
LALR(1)
LL(1)
SLR(1)
LR(0)
LL(0)
Operator
Precedence
Parser
LR(0) < SLR(1) < LALR(1) < CLR(1)
LR(0) ≤ SLR(1) = LALR(1) ≤ CLR(1)

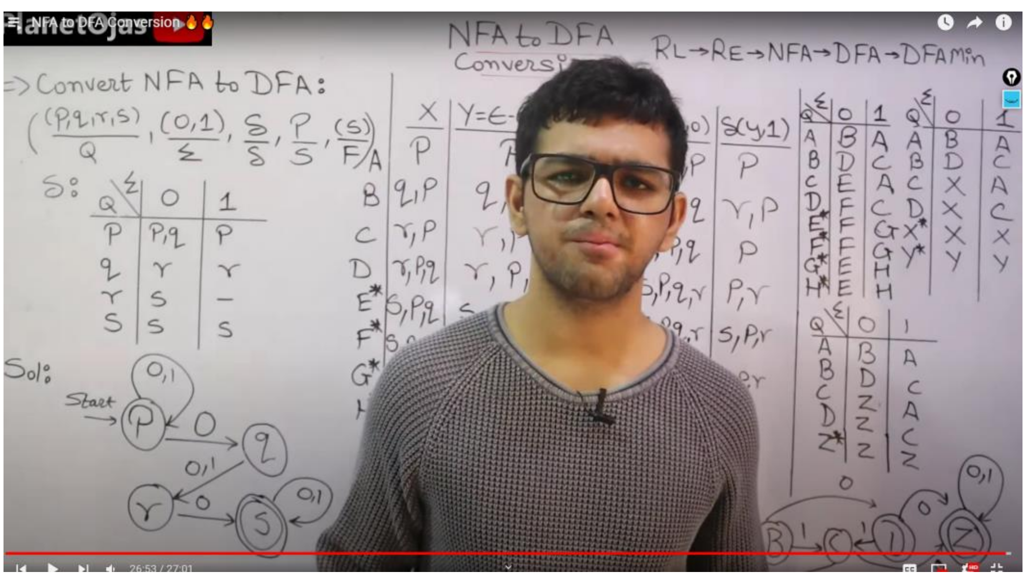

Regular Expression to Regular Language conversion and reverse | RE to RL Conversion in Hindi
Conversion of
RL ⇄ RE
RE(r)
Regular Expression
RL L(r)
Regular Language
ϕ
ϵ
a
b
(a|b) or (a+b)
a b
a*
(a b)*
ϕ
{ϵ}
{a}
{b}
{a, b}
{a b}
{ϵ, a, aa, aaa ...}
{ϵ, ab, abab, ababab ...}
Always returns Even Length
{a, aa, aaa ...}
Any combination of a & b
1. String Starts & Ends with different alphabet over Σ = {a, b}
⇒ r = a (a|b)* b + b (a|b)* a
2. String Starts & Ends with different alphabet over Σ = {x, y, z}
3. String Ends with 011 over Σ = {0, 1}
⇒ r = (0|1)* 011
4. String contains abb as a Substring over Σ = {a, b}
⇒ r = (a|b)* abb (a|b)*
5. String Starts with a double occurence of alphabet over Σ = {0, 1}
6. Third last symbol is a over Σ = {a, b}
⇒ r = (a|b)* a (a|b) (a|b)
7. String Starts with 1 & has an odd length or Starts with 0 & has even length.



