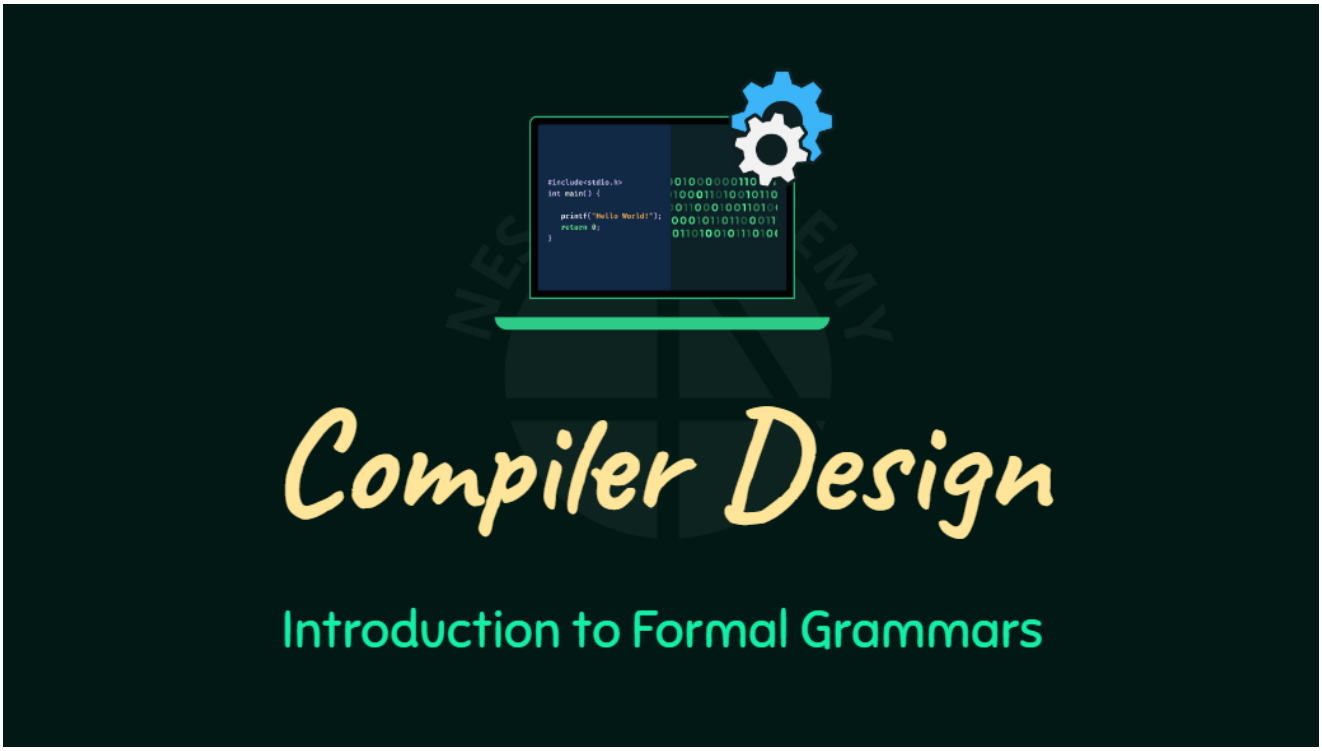
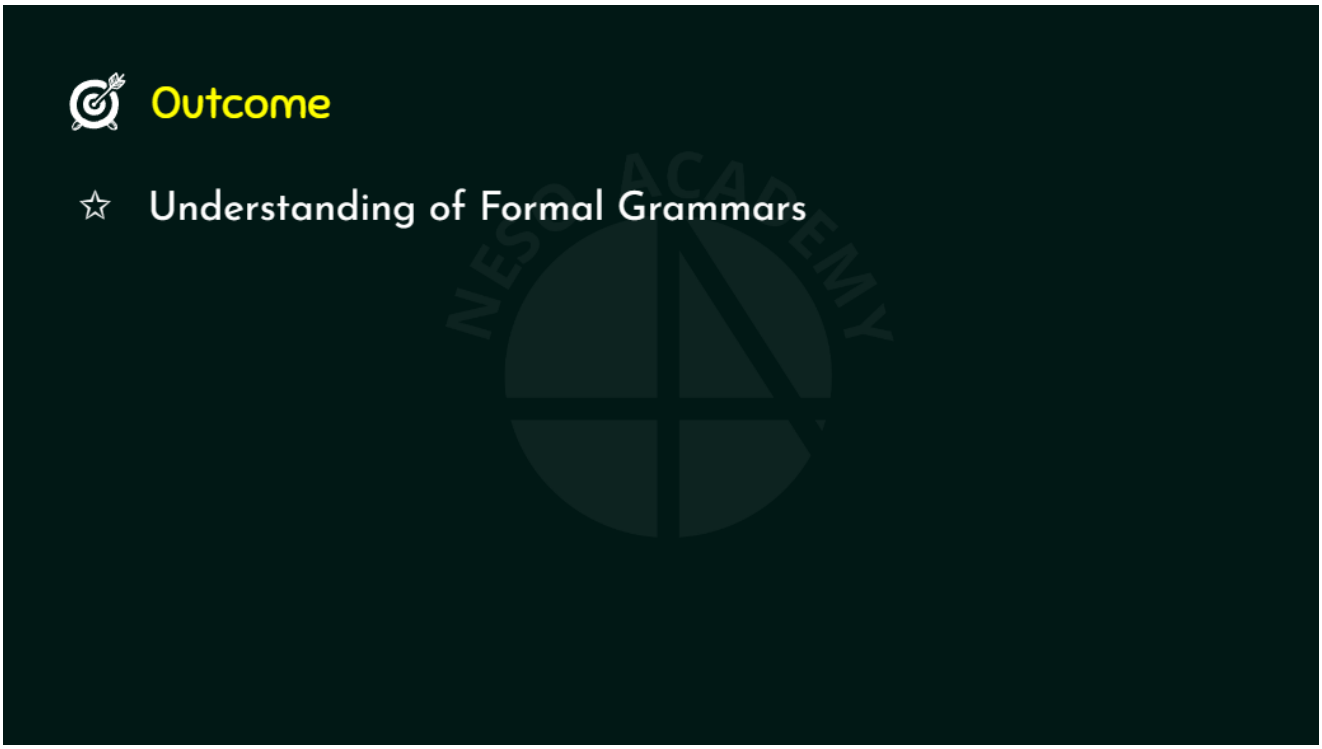
Syntax Analysis | Neso Academy

nesoacademy.org/cs/12-compiler-design/ppts/02-syntaxanalysis

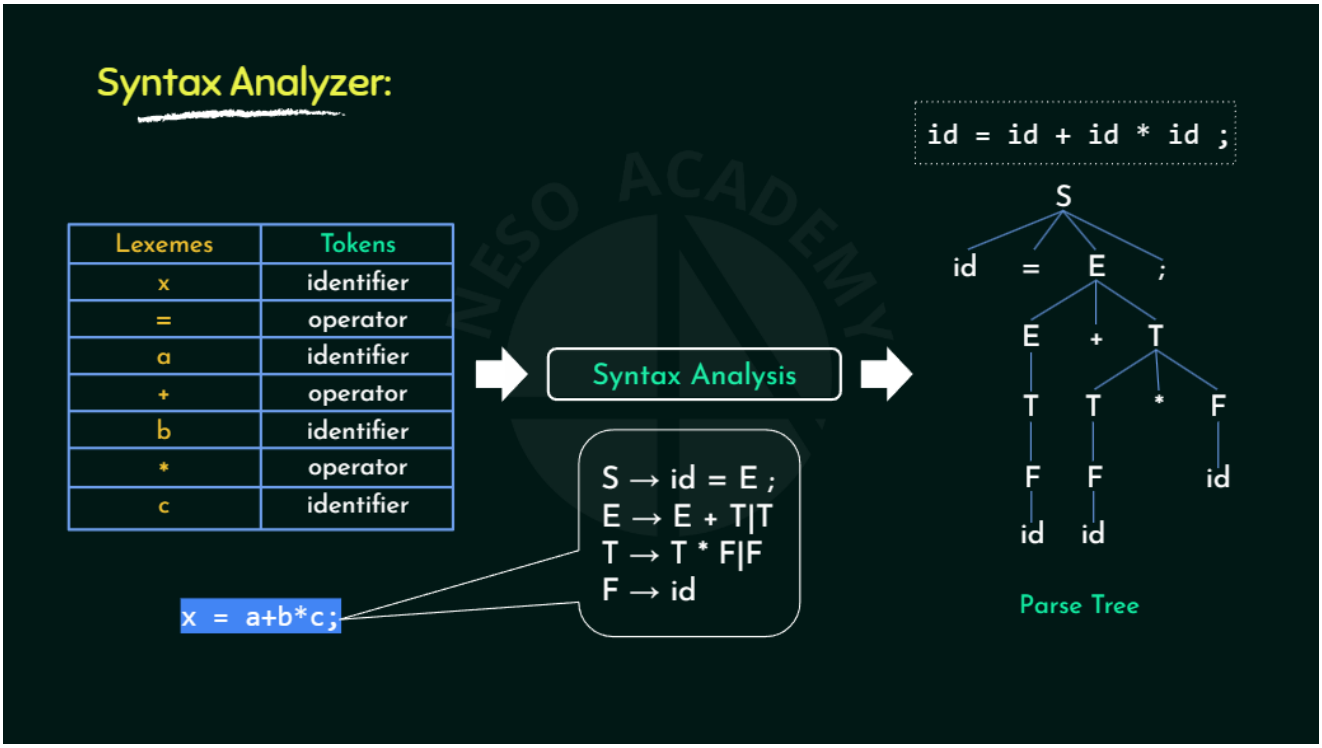
Syntax AnalysisNeso AcademyCHAPTER-2



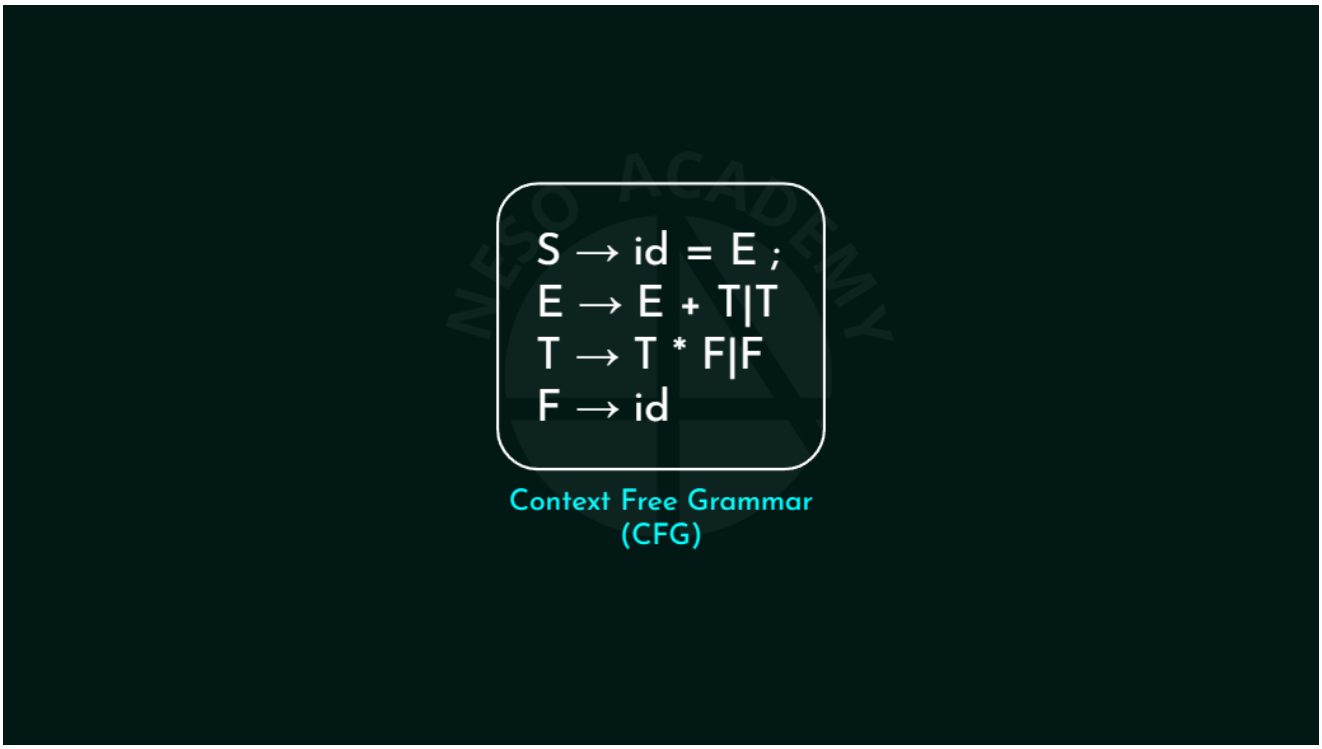
Compiler DesignIntroduction to Formal Grammars



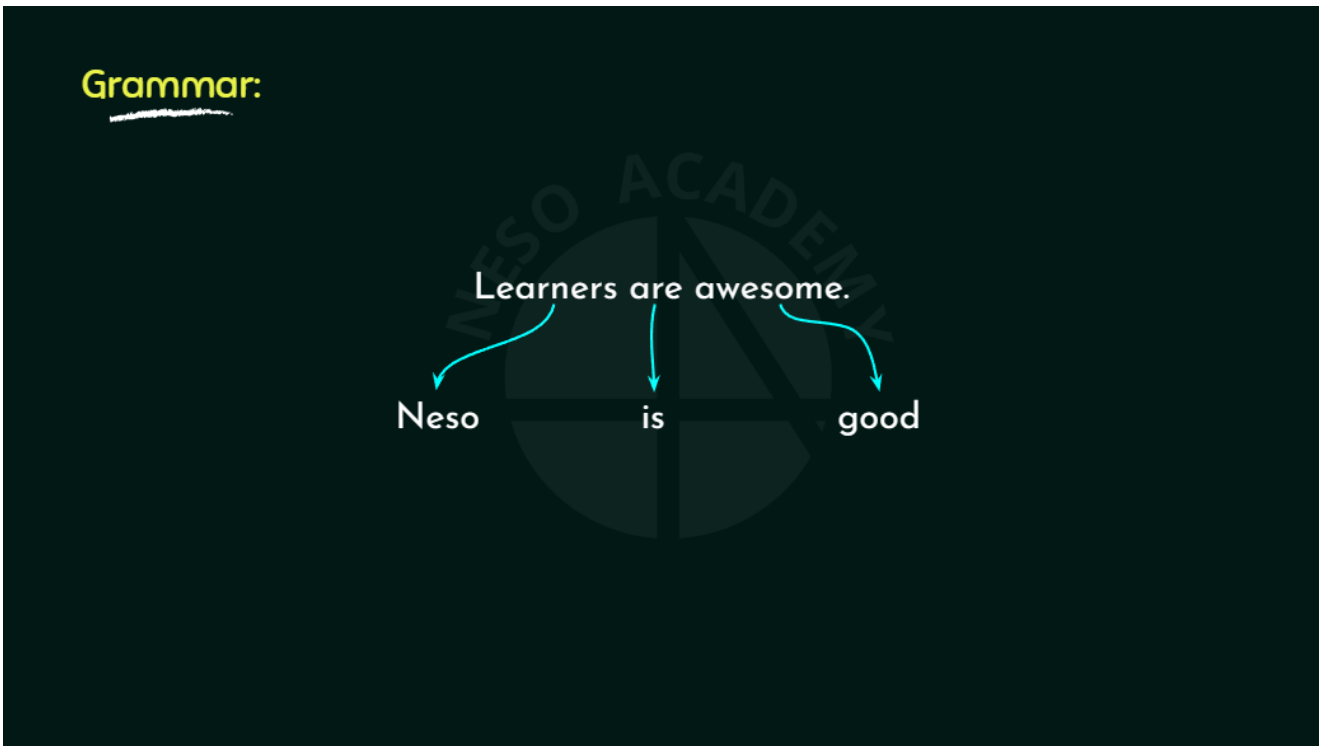
Outcome☆Understanding of Formal Grammars



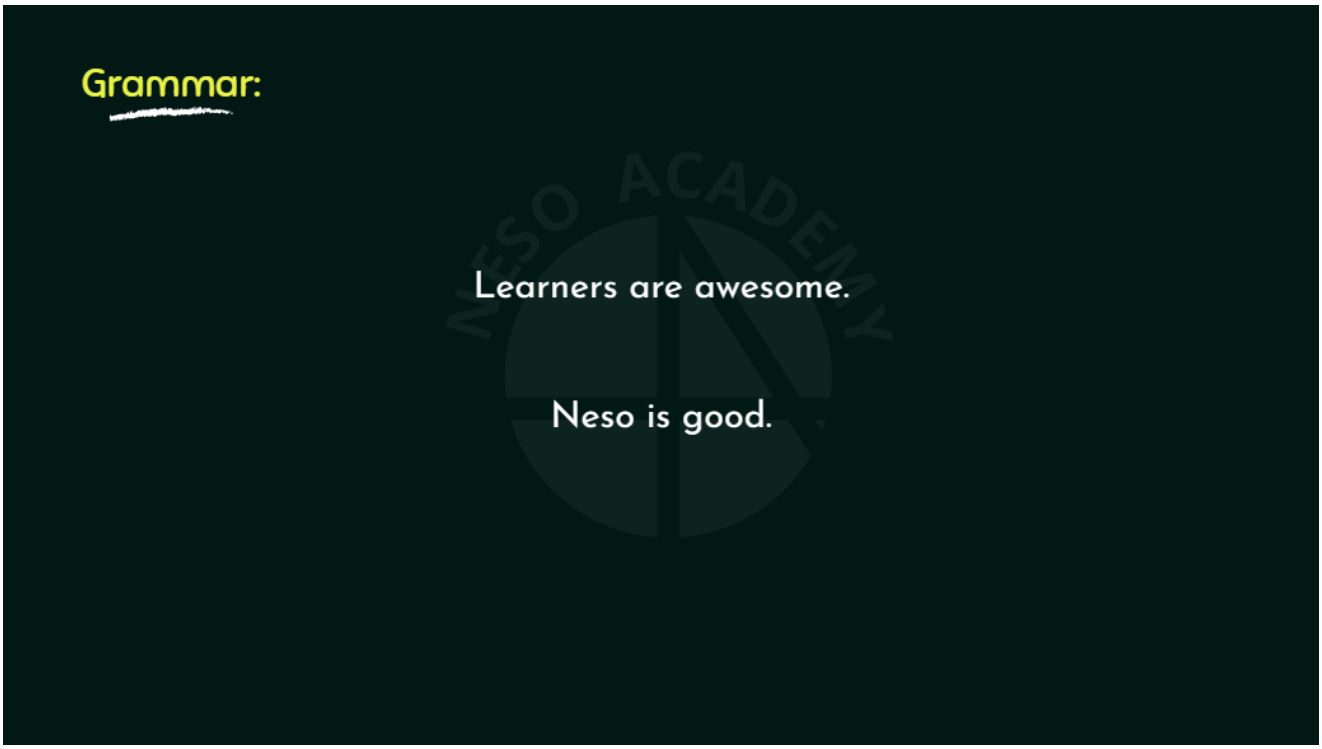
Syntax AnalysisSyntax Analyzer:LexemesTokensxidentifier=operatoraidentifier+operatorbidentifier*operatorcidentifier x = a+b*c;S → id = E ;E → E + T|TT → T * F|FF → idSid=E;E+TT*FTFFidididParse Tree id = id + id * id ;



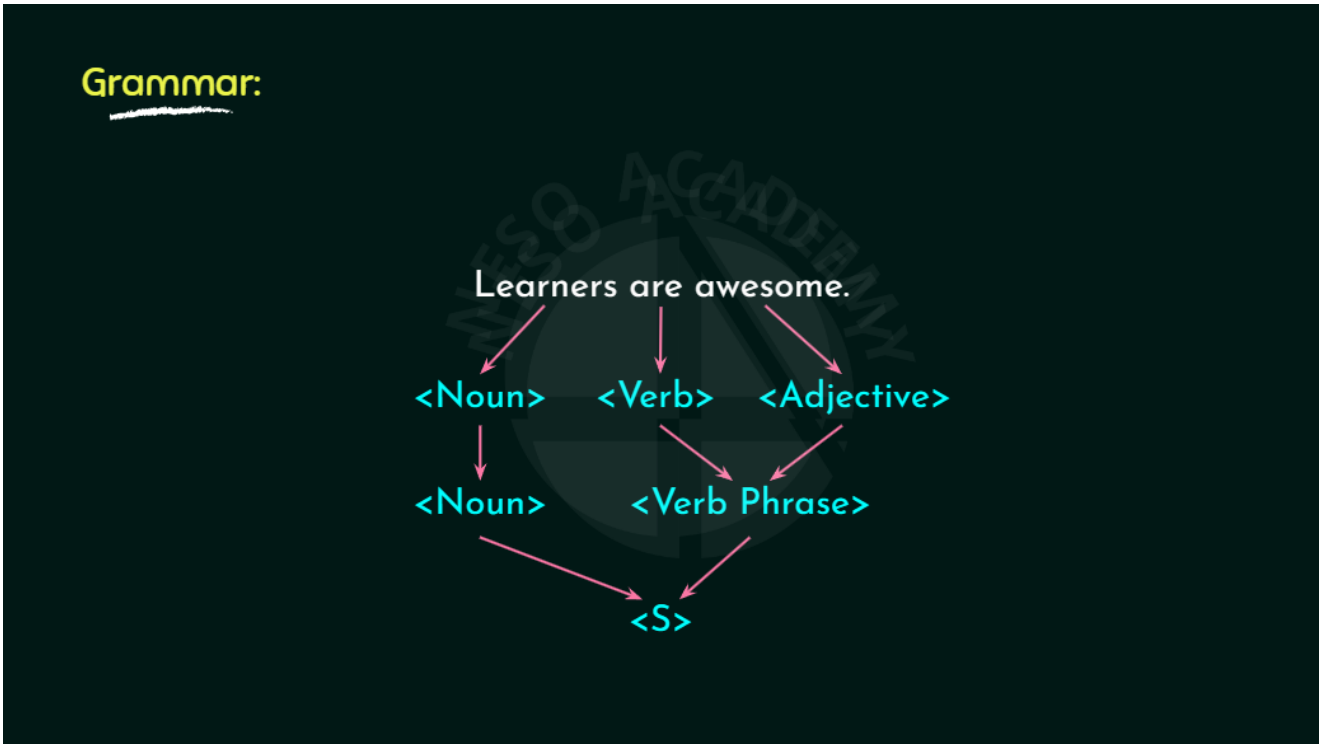
S → id = E ; E → E + T|T T → T * F|F F → idContext Free Grammar(CFG)



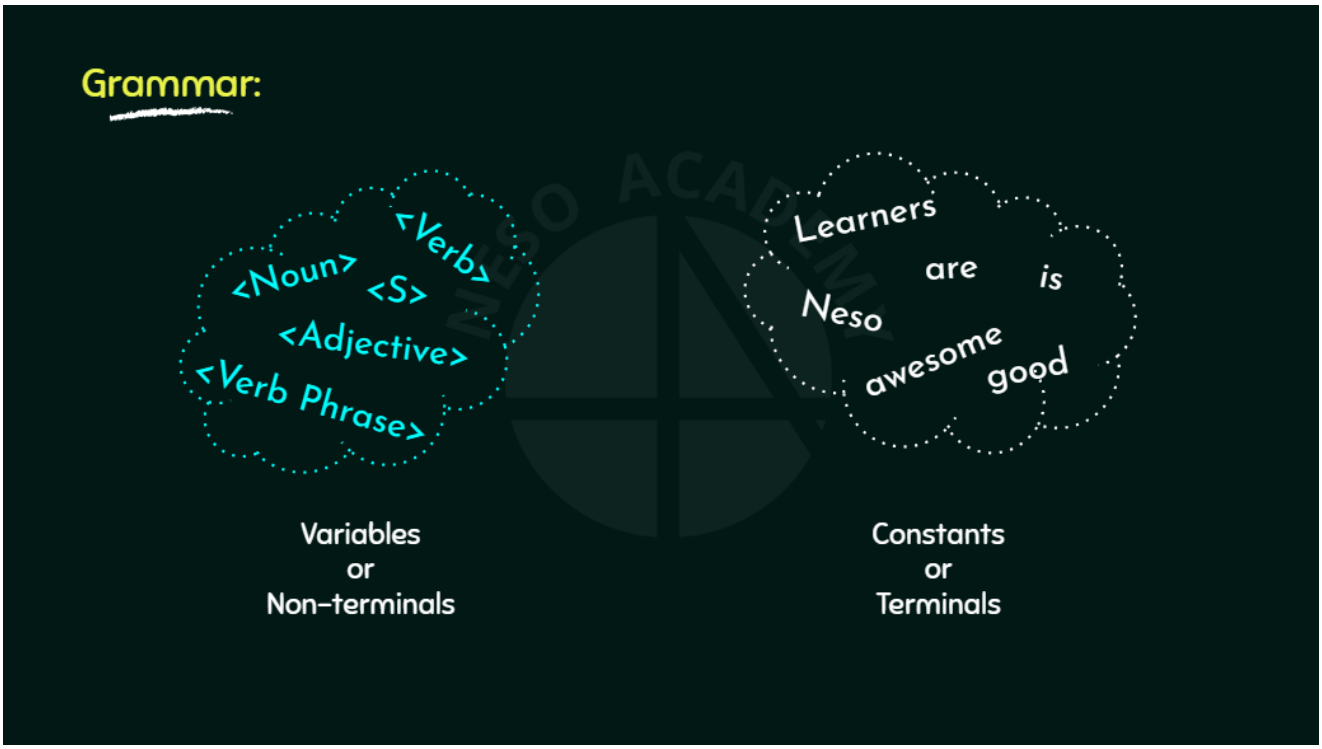
Grammar:Learners are awesome.Nesoisgood



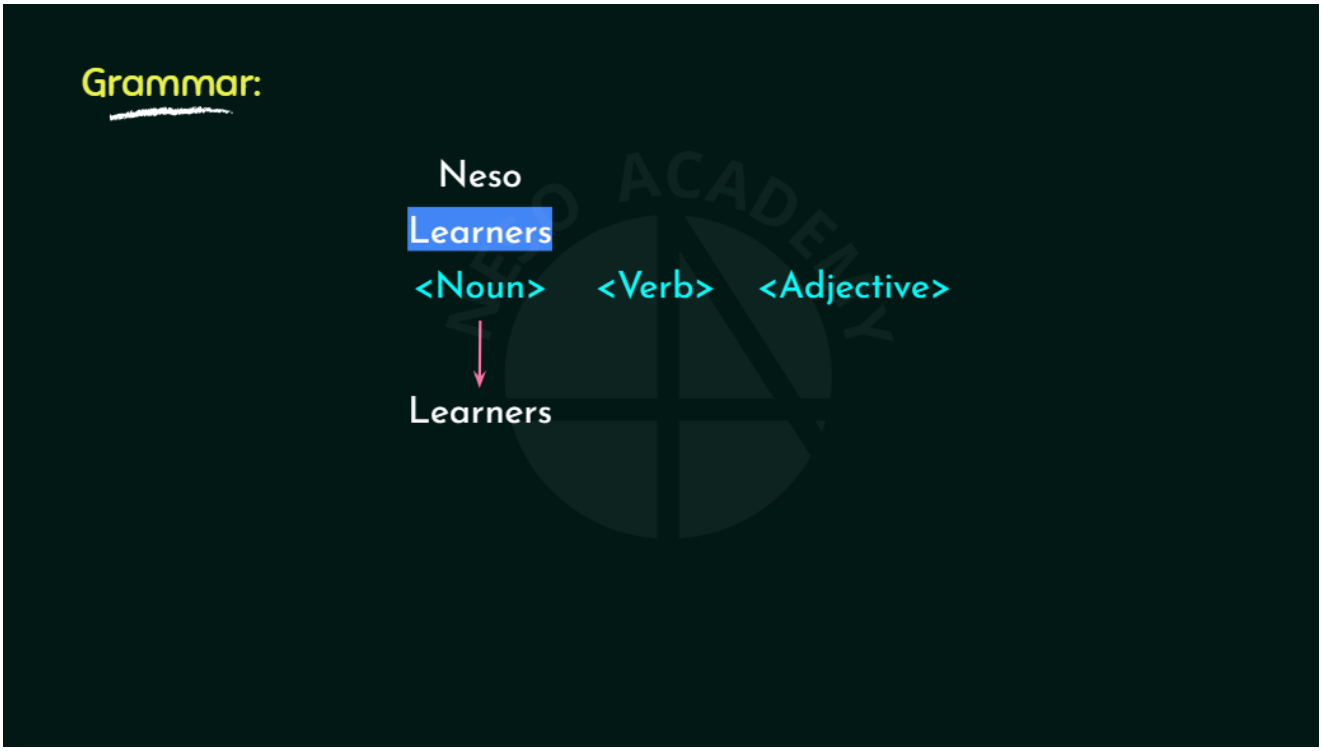
Learners are awesome.Grammar:Neso is good.



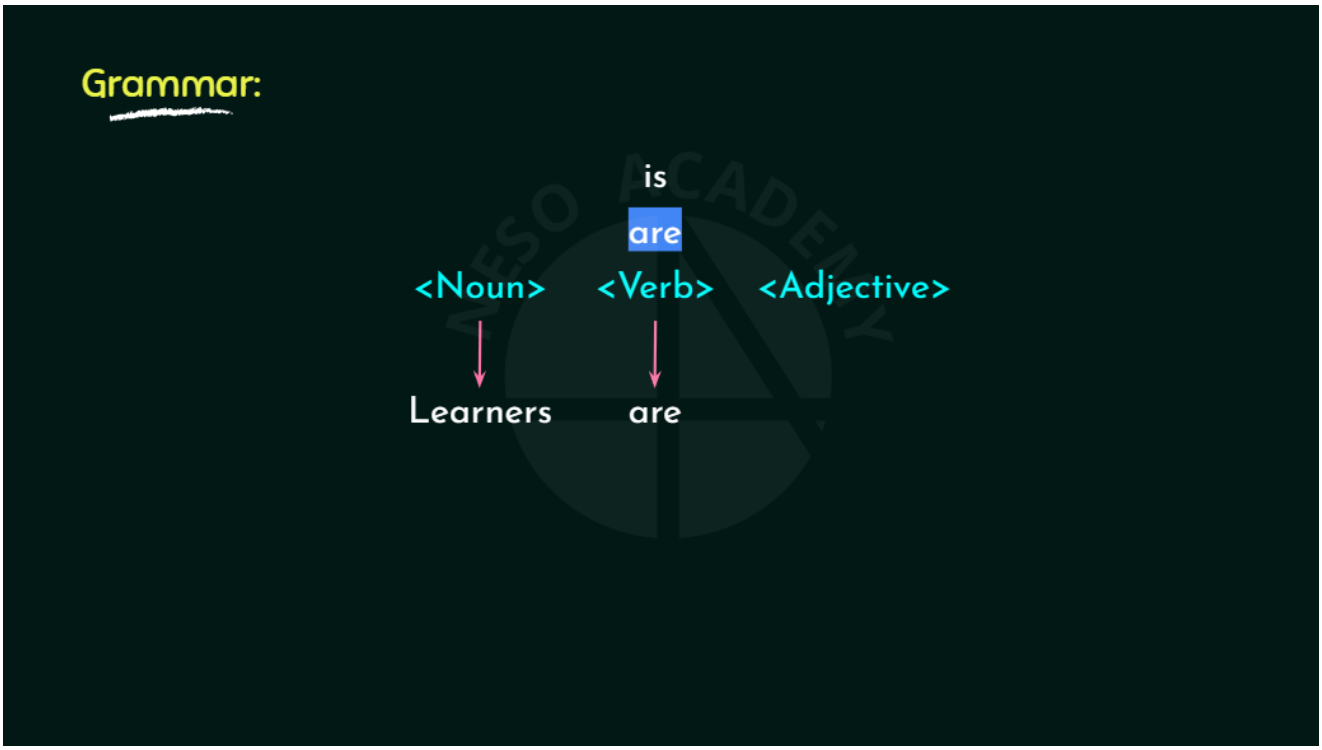
Learners are awesome.Grammar:<Noun><Verb><Adjective><Verb Phrase><Noun><S>



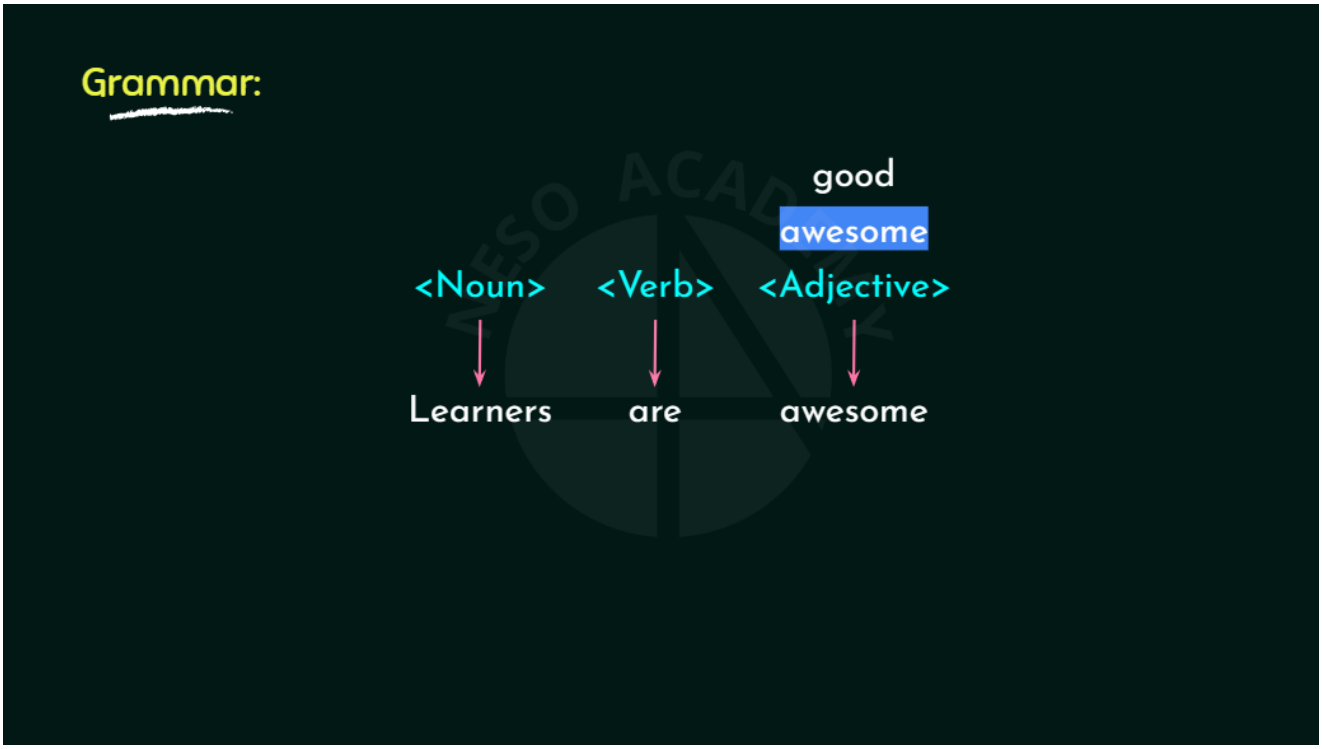
Grammar:NesoisgoodLearnersareawesomeVariablesor Non-terminalsConstantsor Terminals<Noun><Verb><Adjective><Verb Phrase><S>



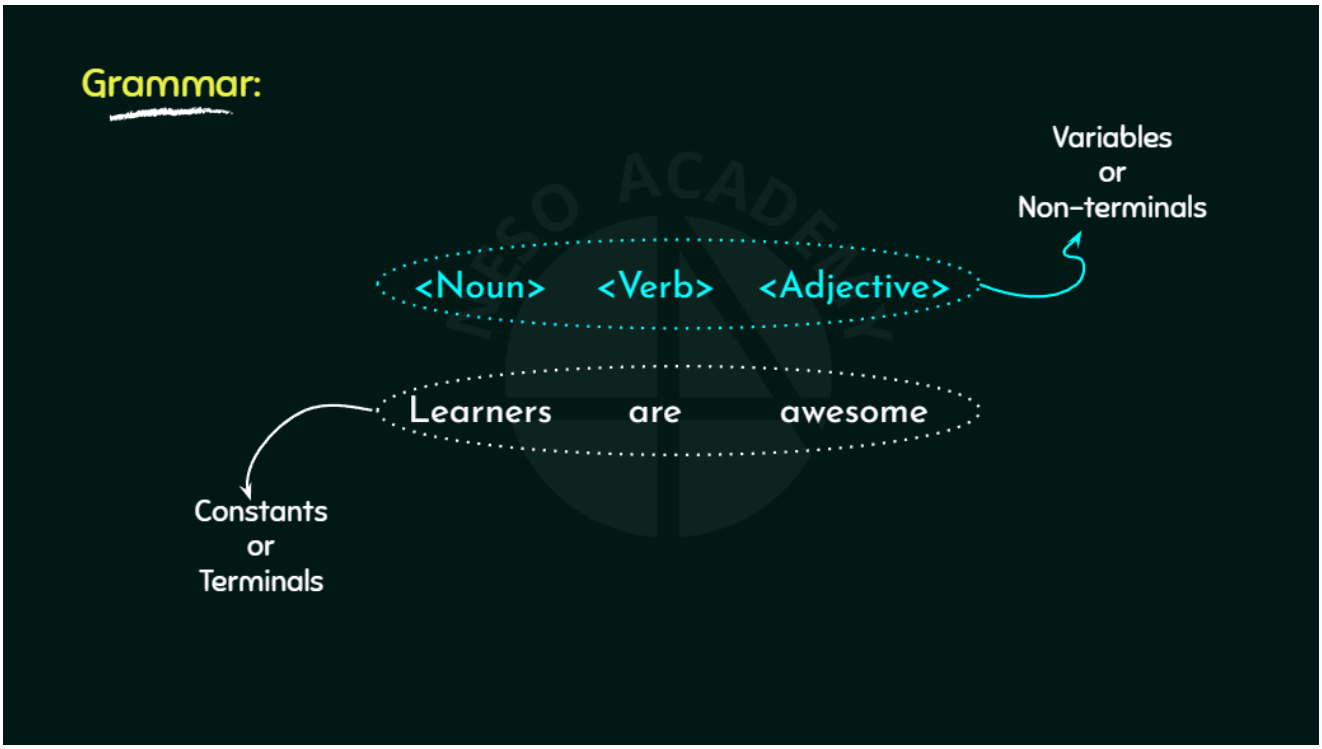
<Noun><Verb><Adjective>Grammar:NesoLearnersLearners



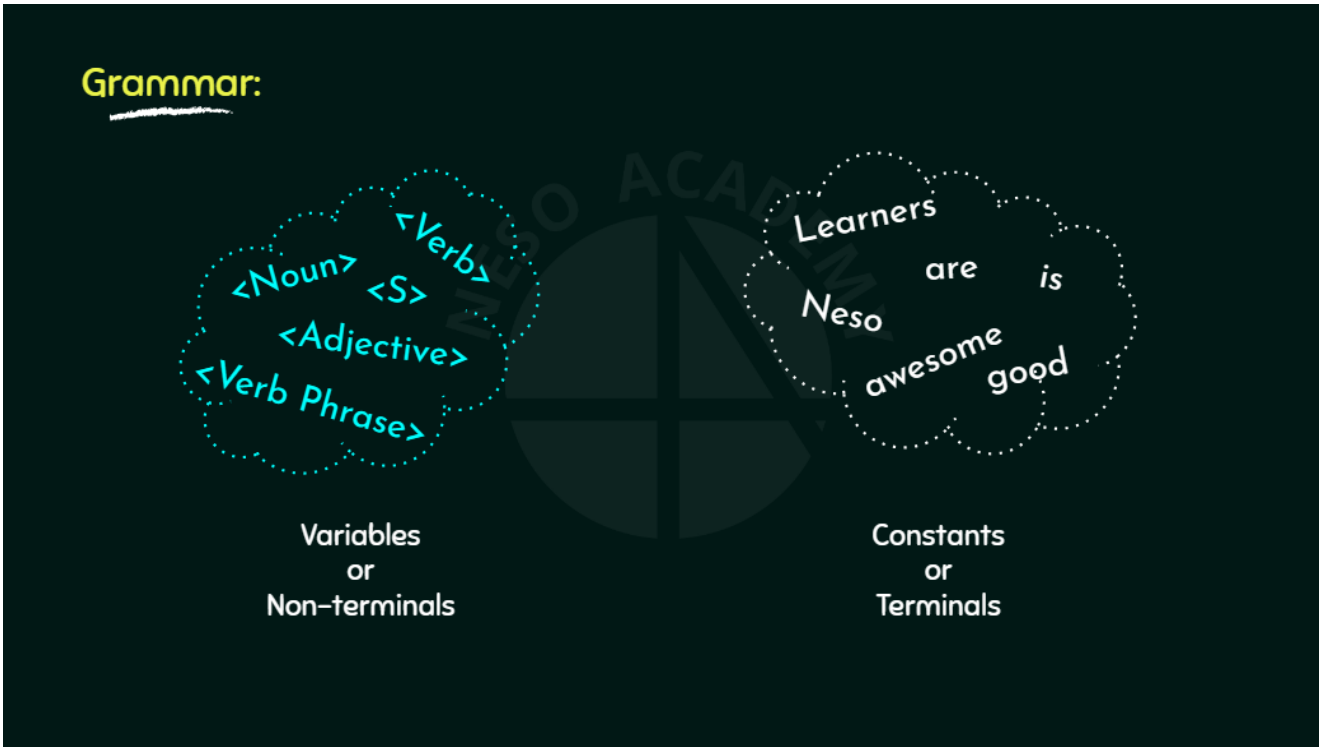
<Noun><Verb><Adjective>Grammar:Learnersisareare



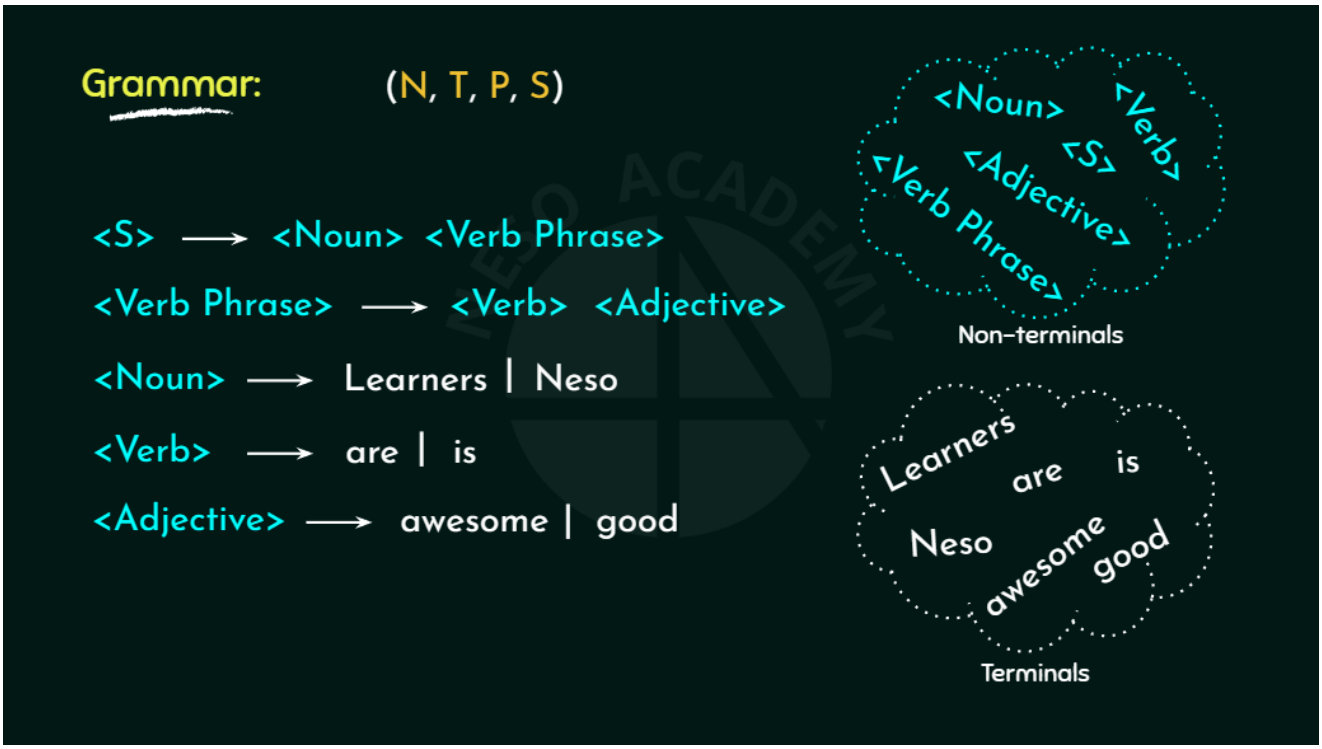
<Noun><Verb><Adjective>Grammar:Learnersaregoodawesomeawesome