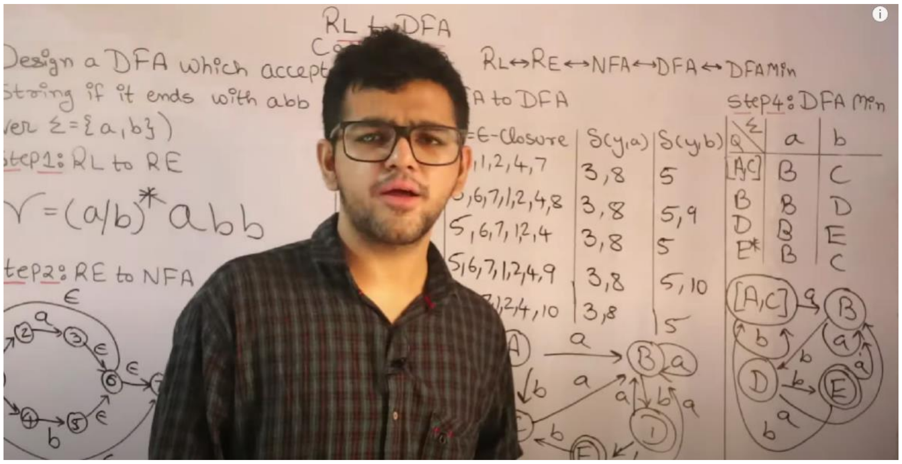
Regular Expression to
NFA Conversion
[RE to NFA]
1. r = ϕ
2. r = ∈
3. r = a
4. r = ab
6. r = a/b or a+b
7. r = 10/01
8. r = (101)*
9. r = (010/100)*
10. aa*
H.W
11. r = 0(0+1)*10
12. (a+b)*
H.W
13. r = (a/b)*abb(a/b)*