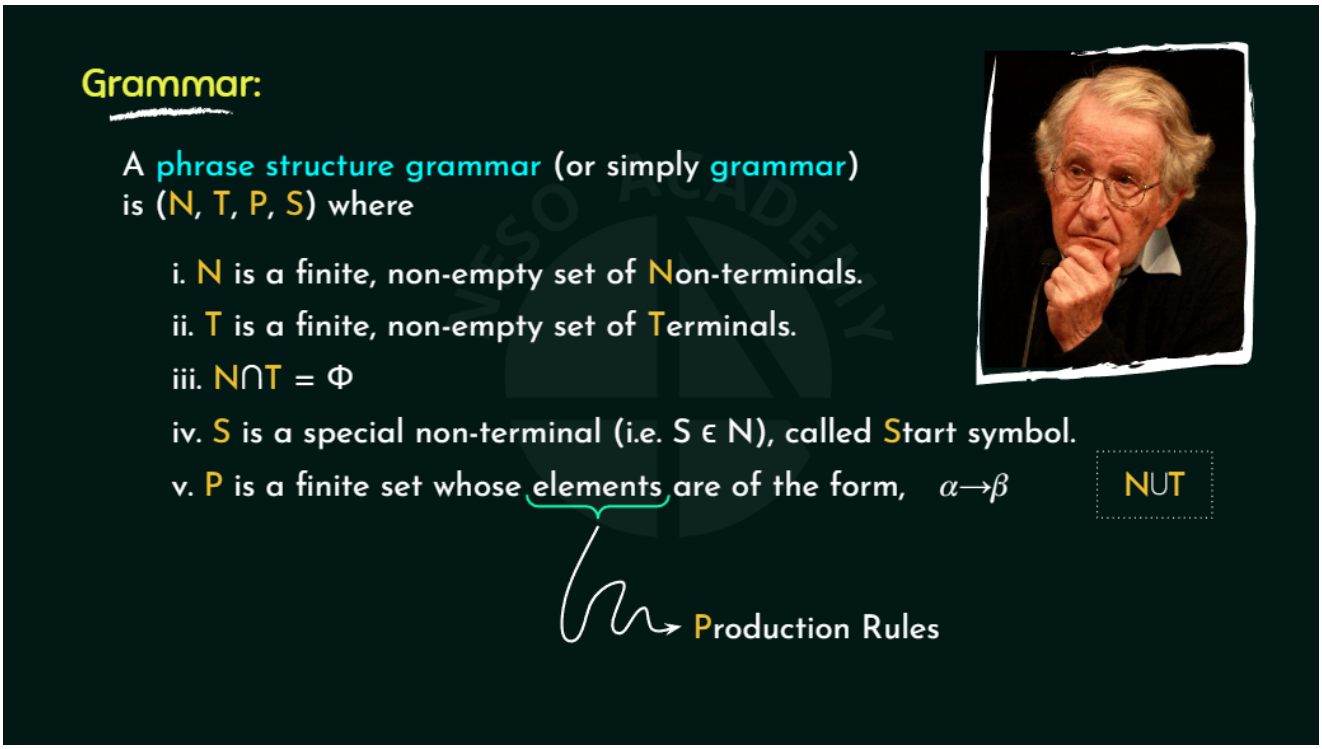
<Noun><Verb><Adjective>Grammar:LearnersareawesomeVariablesor Non-terminalsConstantsor Terminals



Grammar:NesoisgoodLearnersareawesomeVariablesor Non-terminalsConstantsor Terminals<Noun><Verb><Adjective><Verb Phrase><S>

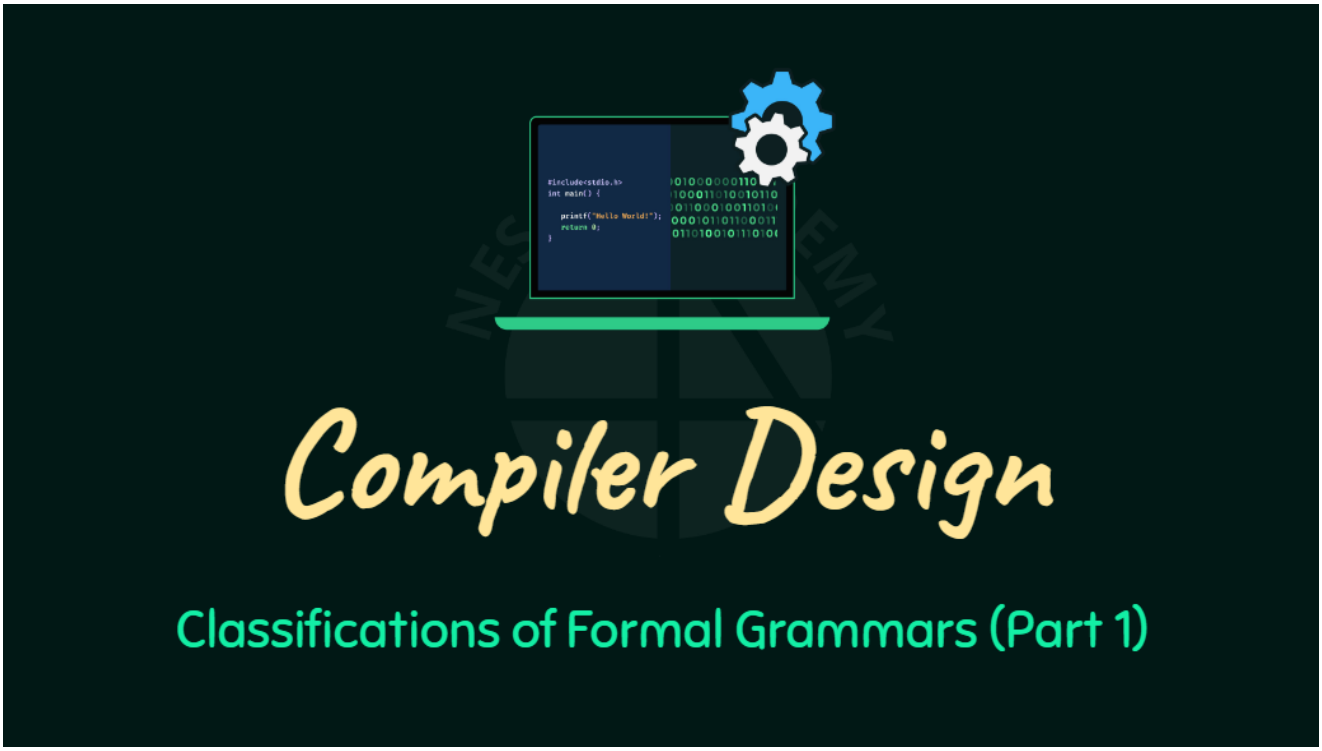


Grammar:NesoisgoodLearnersareawesomeTerminalsNon-terminals<Noun><Verb><Adjective><Verb Phrase><S><S><Noun><Verb Phrase><Verb><Adjective><Verb Phrase><Noun>Learners|<Verb>are|is<Adjective>awesome|good(N, T, P, S)Neso

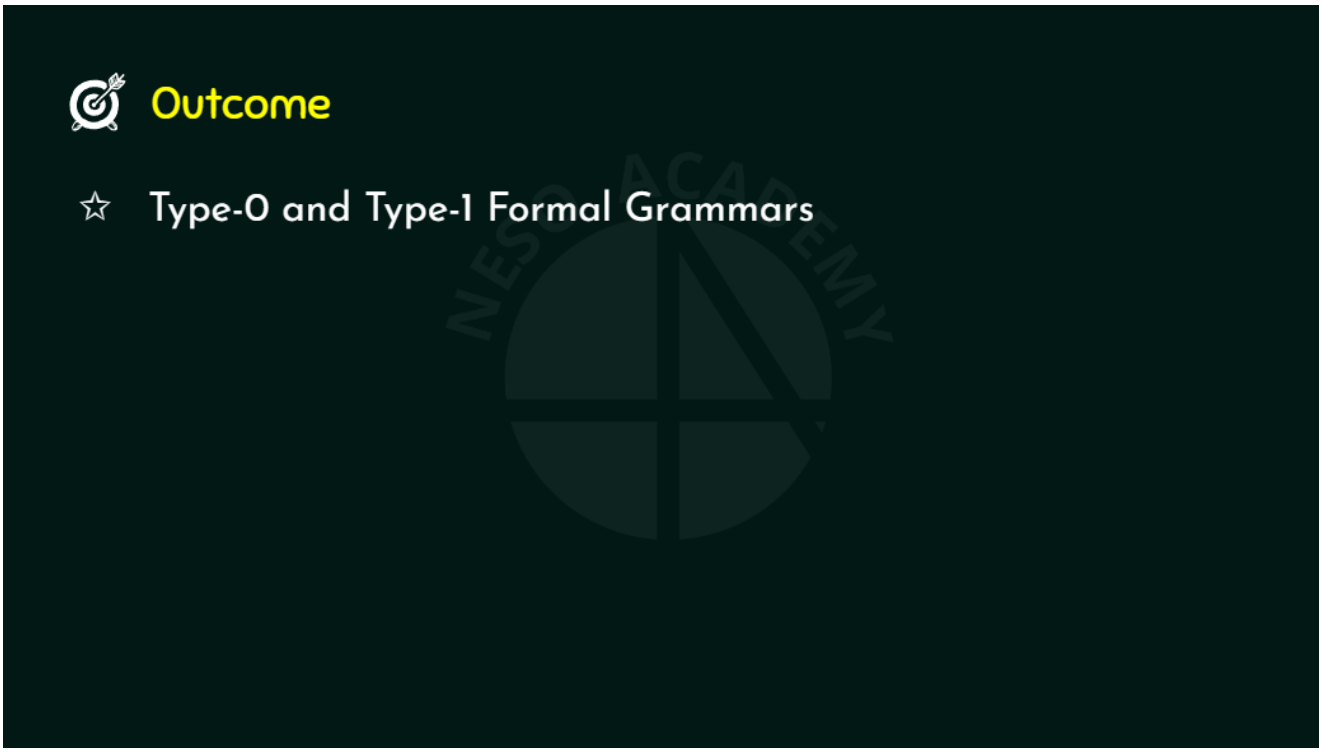


Grammar:A phrase structure grammar (or simply grammar)is (N, T, P, S) i. N is a finite, non-empty set of Non-terminals.ii. T is a finite, non-empty set of Terminals.iii. N∩T = Φiv. S is a special non-terminal (i.e. S ε N), called Start symbol.v. P is a finite set whose elements are of

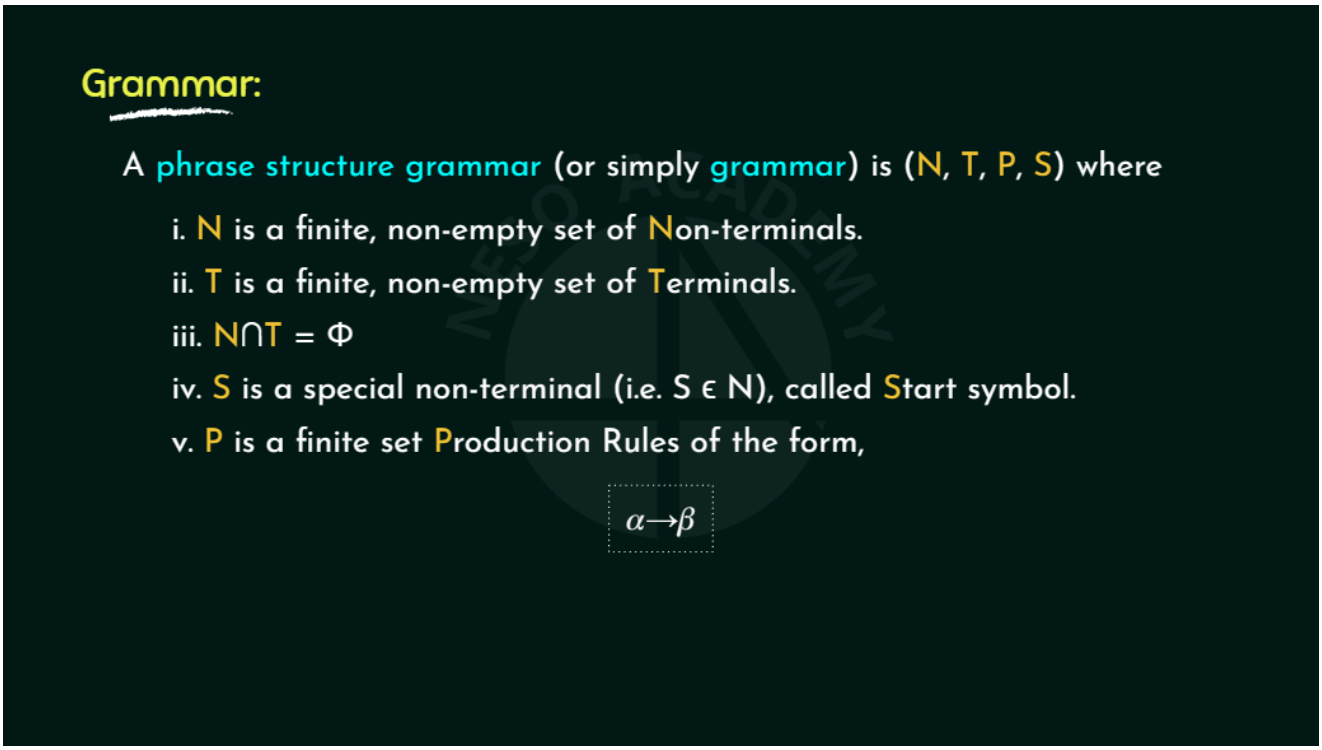
the form,$\alpha \rightarrow \beta$N∪TProduction Ruleswhere



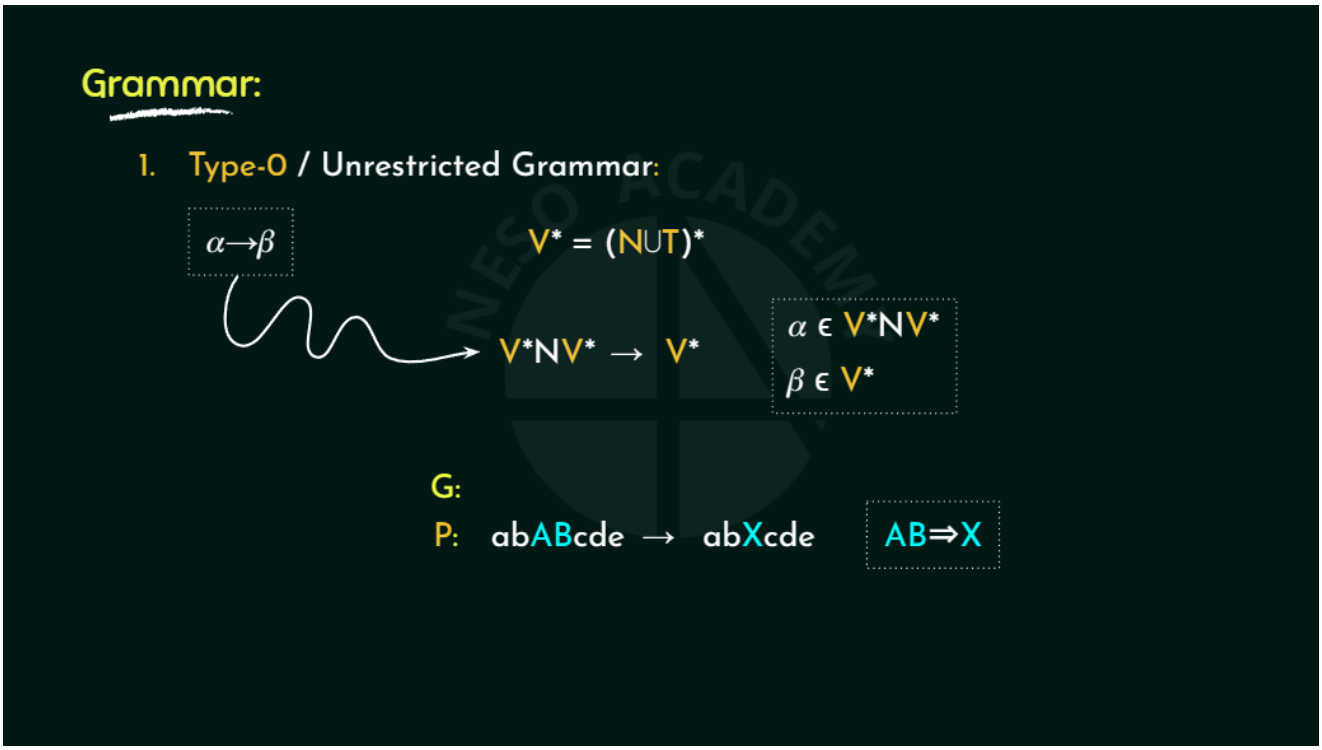
Compiler DesignClassifications of Formal Grammars (Part 1)



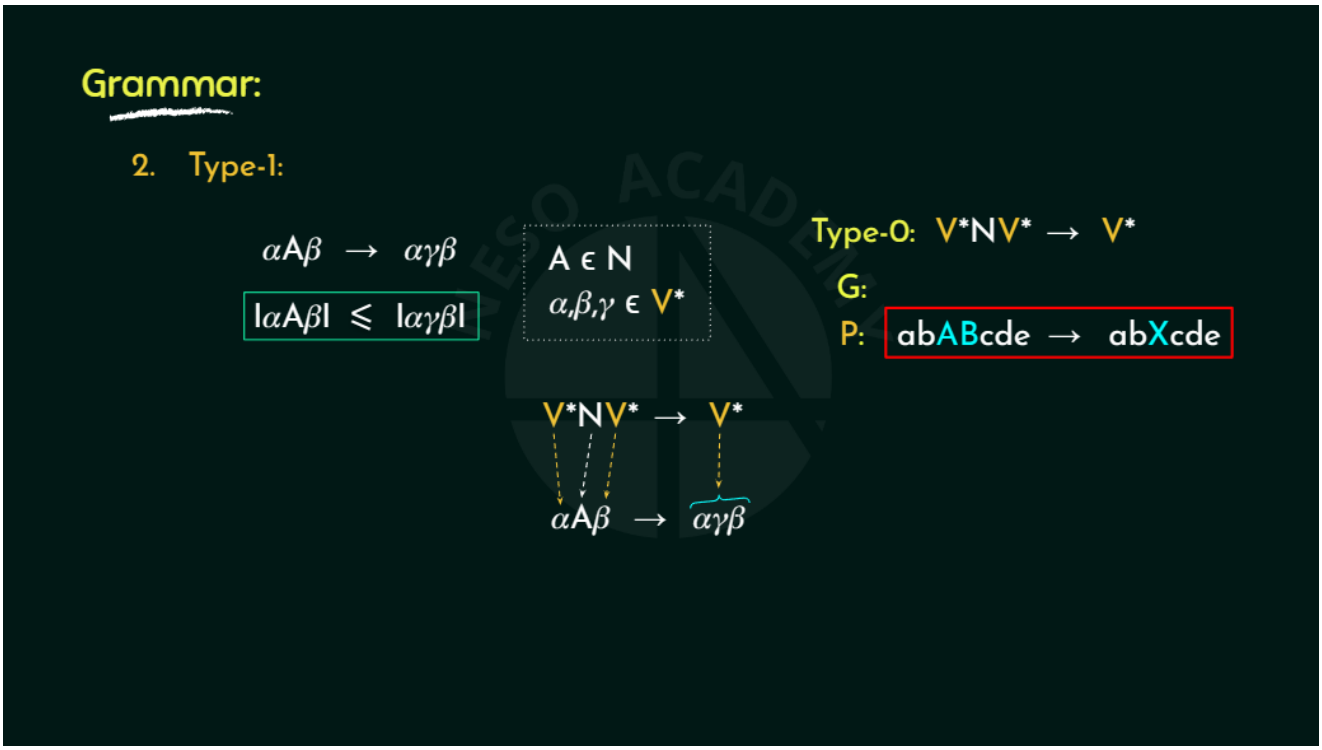
Outcome☆Type-0 and Type-1 Formal Grammars



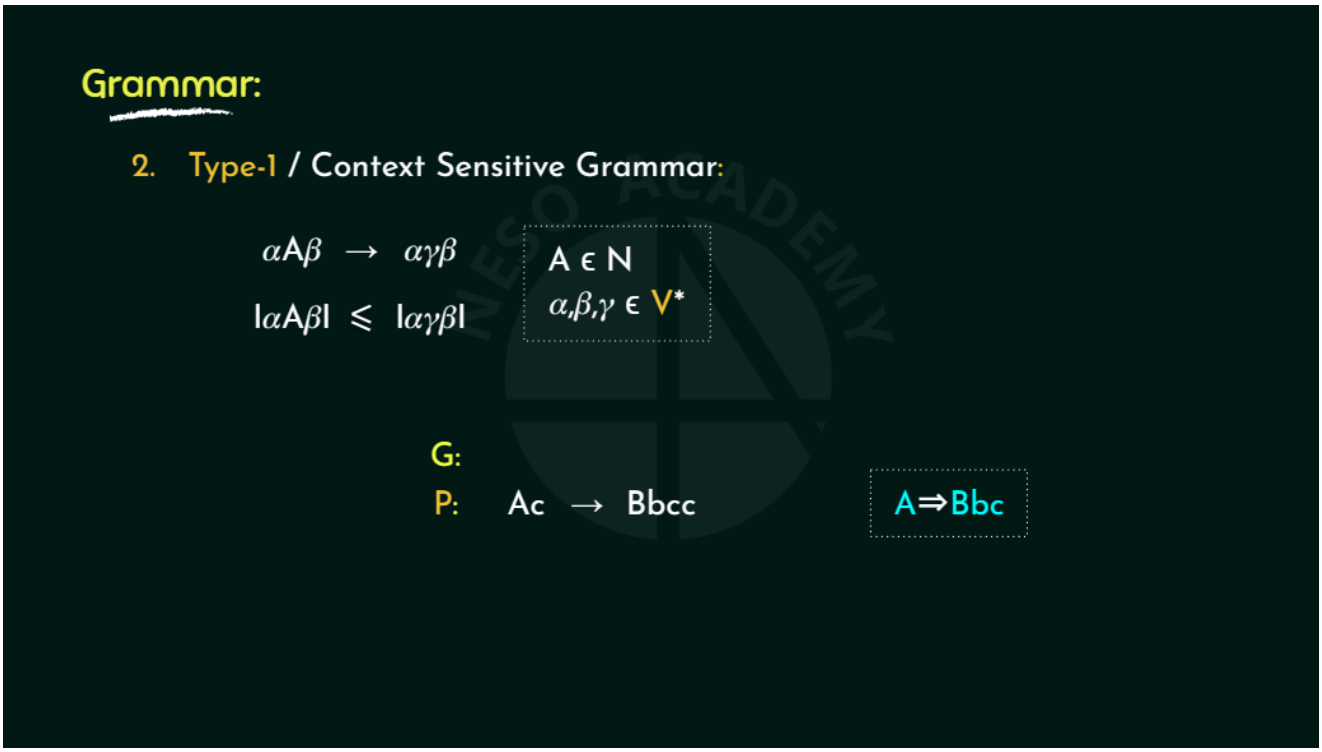
Grammar:A phrase structure grammar (or simply grammar) is (N, T, P, S) where i. N is a finite, non-empty set of Non-terminals.ii. T is a finite, non-empty set of Terminals.iii. N∩T = Φiv. S is a special non-terminal (i.e. S ε N), called Start symbol.v. P is a finite set Production Rules of the form,$\alpha \rightarrow \beta$



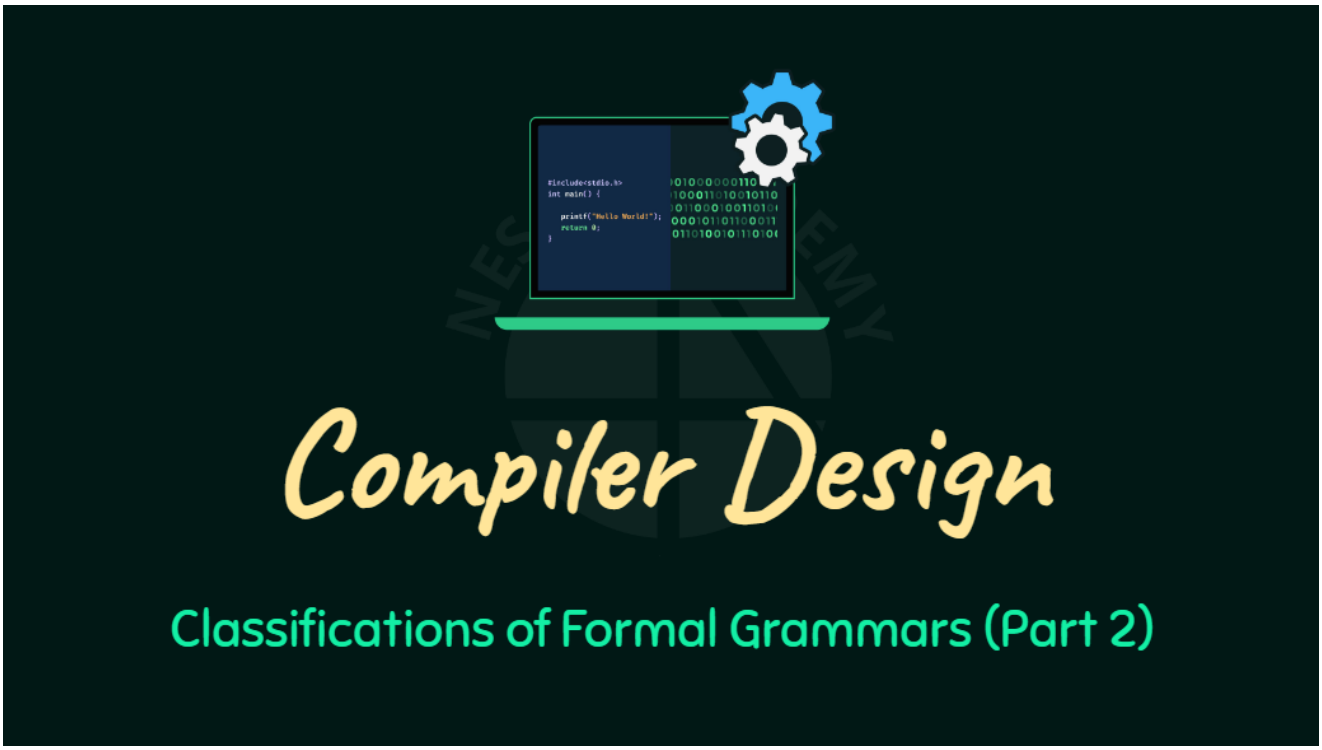
(N∪T)*Grammar:1.Type-0 / Unrestricted Grammar:$\alpha \rightarrow \beta$N∪TV*=V*NV*→V*α ε V*NV*β ε V*G:P:abABcde→abXcdeAB⇒X



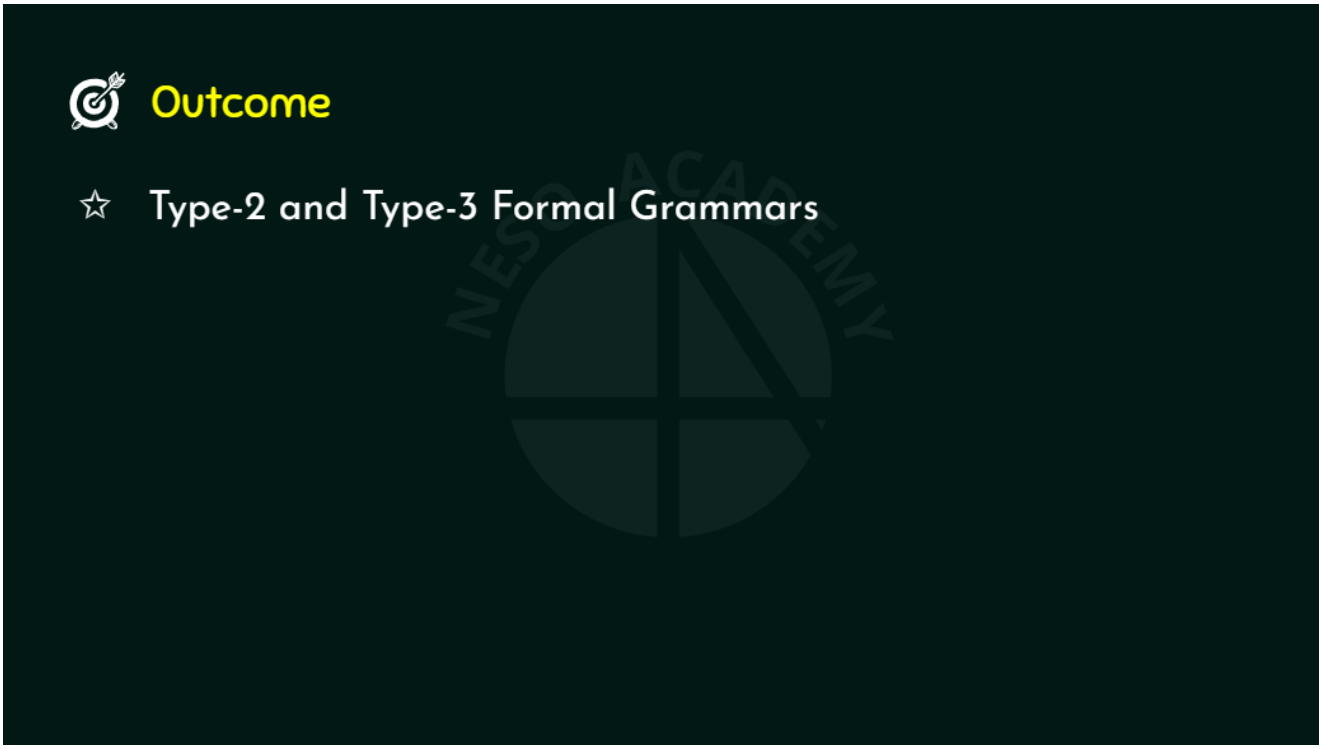
Grammar:2.Type-1:V*NV*→V*G:P:abABcde→abXcdeType-0:α,β,γ ε V* A ε NαAβ→$\alpha\gamma\beta$|αAβ||$\alpha\gamma\beta$|␣ӁV*NV*→V*αAβ→$\alpha\gamma\beta$



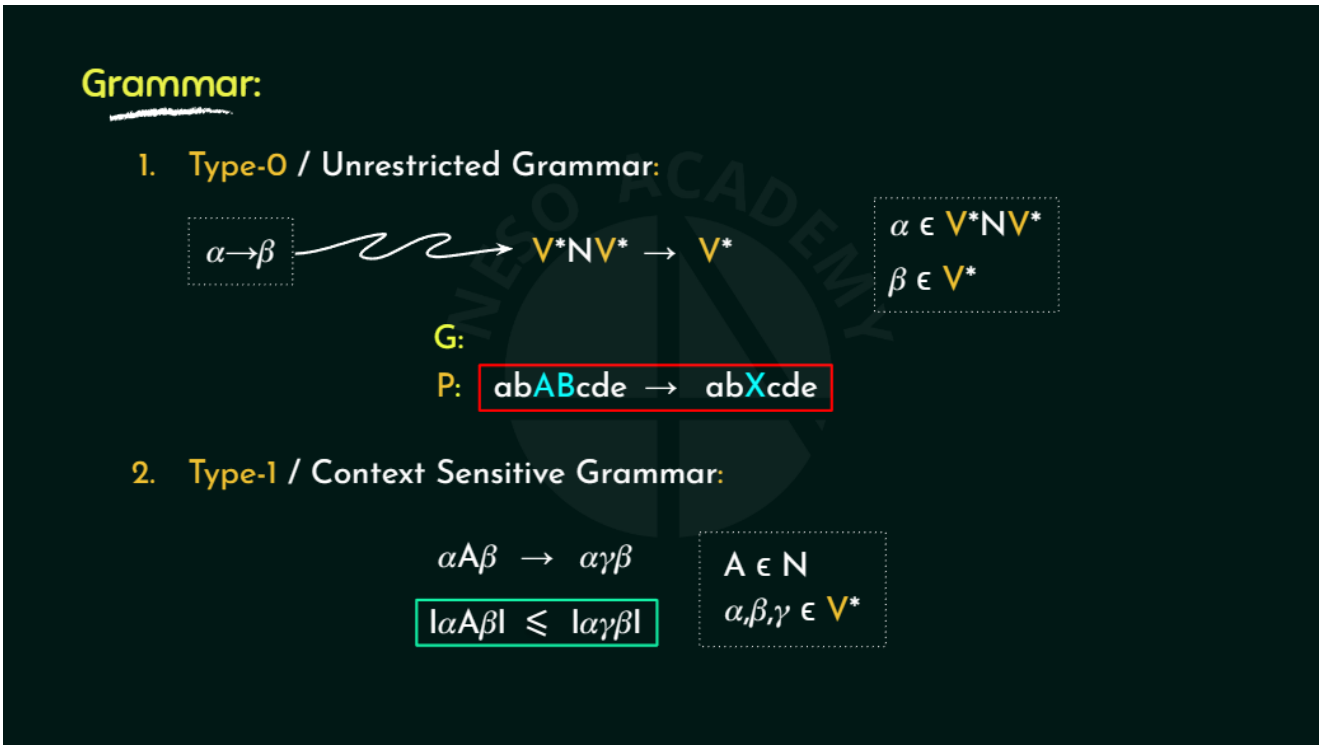
Grammar:2.Type-1 / Context Sensitive Grammar:α,β,γ ε V* A ε NαAβ→$\alpha\gamma\beta$|αAβ||$\alpha\gamma\beta$|␣ӁG:P:Ac→BbccA⇒Bbc



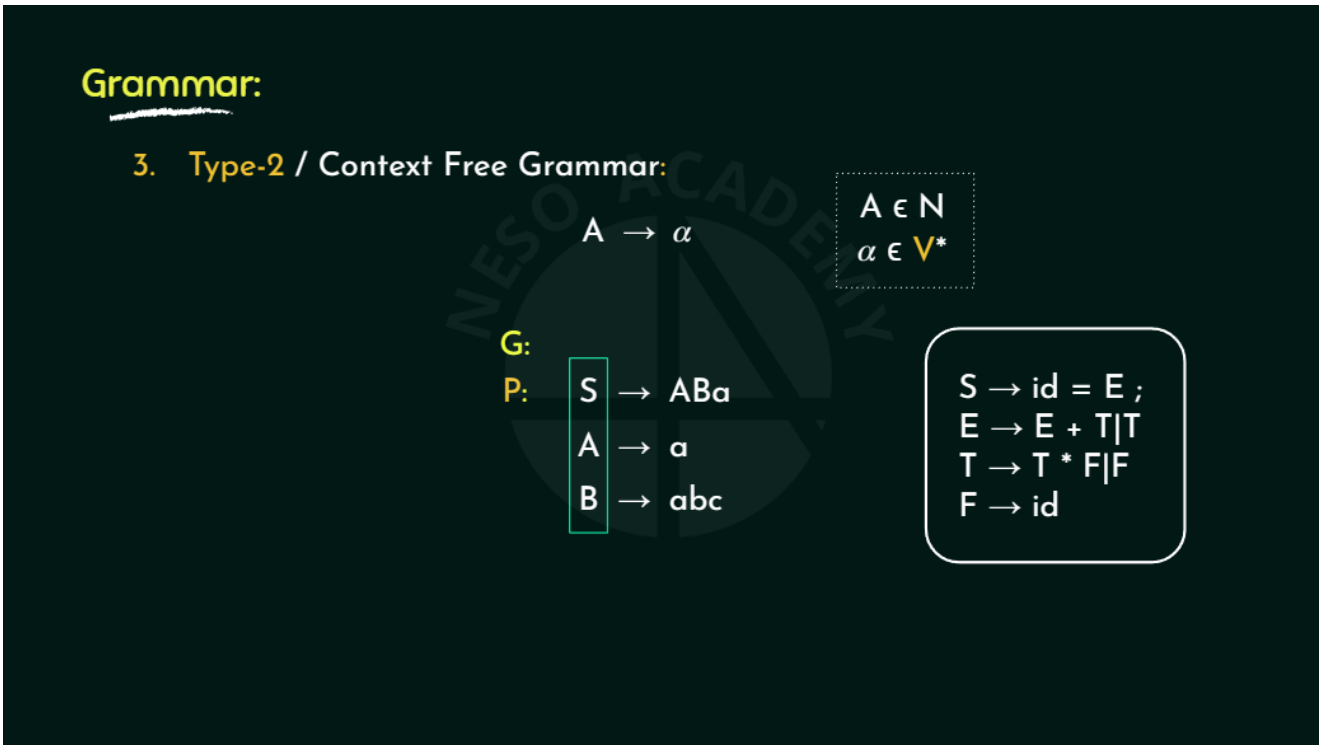
Compiler DesignClassifications of Formal Grammars (Part 2)



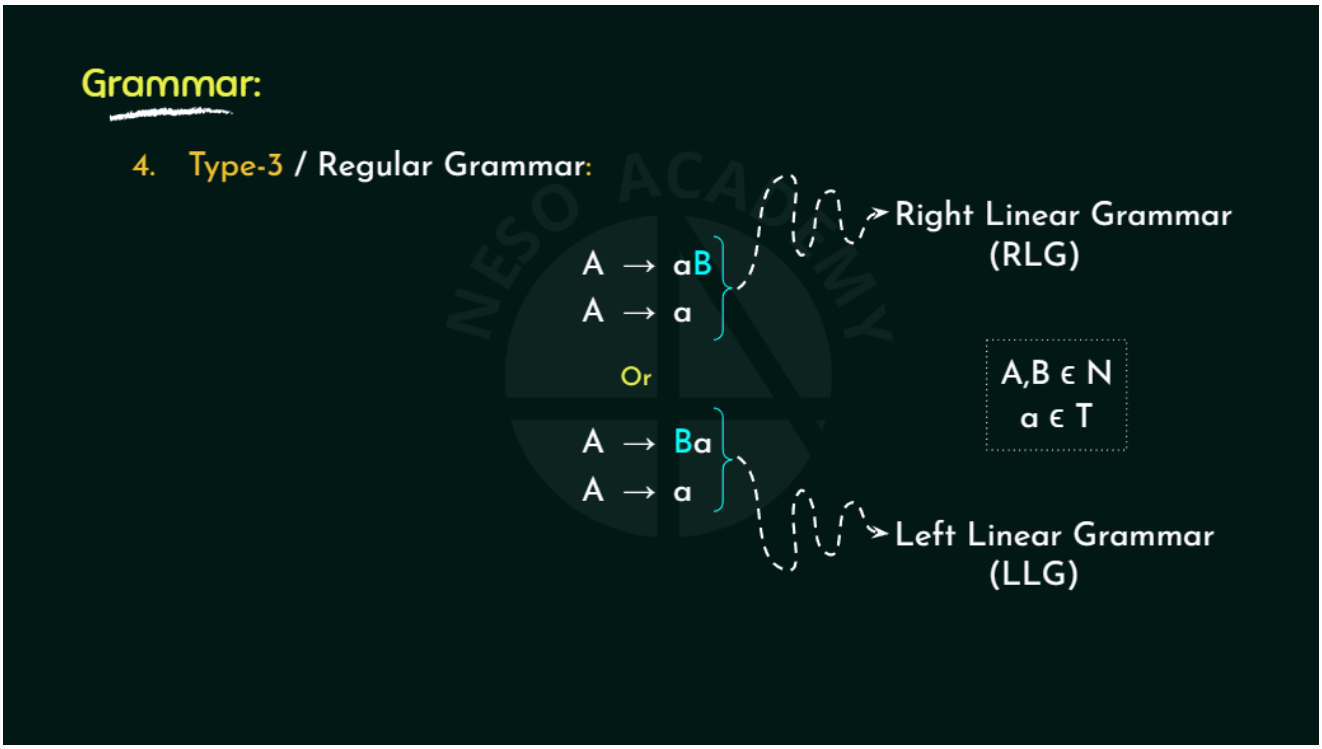
Outcome☆Type-2 and Type-3 Formal Grammars



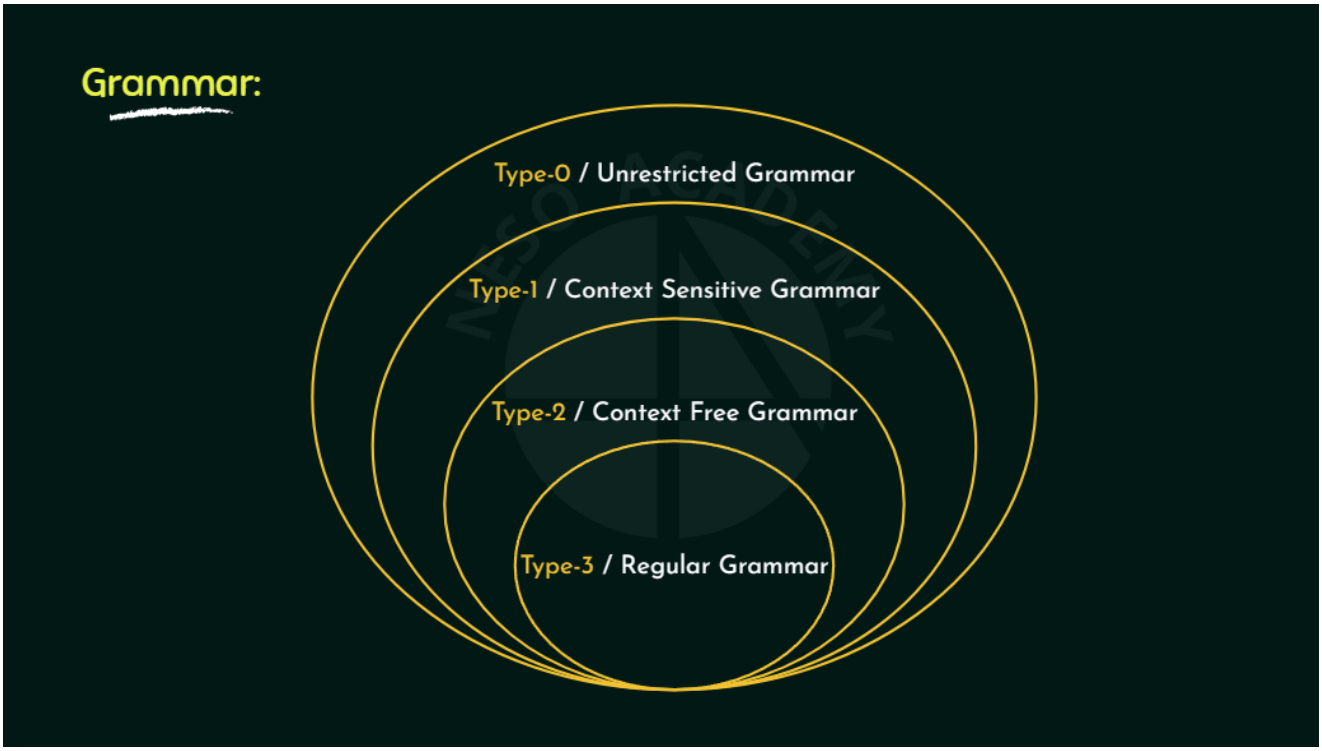
Grammar:1.Type-0 / Unrestricted Grammar:$\alpha \rightarrow \beta$V*NV*→V*α ε V*NV*β ε V*G:P:abABcde→abXcde2.Type-1 / Context Sensitive Grammar:α, β, γ ε V* A ε NαAβ→$\alpha\gamma\beta$|αAβ||$\alpha\gamma\beta$|␣Ж



Grammar:3.Type-2 / Context Free Grammar:α ε V* A ε Nα→G:P:S→ABaA→aB→abc S → id = E ; E → E + T|T T → T * F|F F → id