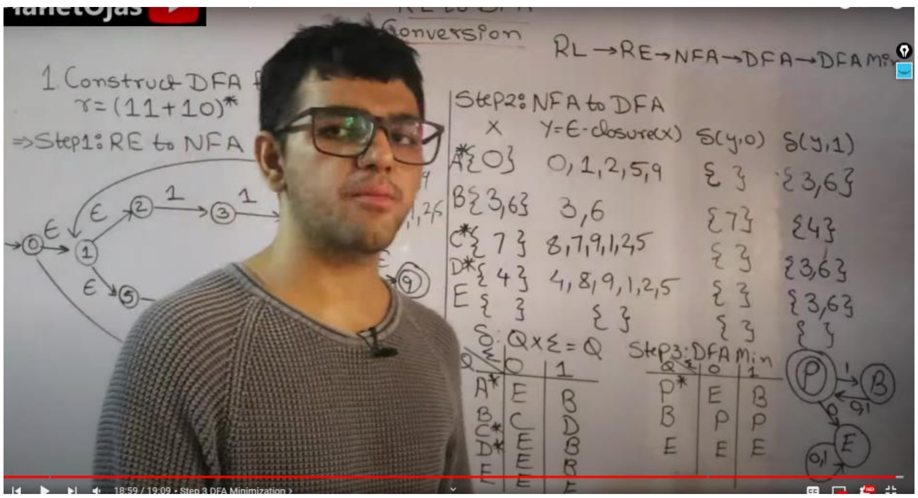


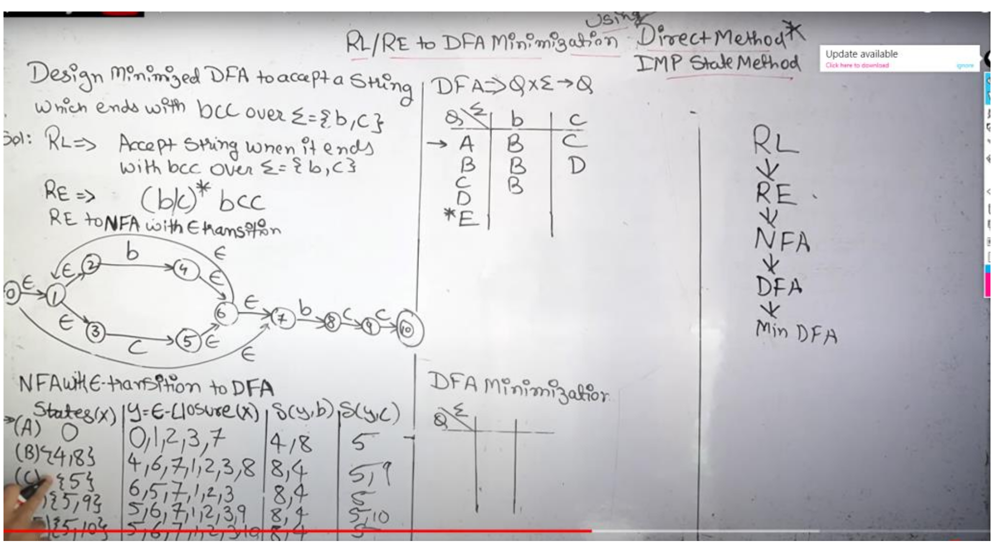
13. r = (a/b)*abb(a/b)*

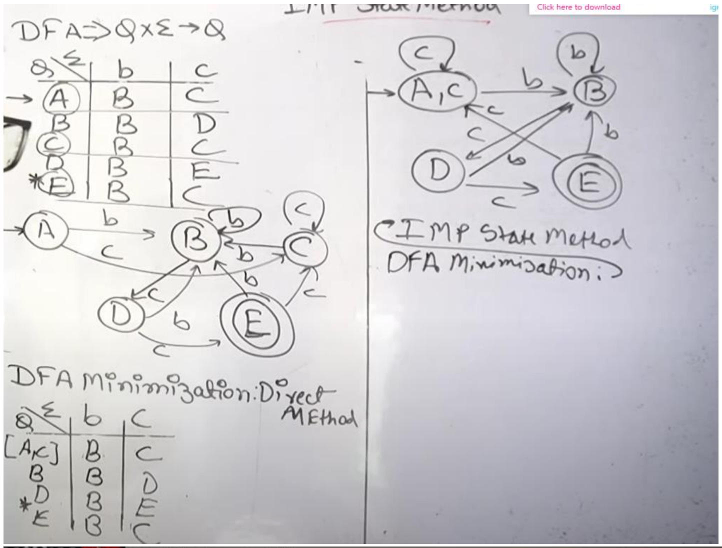


NFA to DFA Conversion
RL → RE → NFA → DFA → DFA Min
=> Convert NFA to DFA:
Sol:
Start

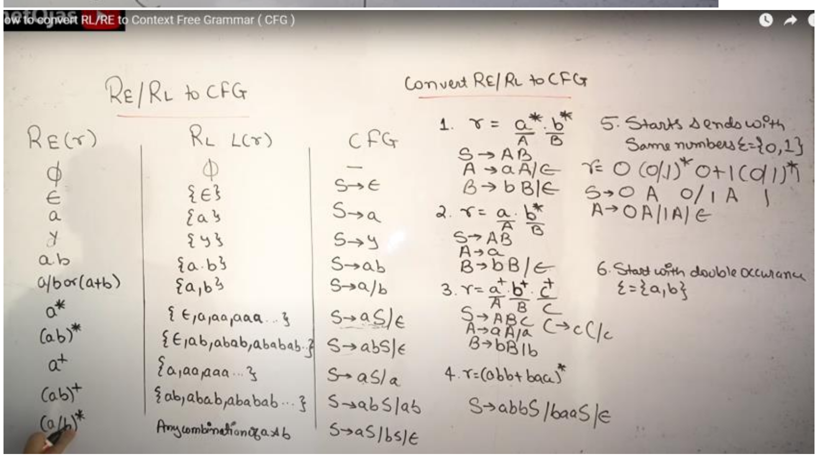

RL to DFA
Design a DFA which accept
String if it ends with abb
over Σ={a,b}
Step1: RL to RE
r=(a/b)* abb
Step2: RE to NFA
RL↔RE↔NFA↔DFA↔DFA Min
Step4: DFA Min


Conversion
RL→RE→NFA→DFA→DFA Min
1 Construct DFA
r=(11+10)*
=>Step1: RE to NFA
Step2: NFA to DFA
Y=E-closure(X)
δ: Q×Σ=Q
Step3: DFA Min