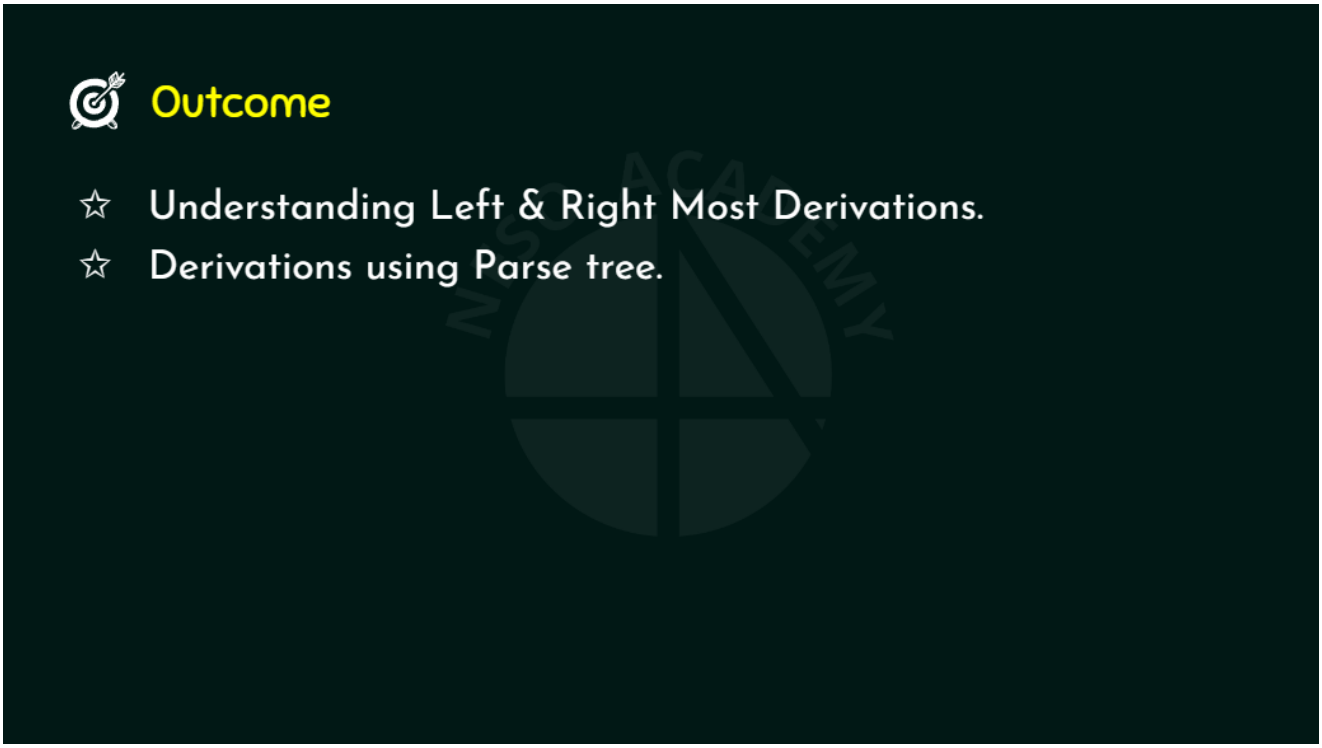


a ε T A,B ε NGrammar:4.Type-3 / Regular Grammar:AaB→Aa→ABa→Aa→OrRight Linear Grammar (RLG)Left Linear Grammar (LLG)

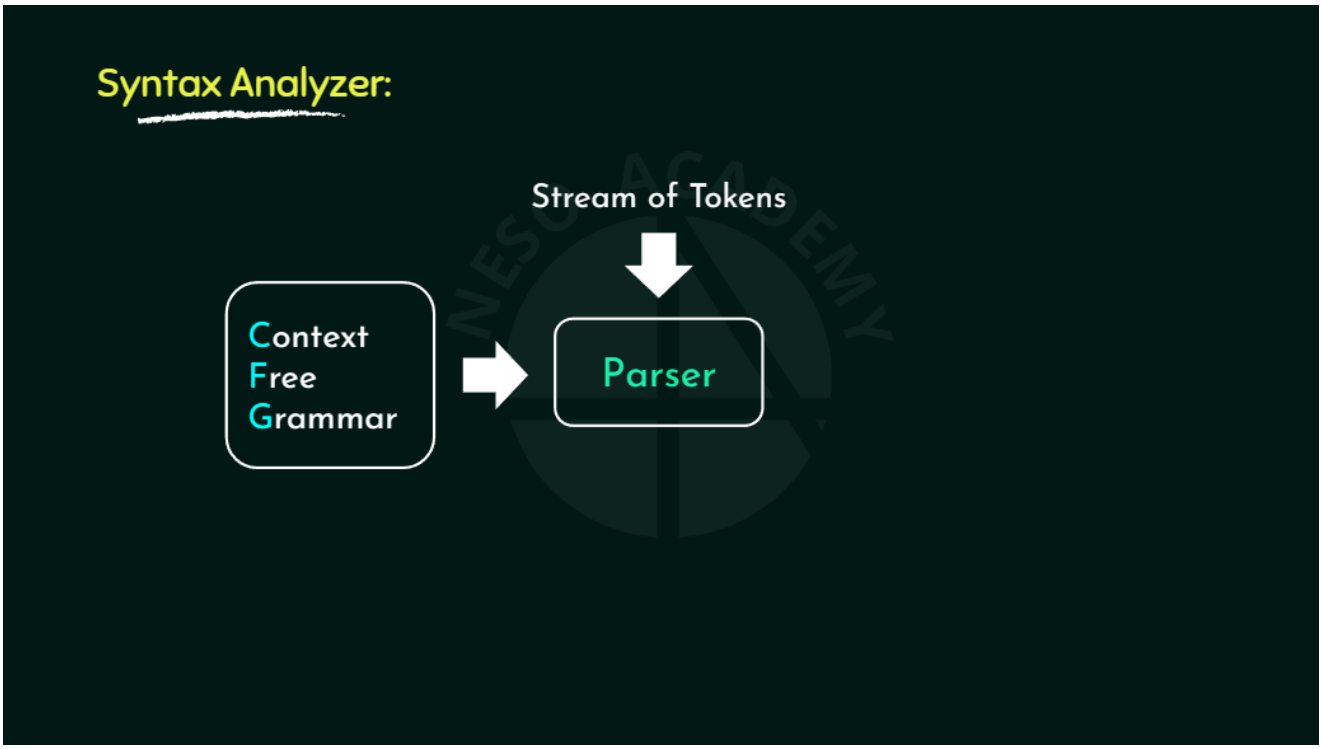


Type-0 / Unrestricted GrammarGrammar:Type-1 / Context Sensitive GrammarType-2 / Context Free GrammarType-3 / Regular Grammar

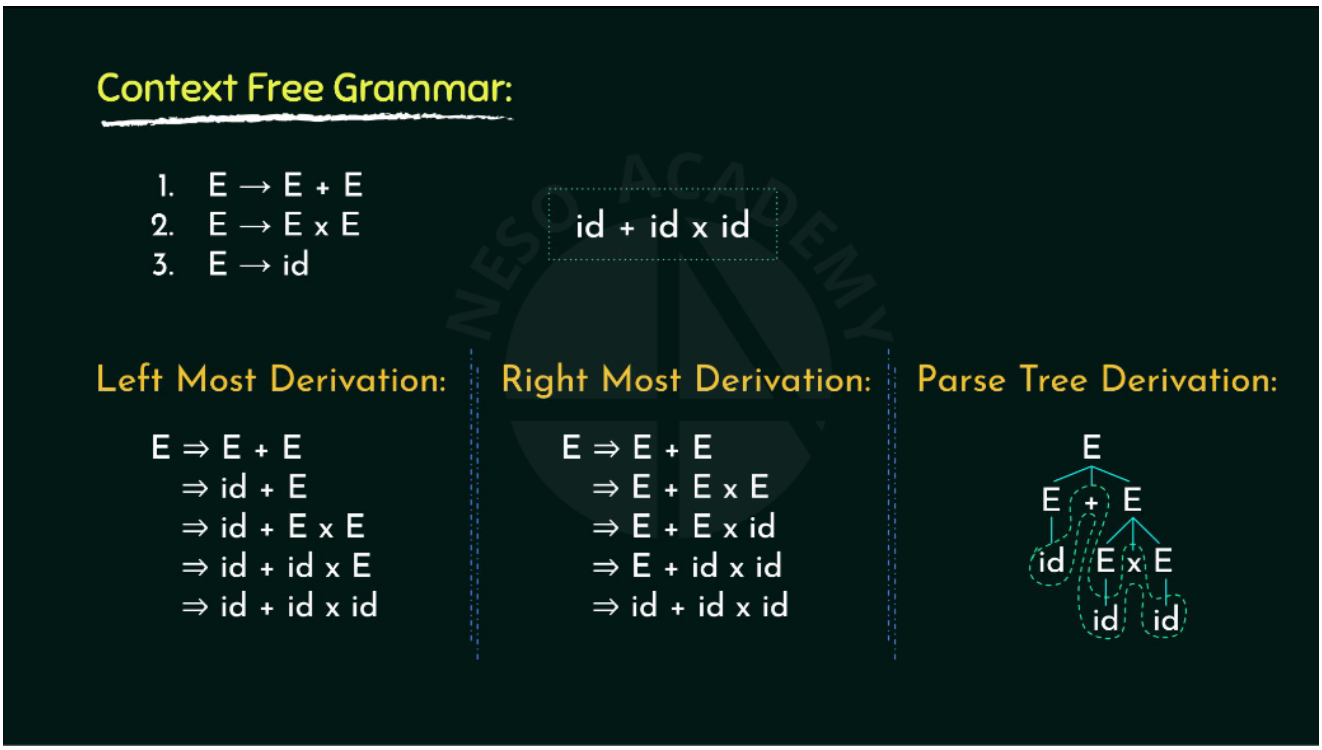
Compiler DesignDerivations of CFGs

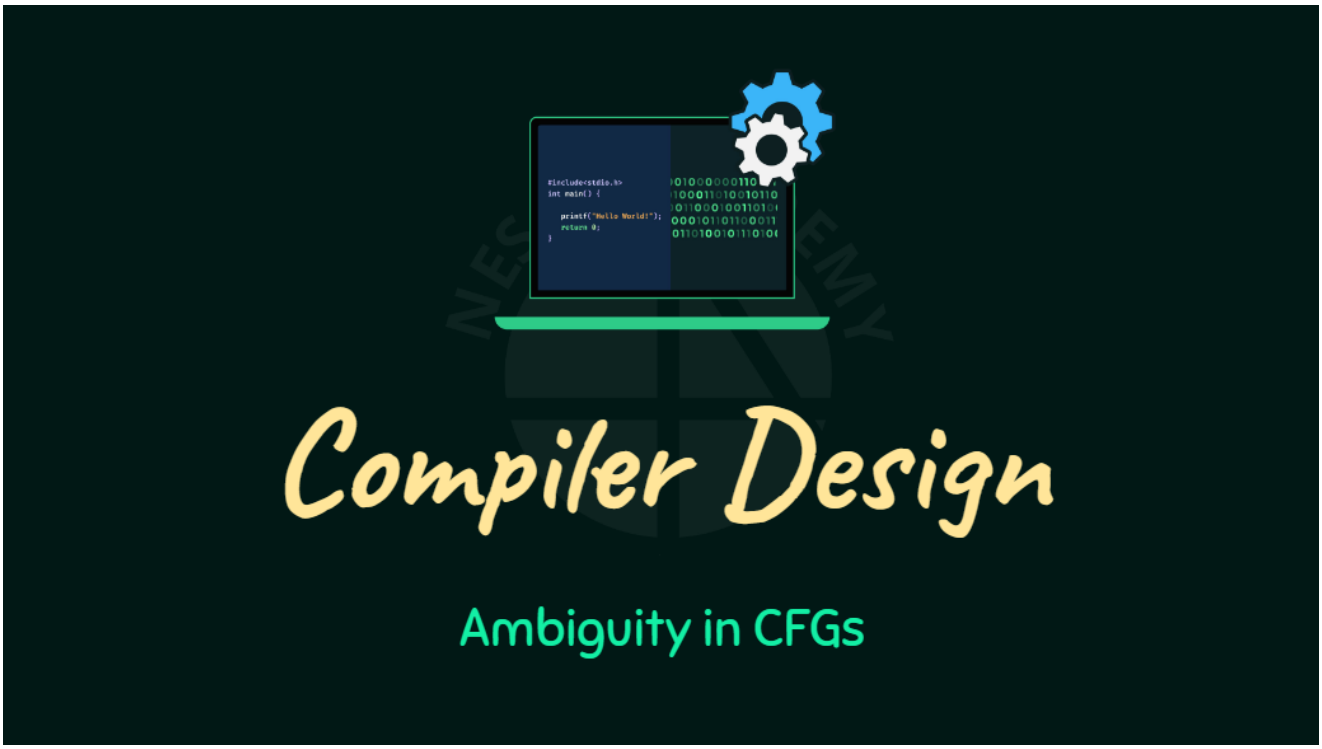


Outcome☆Understanding Left & Right Most Derivations.☆Derivations using Parse tree.

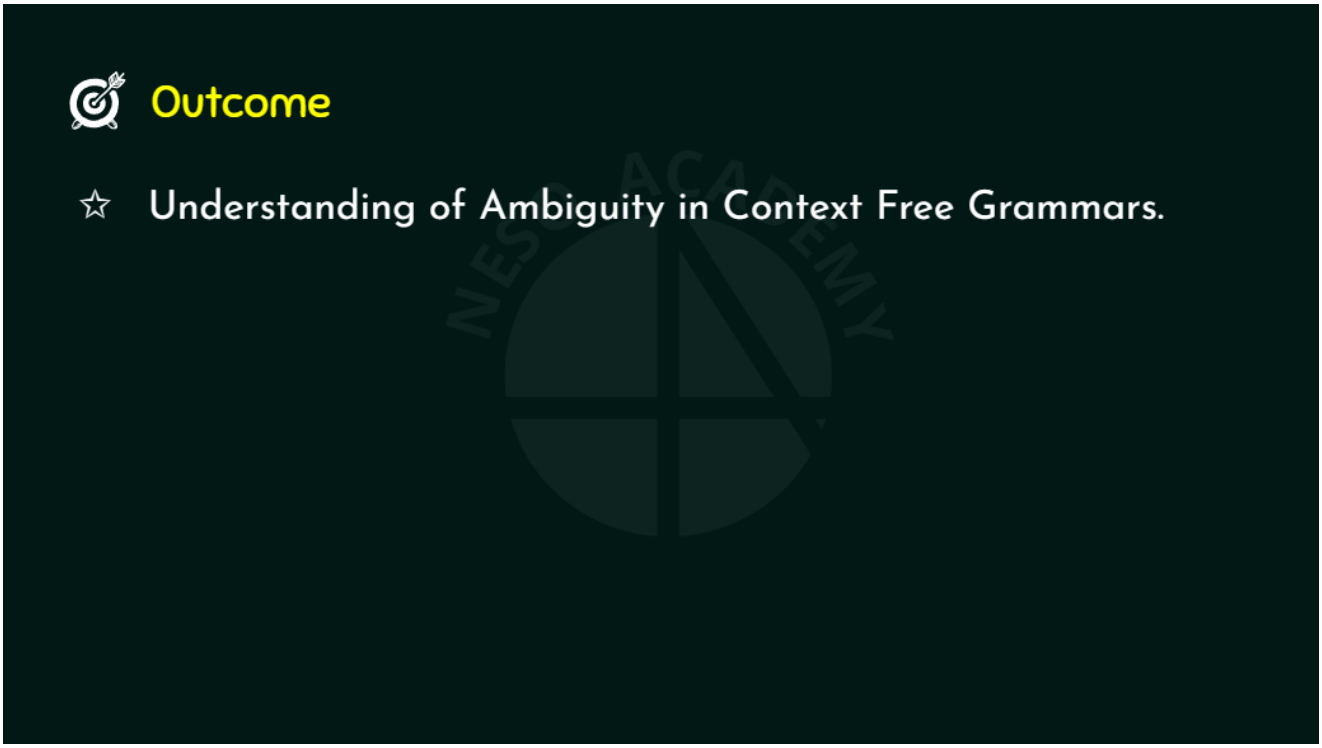


Syntax Analyzer:ParserStream of TokensContextFreeGrammar





Compiler DesignAmbiguity in CFGs



Outcome☆Understanding of Ambiguity in Context Free Grammars.

Context Free Grammar:

1. E → E + E
2. E → E x E
3. E → id

id + id x id

Left Most Derivation:

E ⇒ E + E
⇒ id + E
⇒ id + E x E
⇒ id + id x E
⇒ id + id x id

Right Most Derivation:

E ⇒ E + E
⇒ E + E x E
⇒ E + E x id
⇒ E + id x id
⇒ id + id x id

Parse Tree Derivation:

Context Free Grammar:

E → E + E | E x E | id

id + id x id

Left Most Derivation:

E ⇒ E + E
⇒ id + E
⇒ id + E x E
⇒ id + id x E
⇒ id + id x id

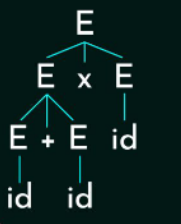
E ⇒ E x E
⇒ E + E x E
⇒ id + E x E
⇒ id + id x E
⇒ id + id x id

Right Most Derivation:

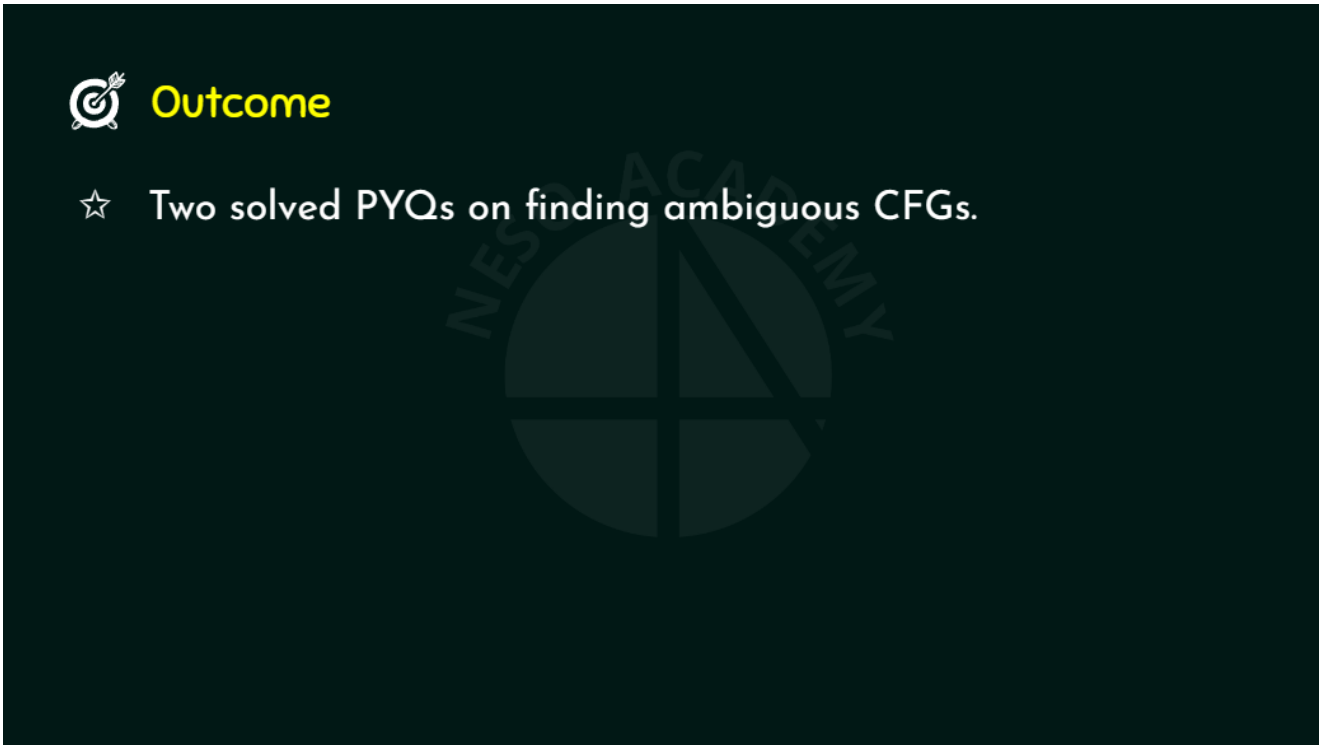
E ⇒ E + E
⇒ E + E x E
⇒ E + E x id
⇒ E + id x id
⇒ id + id x id

E ⇒ E x E
⇒ E x id
⇒ E + E x id
⇒ E + id x id
⇒ id + id x id

Parse Tree Derivation:



Compiler DesignAmbiguity in CFGs - Solved Problems (Set 1)



Outcome☆Two solved PYQs on finding ambiguous CFGs.

Q1: Determine whether the following grammar is suitable for predictive parsing,

$$S \rightarrow (S) \mid SS \mid \varepsilon$$

UGC-NET DEC 2018

Sol. Parse Tree Derivation: ()()()

Q1:Determine whether the following grammar is suitable for predictive parsing,S → (S) I SS I εSol.UGC-NETDEC 2018Parse Tree Derivation:()()()SSS(S)(S)(S)SSεεεSSS(S)ε(S)(S)SSεε

Q2: Consider the following two Grammars,

G1: $S \rightarrow SbS \mid a$

G2: $S \rightarrow aB \mid ab$, $A \rightarrow AB \mid a$, $B \rightarrow ABb \mid b$

UGC-NET JUNE 2018

Which of the following option is correct?

(A) Only G1 is ambiguous

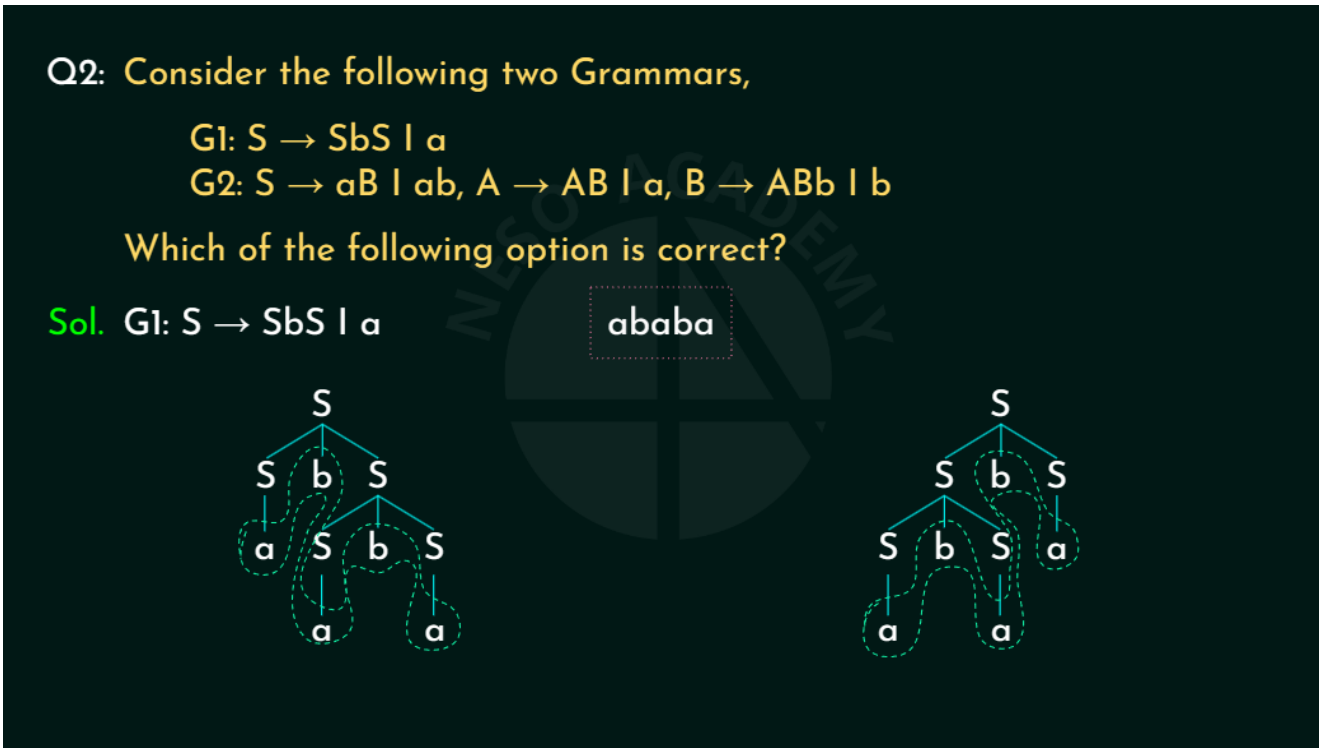
(B) Only G2 is ambiguous

(C) Both G1 and G2 are ambiguous

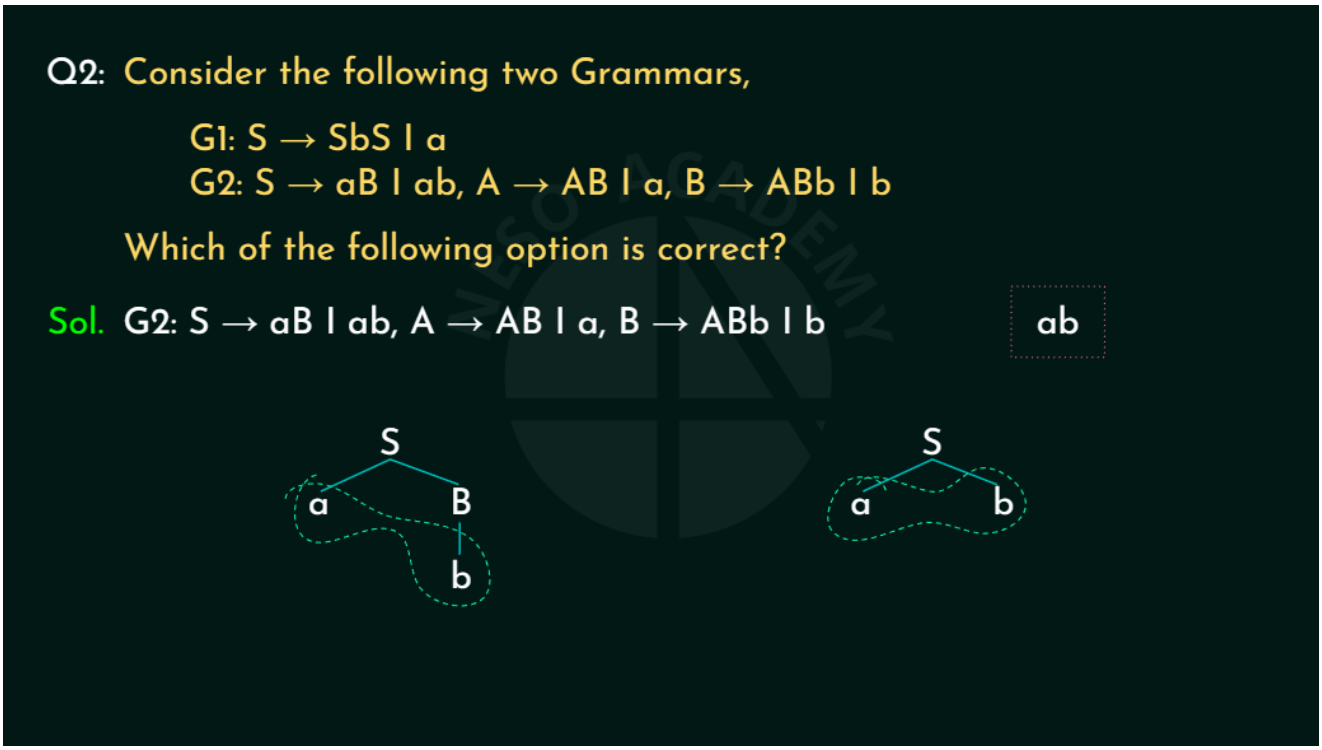
(D) Both G1 and G2 are unambiguous

Q2:Consider the following two Grammars,G1: S → SbS I aG2: S → aB I ab, A → AB I a, B → ABb I bWhich of the following option is correct?UGC-NETJUNE 2018(A) Only G1 is ambiguous(B) Only G2 is ambiguous(C) Both G1 and G2 are ambiguous(D) Both G1 and G2

are unambiguous

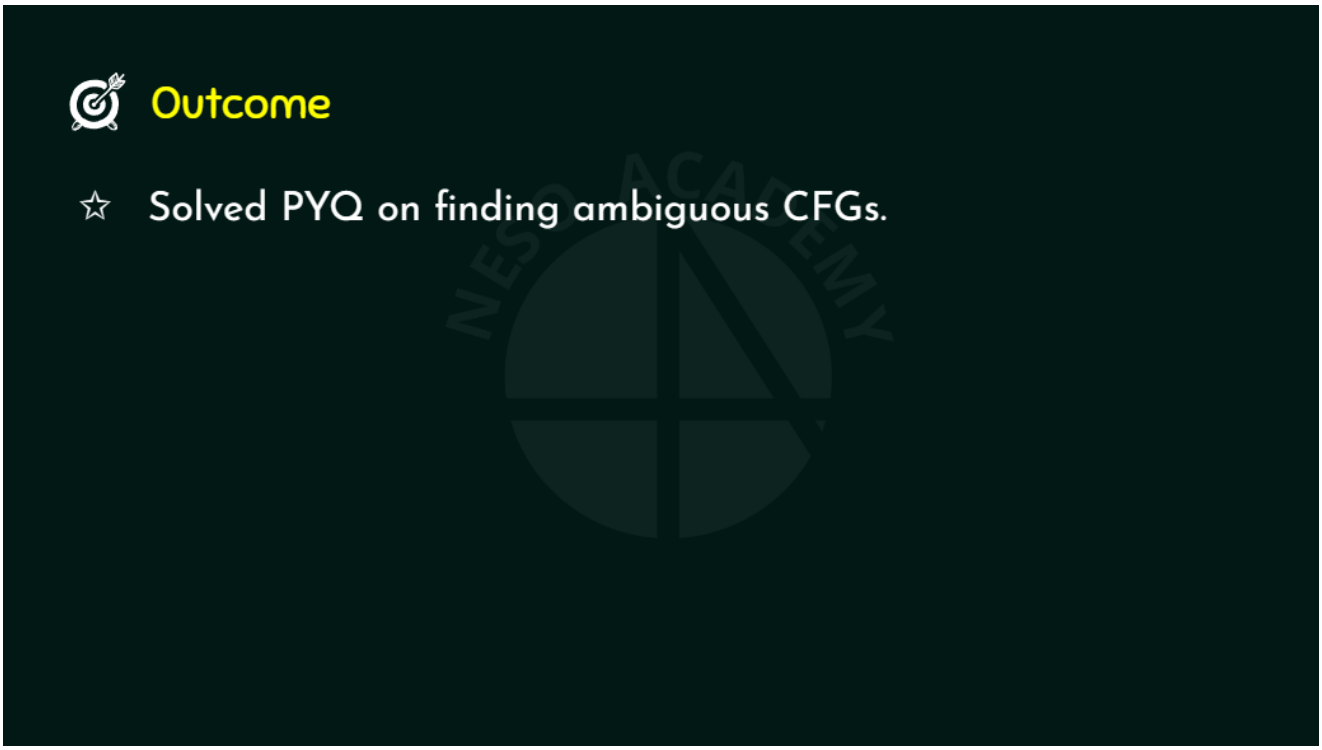


Q2:Consider the following two Grammars,G1: S → SbS I aG2: S → aB I ab, A → AB I a, B → ABb I bWhich of the following option is correct?Sol.G1: S → SbS I aababaSS b SS b SaaaSS b SS b Saaa

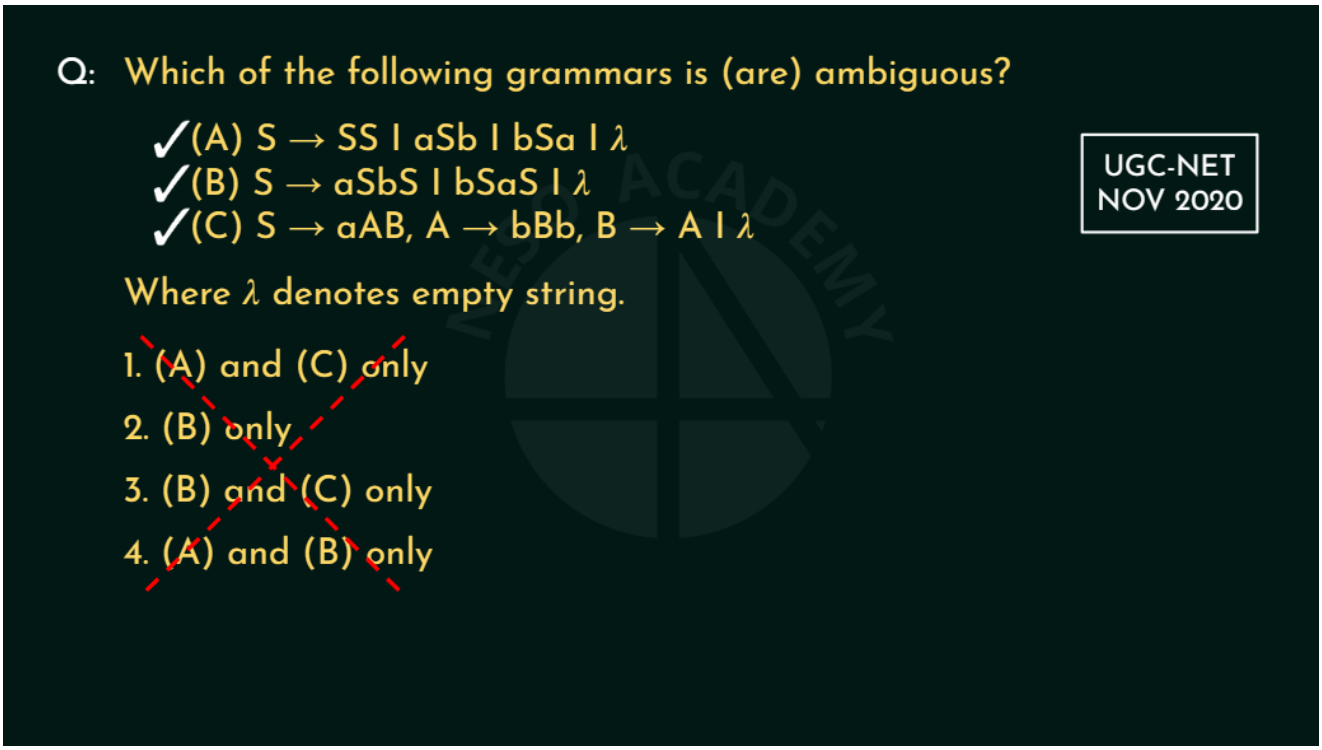


Q2:Consider the following two Grammars,G1: S → SbS I aG2: S → aB I ab, A → AB I a, B → ABb I bWhich of the following option is correct?Sol.G2: S → aB I ab, A → AB I a, B → ABb I babSaBbSab

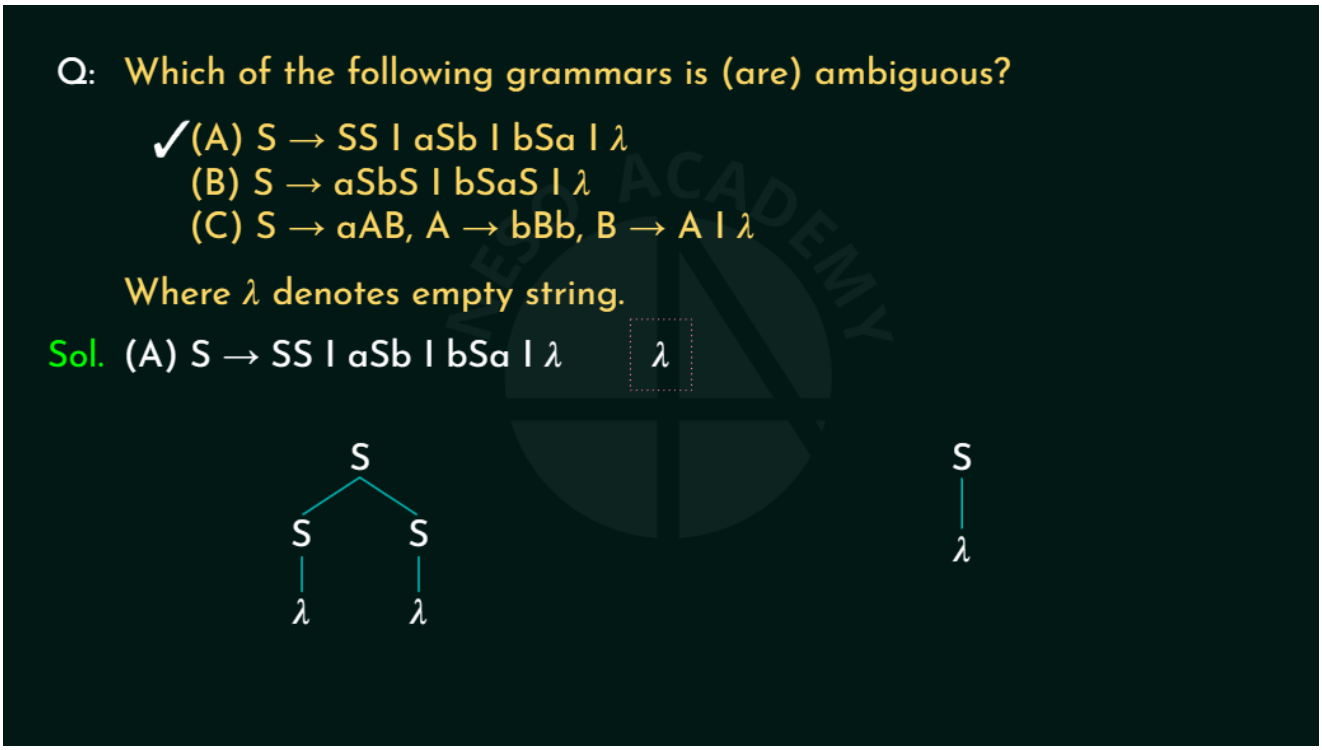
Compiler DesignAmbiguity in CFGs - Solved Problems (Set 2)



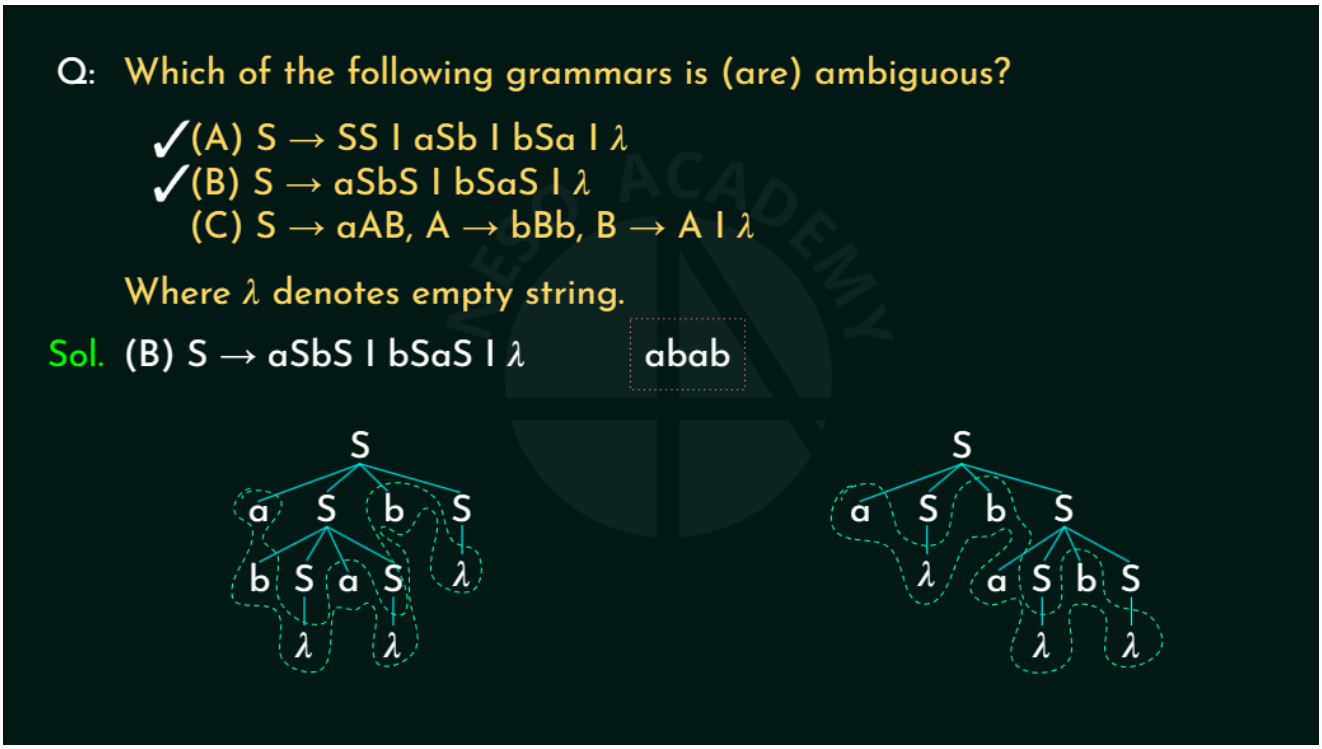
Outcome☆Solved PYQ on finding ambiguous CFGs.



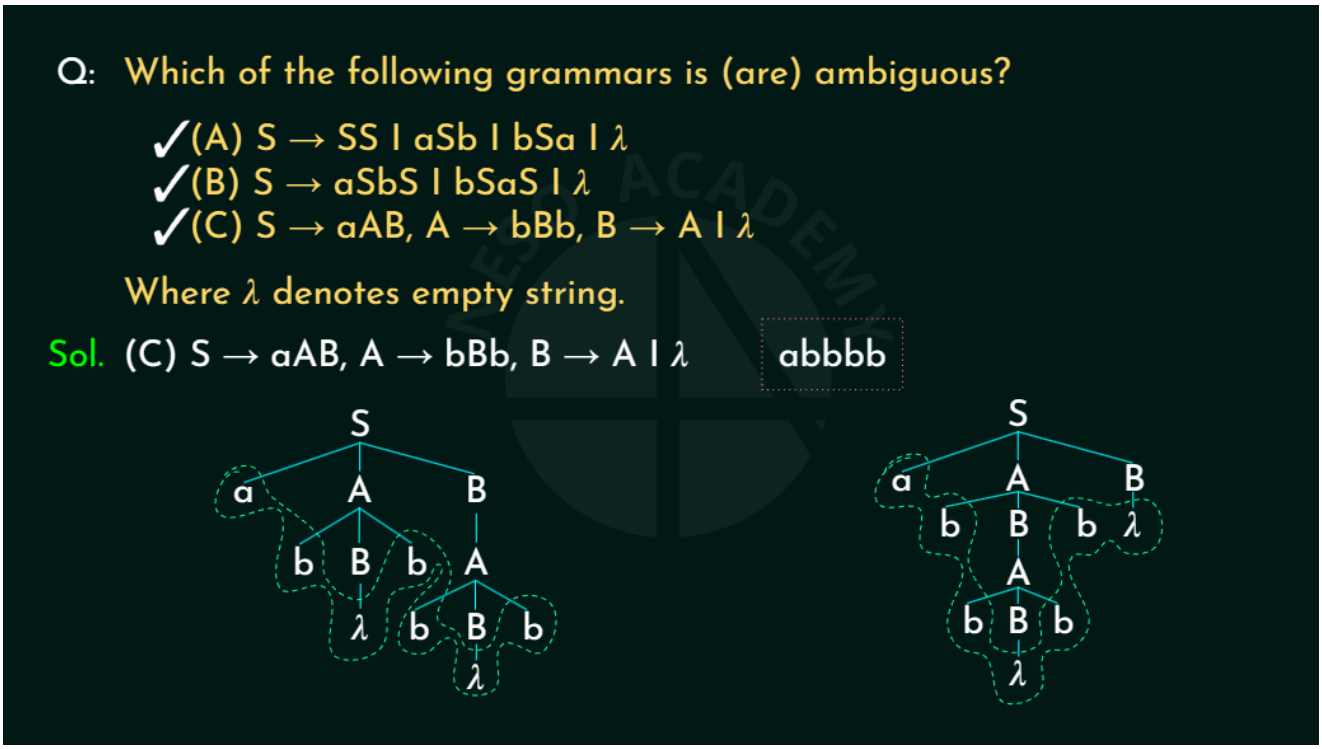
Q:Which of the following grammars is (are) ambiguous?(A) S → SS I aSb I bSa I λ(B) S → aSbS I bSaS I λ(C) S → aAB, A → bBb, B → A I λWhere λ denotes empty string.UGC-NETNOV 20201. (A) and (C) only2. (B) only3. (B) and (C) only4. (A) and (B) only