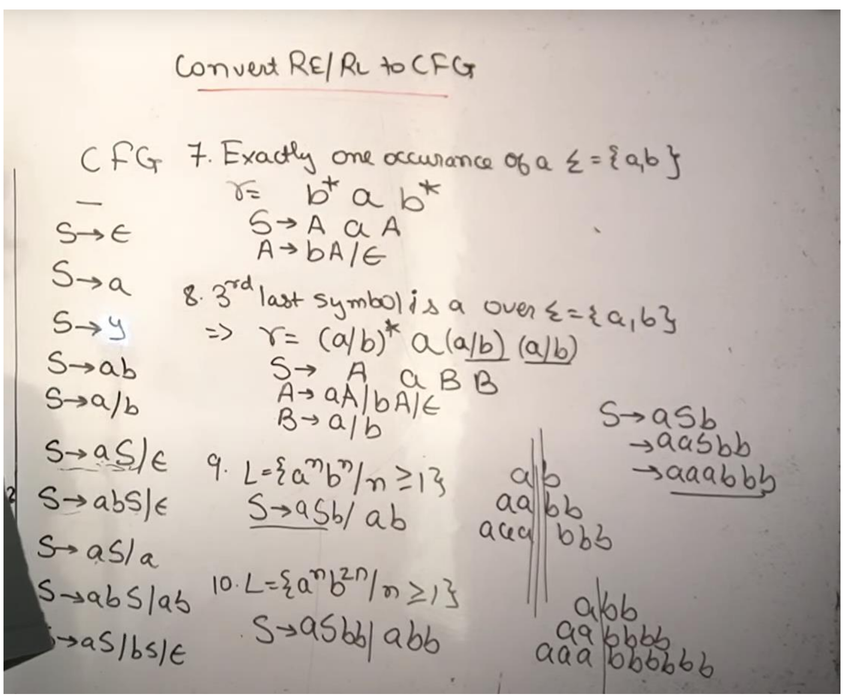

RL/RE to DFA Minimization using Direct Method
IMP State Method
Design minimized DFA to accept a String which ends with bcc over Σ={b,c}
Sol: RL=> Accept String when it ends with bcc over Σ={b,c}
RE => (b/c)* bcc
RE to NFA with ε transition
NFA with ε-transition to DFA
DFA=>Q×Σ→Q
DFA Minimization
RL
RE
NFA
DFA
Min DFA



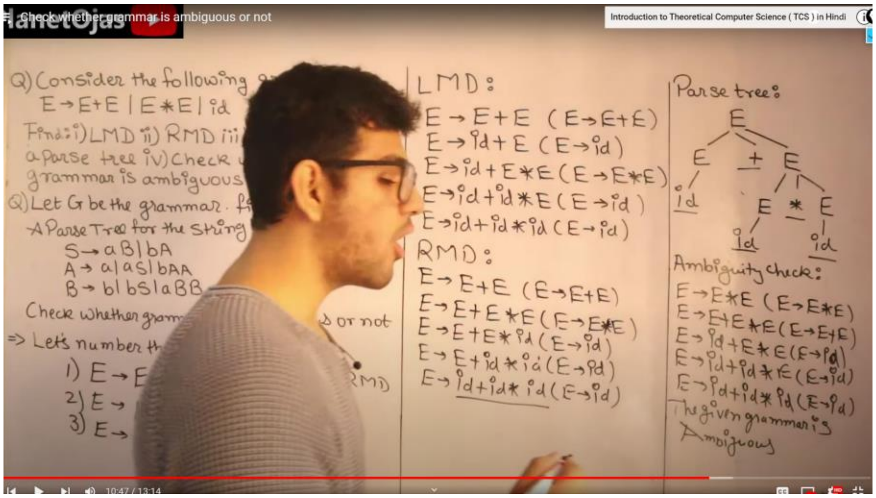
Click here to download
DFA ⇒ Q × Σ → Q
IMP State Method
DFA Minimization:
DFA Minimization: Direct Method



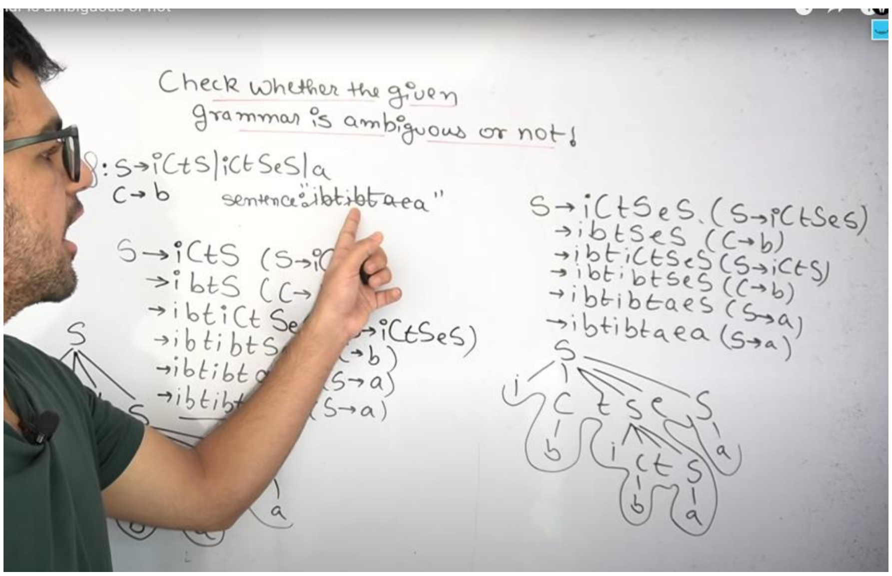
ow to convert RL/RE to Context Free Grammar (CFG)
RE/RL to CFG
RE(r)
RL L(r)
CFG
Convert RE/RL to CFG
1. r = a*. b*
S→AB
A→aA/∈
B→bB/∈
2. r = a. b*
S→AB
A→a
B→bB/∈
3. r = a+. b+. c+
S→ABC
A→aA/a
B→bB/b
C→cC/c
4. r = (abb+baa)*
S→abbS/baaS/∈
5. Starts & ends with Same numbers Σ={0,1}
r= 0 (0/1)* 0+1(0/1)*1
S→0 A 0/1 A 1
A→0A/1A/∈
6. Start with double occurance Σ={a,b}
Any combination of a & b



Convert RE/RL to CFG
CFG
S→∈
S→a
S→y
S→ab
S→a/b
S→aS/∈
S→abS/∈
S→aS/a
S→abS/ab
S→aS/bS/∈
7. Exactly one occurance of a Σ={a,b}
r= b* a b*
S→A a A
A→bA/∈
8. 3rd last symbol is a over Σ={a,b}
=> r= (a/b)* a (a/b) (a/b)
S→ A a B B
A→aA/bA/∈
B→a/b
9. L={aⁿbⁿ/n≥1}
S→aSb/ab
S→aSb
→aaSbb
→aaabbb
10. L={aⁿb²ⁿ/n≥1}
S→aSbb/abb



Check whether grammar is ambiguous or not
Introduction to Theoretical Computer Science (TCS) in Hindi
Q) Consider the following
E→E+E | E*E | id
Find: i) LMD ii) RMD iii)
a parse tree iv) Check
grammar is ambiguous
Q) Let G be the grammar. f
A Parse Tree for the string
S→aB|bA
A→a|aS|bAA
B→b|bS|aBB
Check whether gram s or not
=> Let's number th
1) E→E
2) E→
3) E→
LMD:
E→E+E (E→E+E)
E→id+E (E→id)
E→id+E*E (E→E*E)
E→id+id*E (E→id)
E→id+id*id (E→id)
RMD:
E→E+E (E→E+E)
E→E+E*E (E→E*E)
E→E+E*id (E→id)
E→E+id*id (E→id)
E→id+id*id (E→id)
Parse tree:
Ambiguity check:
E→E*E (E→E*E)
E→E+E*E (E→E+E)
E→id+E*E (E→id)
E→id+id*E (E→id)
E→id+id*id (E→id)
The given grammar is Ambiguous
10:47 / 13:14



Check whether the given grammar is ambiguous or not!
S→iCtS|iCtSeS|a
C→b
sentence: "ibtibtaea"
S→iCtS (S→i
→ibtS (C→
→ibtiCtSe →iCtSeS)
→ibtibtS →b)
→ibtibt (S→a)
→ibtib (S→a)
S→iCtSeS (S→iCtSeS)
→ibtSeS (C→b)
→ibtiCtSeS (S→iCtS)
→ibtibtSeS (C→b)
→ibtibtaeS (S→a)
→ibtibtaea (S→a)