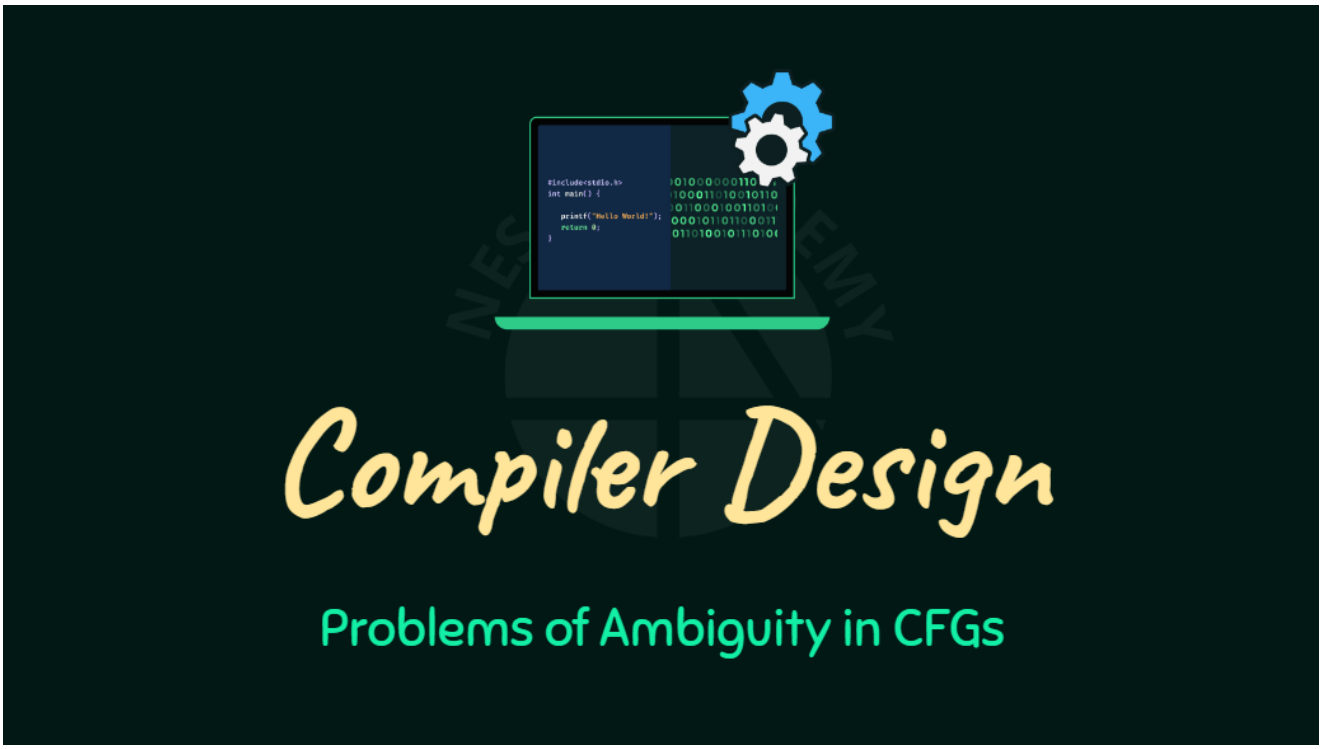
Q:Which of the following grammars is (are) ambiguous?(A) S → SS I aSb I bSa I λ(B) S → aSbS I bSaS I λ(C) S → aAB, A → bBb, B → A I λWhere λ denotes empty string.Sol.(A) S → SS I aSb I bSa I λλSSSSλλλ



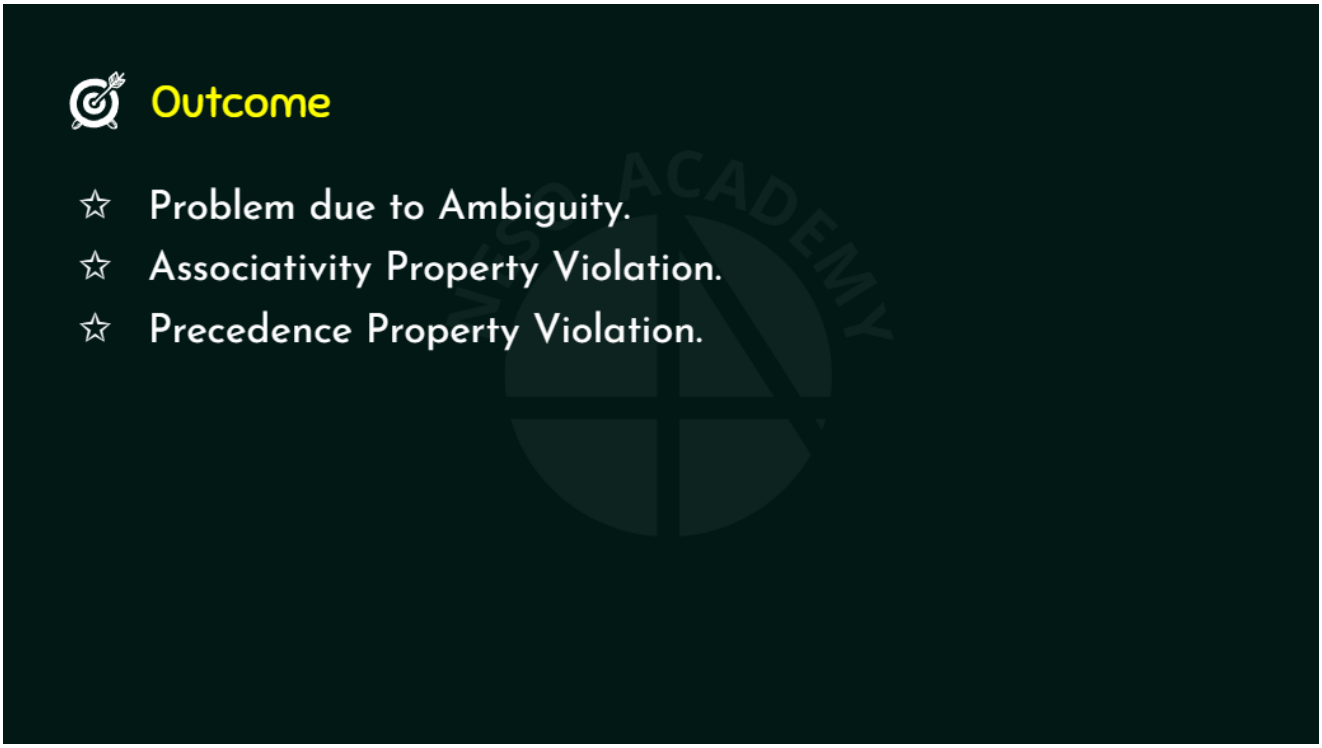
Q:Which of the following grammars is (are) ambiguous?(A) S → SS I aSb I bSa I λ(B) S → aSbS I bSaS I λ(C) S → aAB, A → bBb, B → A I λWhere λ denotes empty string.Sol.(B) S → aSbS I bSaS I λababSSλa S b Sb S a Sλλa S b Sa S b Sλλλ



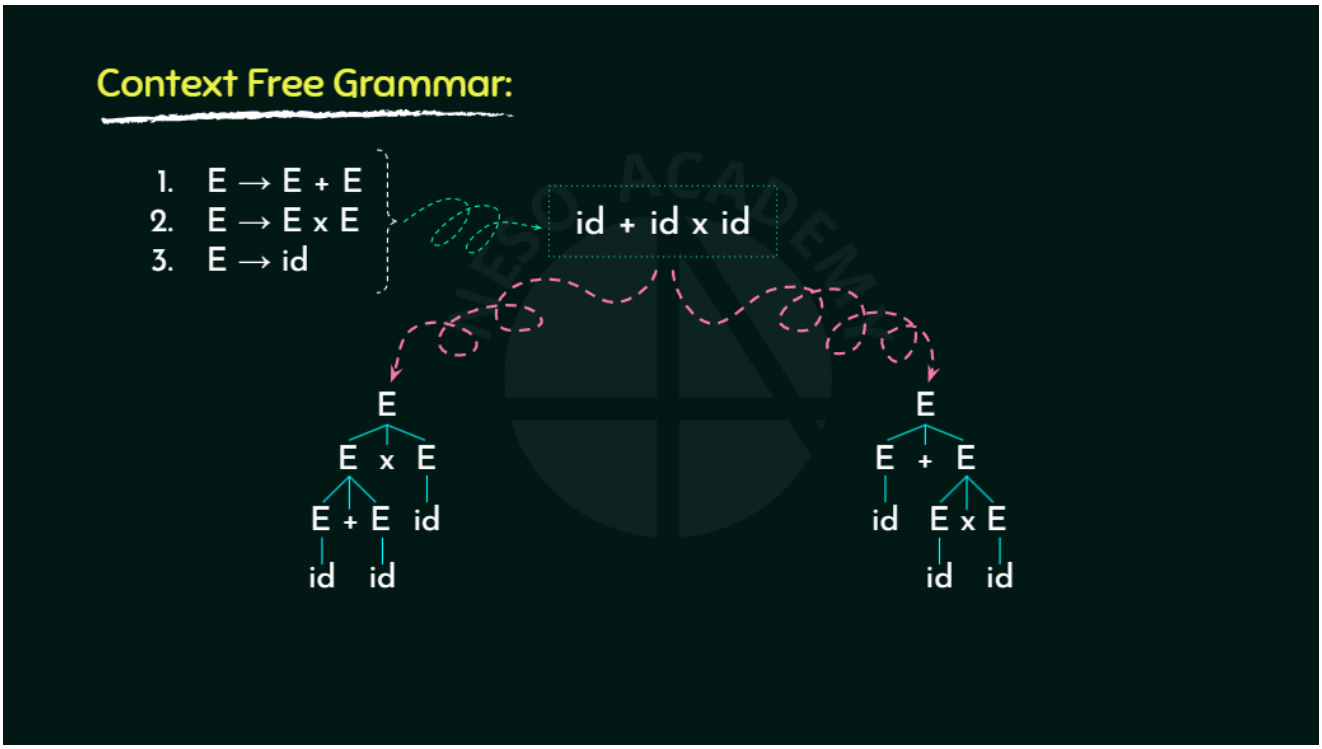
Q:Which of the following grammars is (are) ambiguous?(A) S → SS I aSb I bSa I λ(B) S → aSbS I bSaS I λ(C) S → aAB, A → bBb, B → A I λWhere λ denotes empty string.Sol.(C) S → aAB, A → bBb, B → A I λabbbbSλAλa A Bb B bb B bSa A Bλb B bAb B bλ



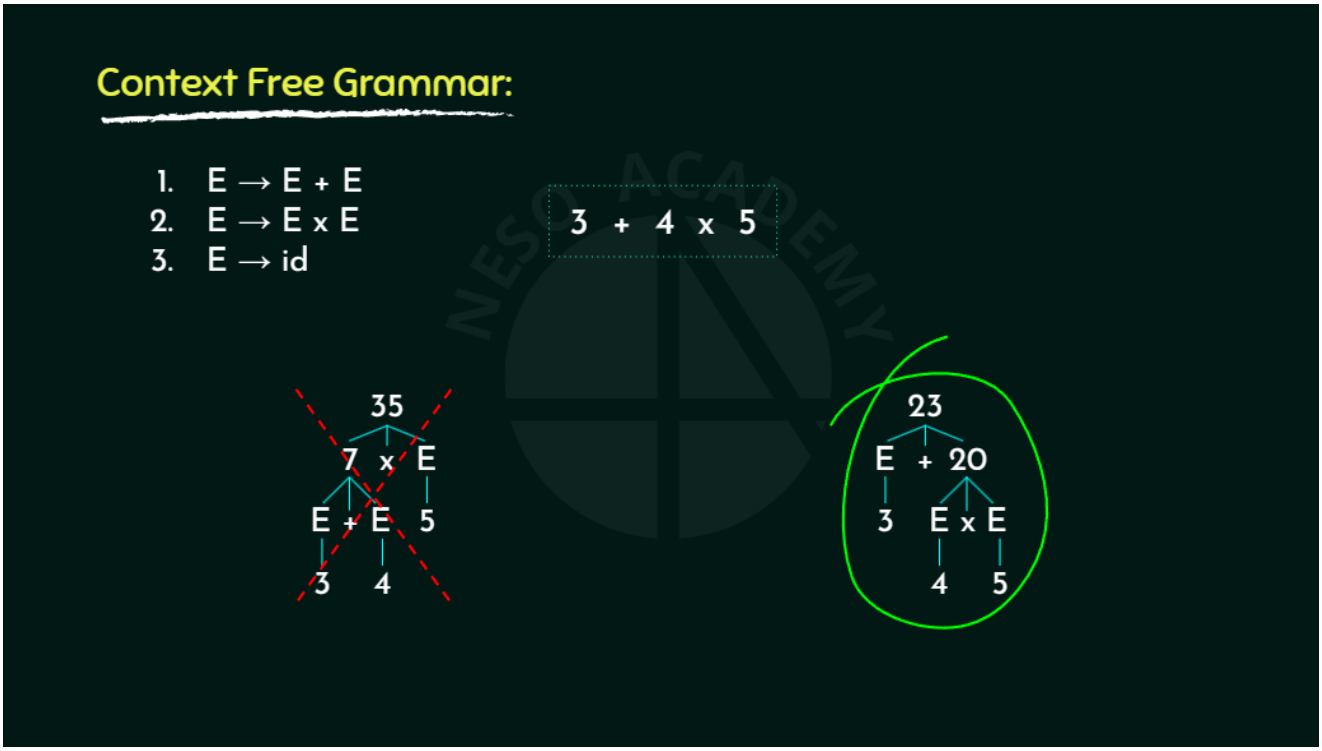
Compiler DesignProblems of Ambiguity in CFGs



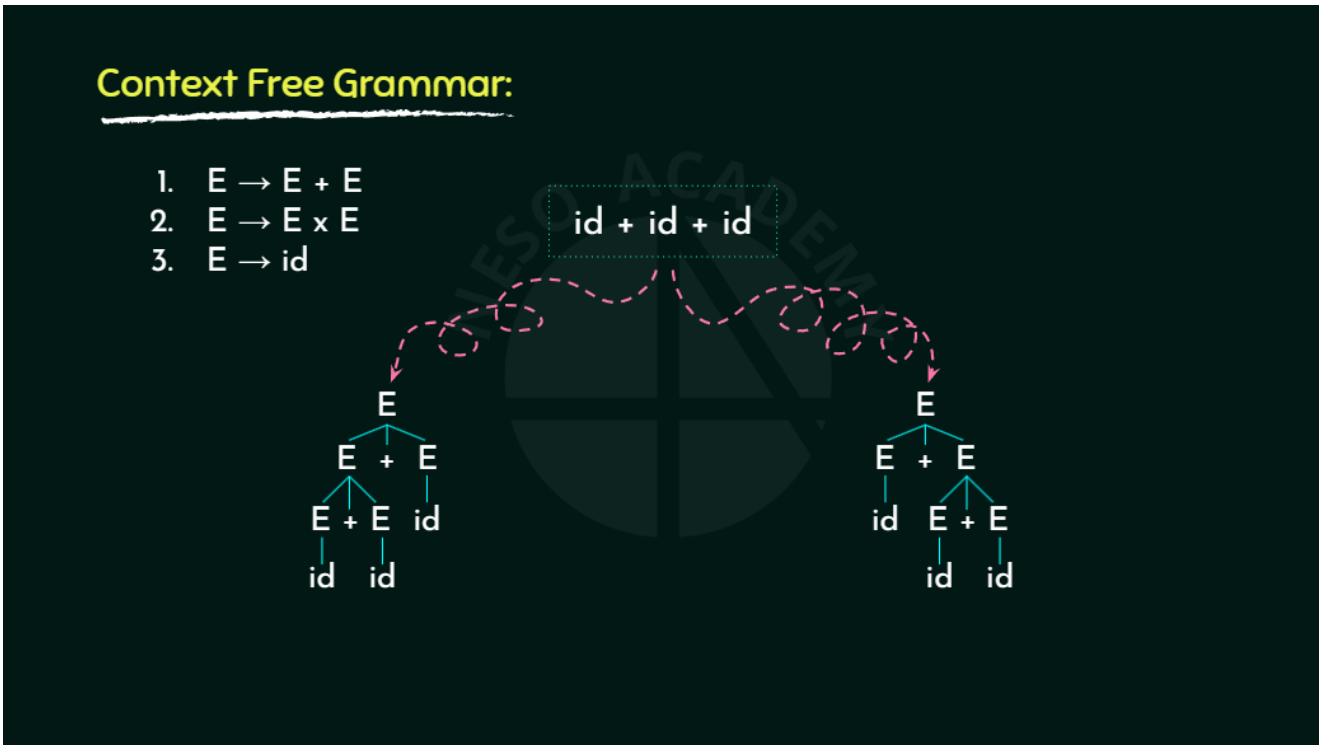
Outcome☆Problem due to Ambiguity.☆Associativity Property Violation.☆Precedence Property Violation.



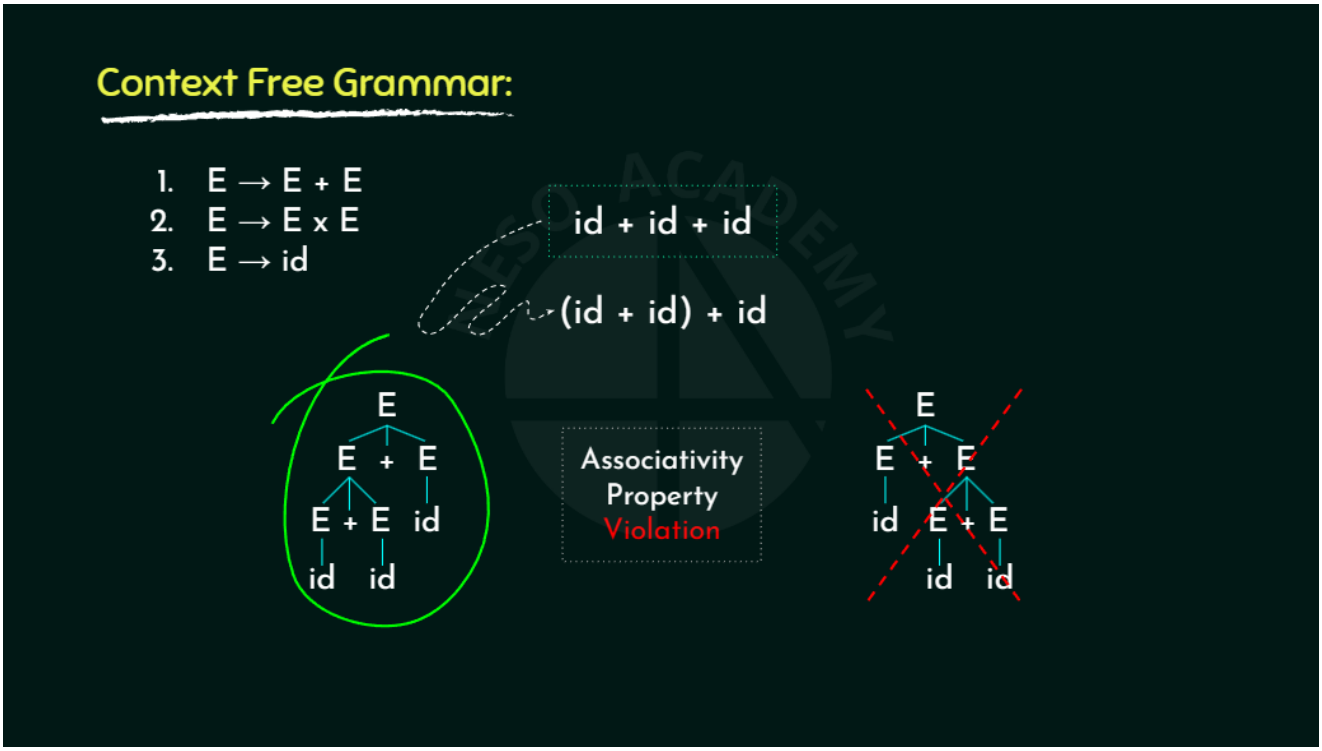
E E + Eid id id E x EE E x Eid id id E + E1.E → E + E2.E → E x E3.E → idContext Free Grammar:id + id x id



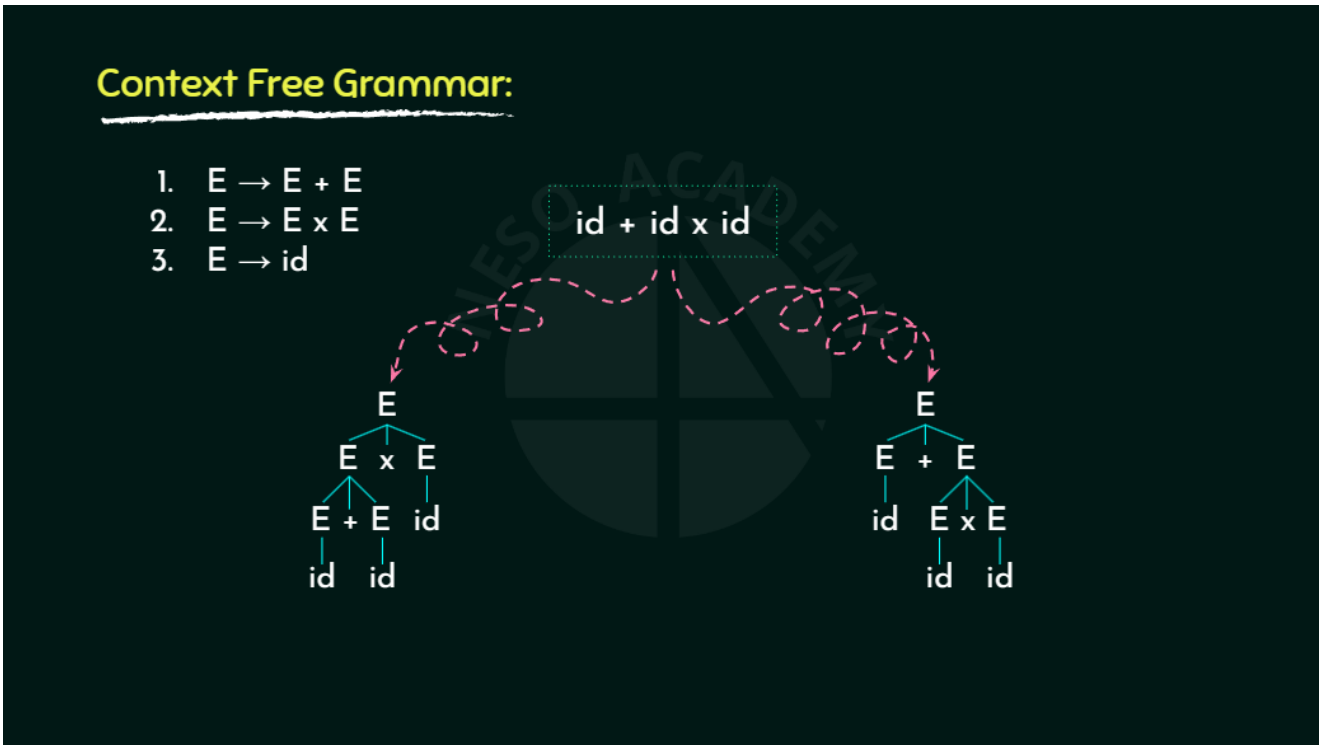
1.E → E + E2.E → E x E3.E → idContext Free Grammar:3 + 4 x 535 E x E5 3 4 E + E7 23 E + E34 5 E x E20



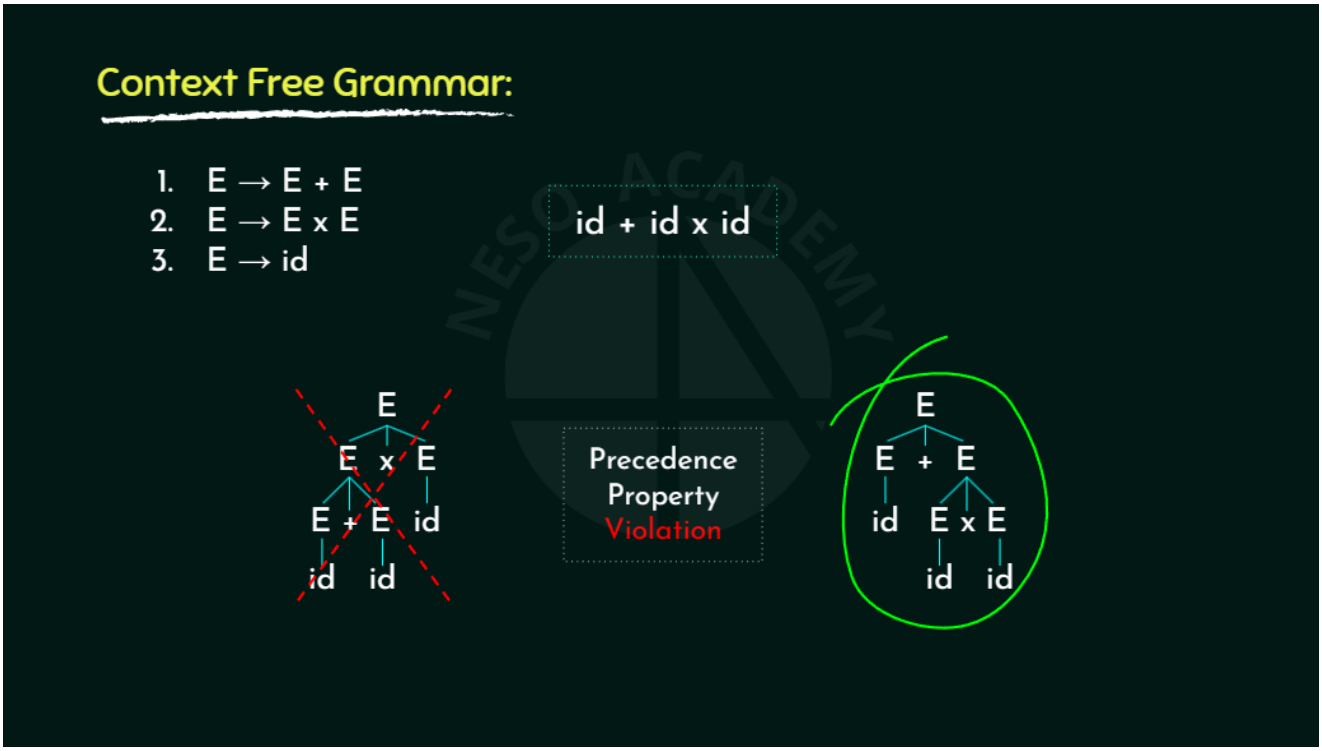
E E + Eid id id E + EE E + Eid id id E + E1.E → E + E2.E → E x E3.E → idContext Free Grammar:id + id + id



+ idE E + Eid id id E + EE E + Eid id id E + E1.E → E + E2.E → E x E3.E → idContext Free Grammar:id + id + id(id + id)Associativity PropertyViolation

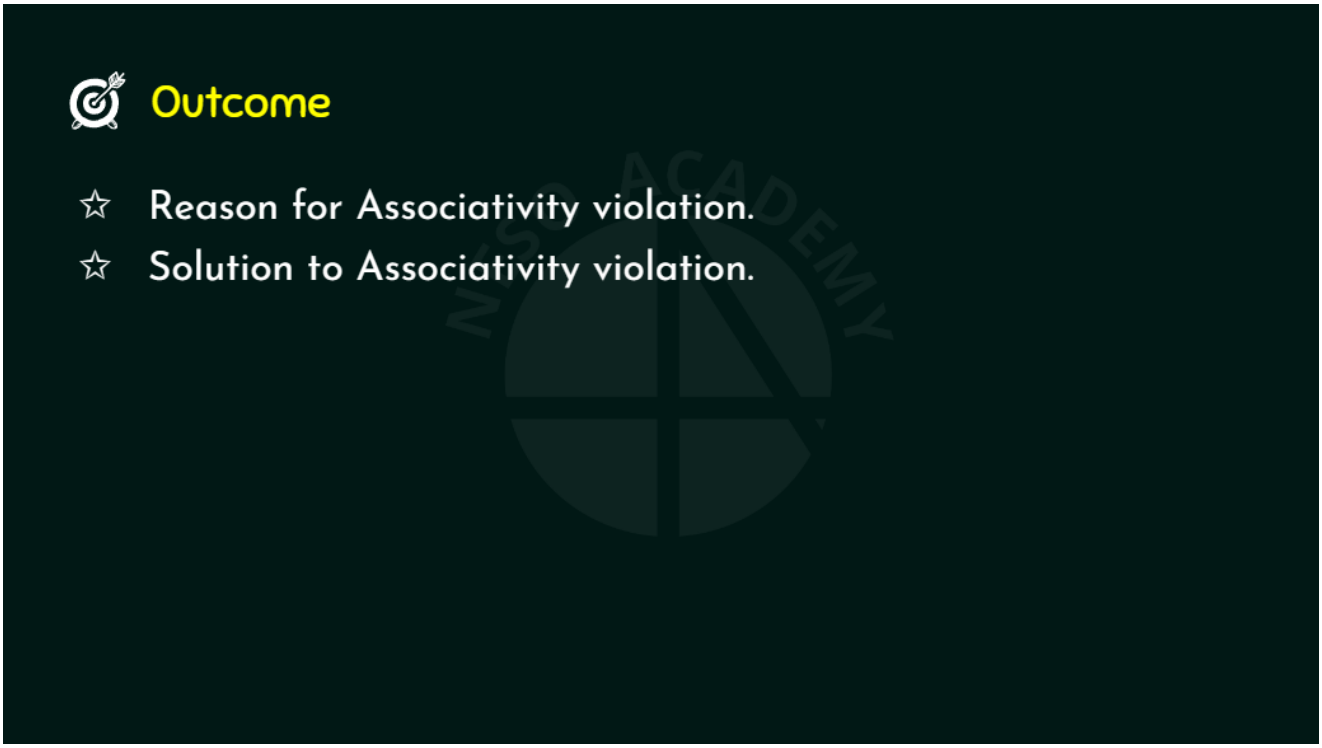


E E + Eid id id E x EE E x Eid id id E + E1.E → E + E2.E → E x E3.E → idContext Free Grammar:id + id x id

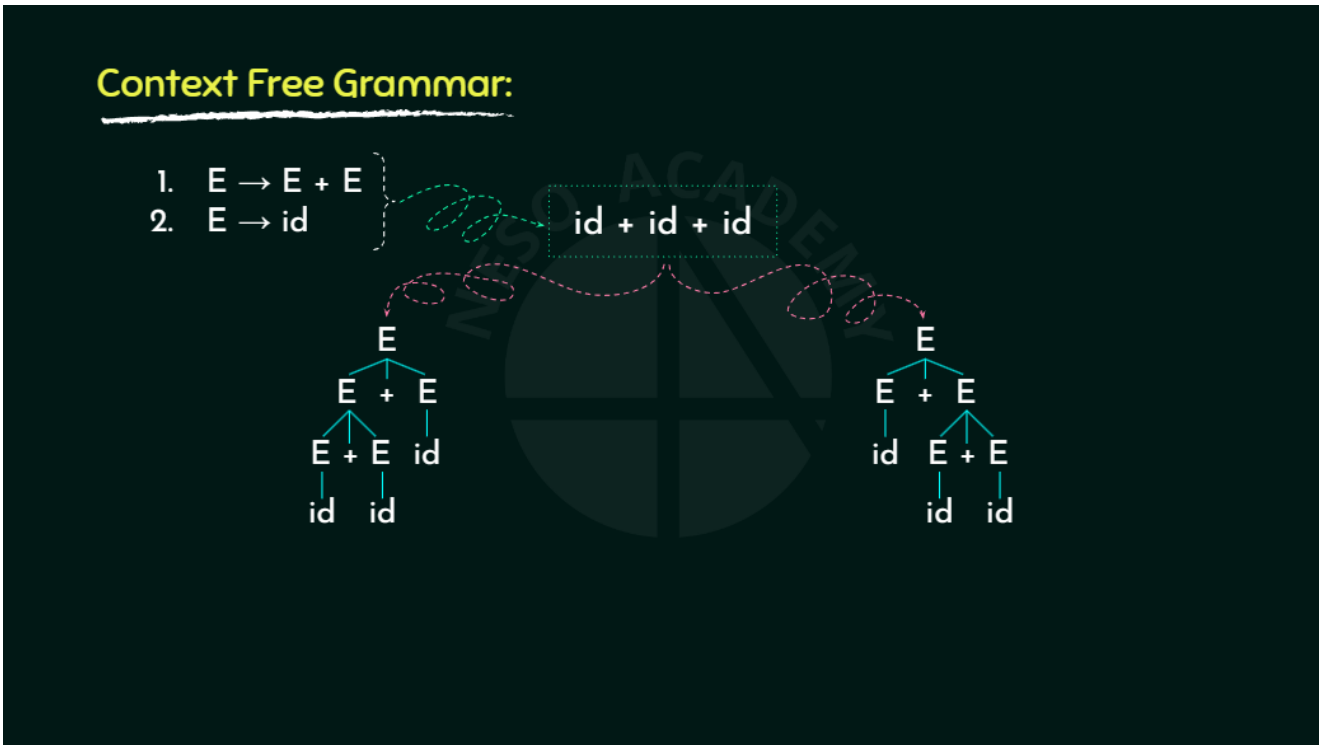


E E + Eid id id E x EE E x Eid id id E + E1.E → E + E2.E → E x E3.E → idContext Free Grammar:id + id x idPrecedence PropertyViolation

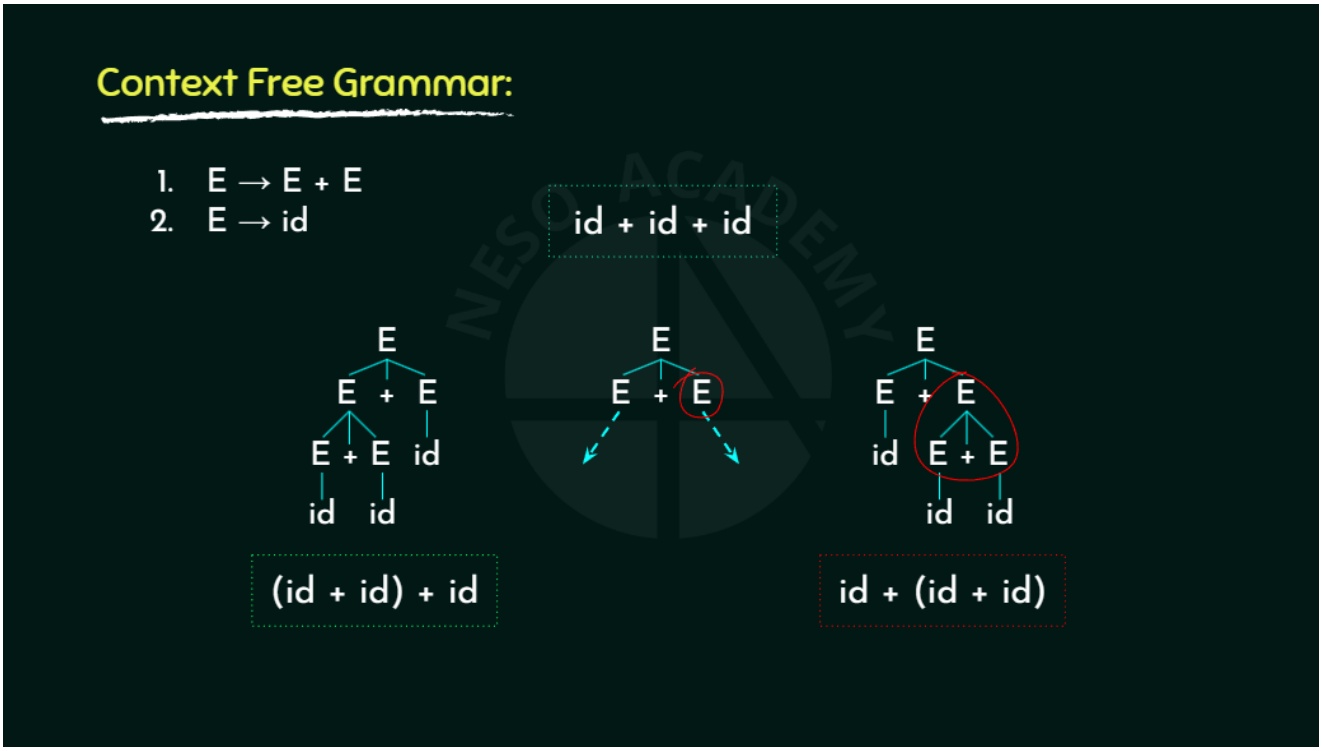
Compiler DesignAssociativity violation and solution in CFGs



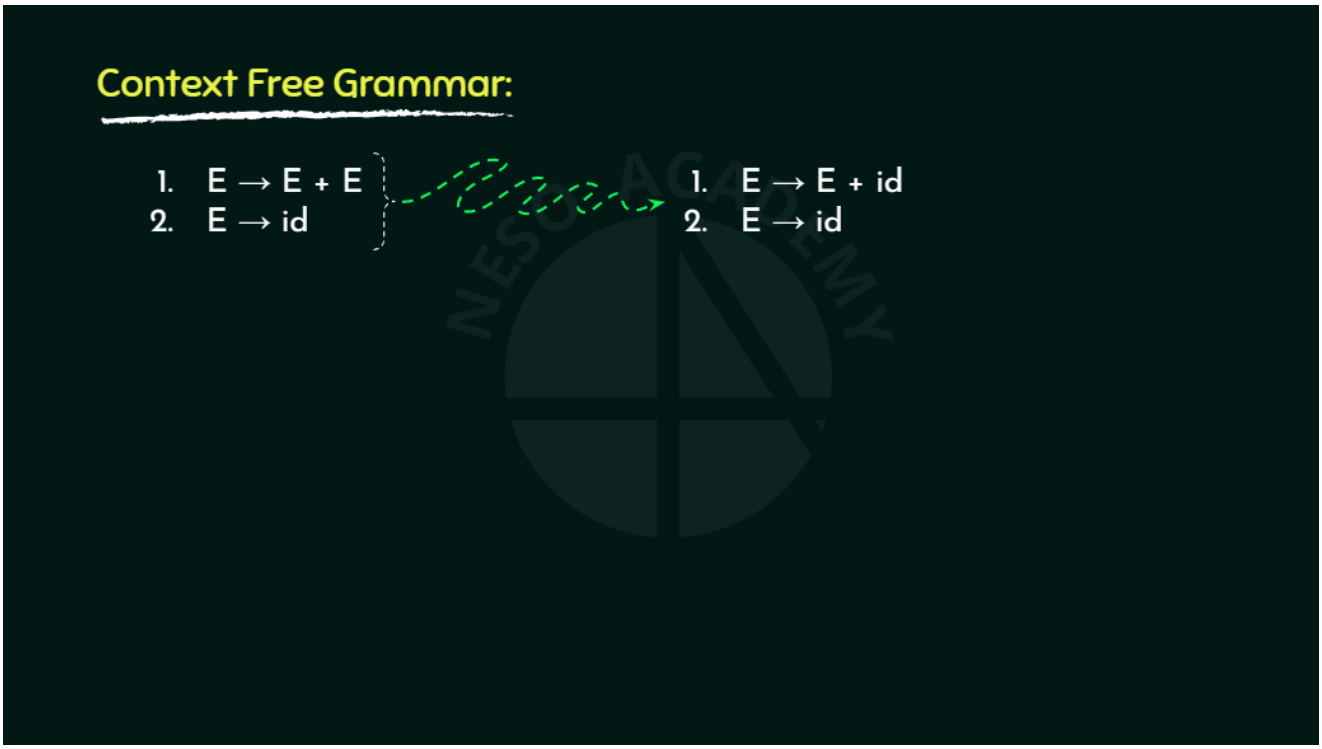
Outcome☆Reason for Associativity violation.☆Solution to Associativity violation.



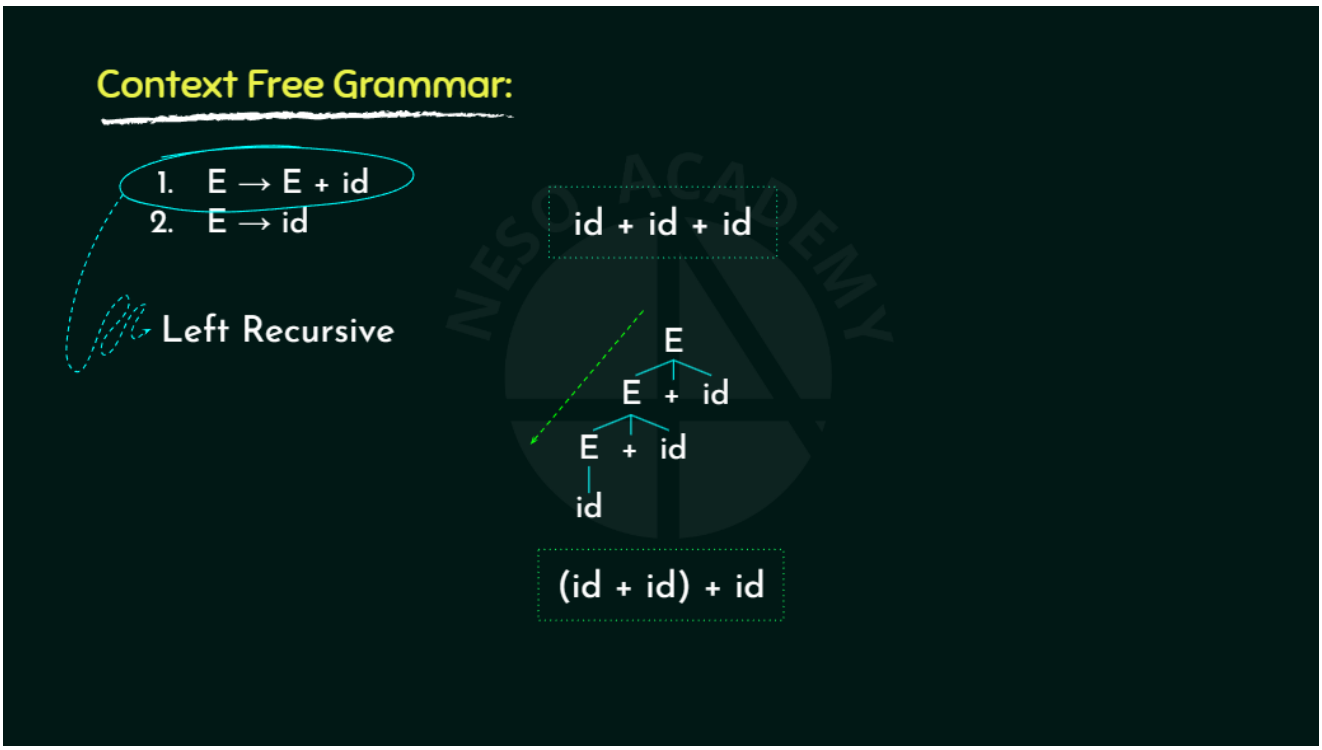
E E + Eid id id E + EE E + Eid id id E + E1.E → E + E2.E → idContext Free Grammar:id + id + id



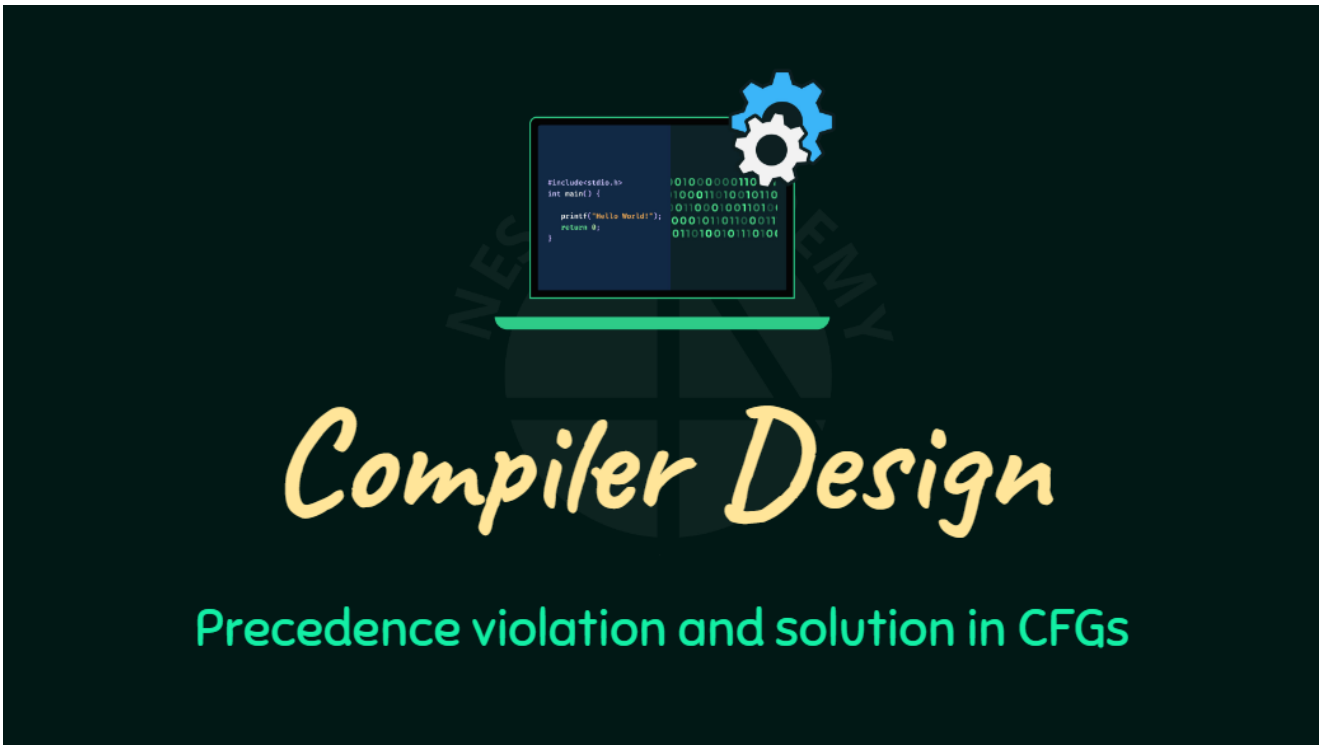
E E + Eid id id E + EE E + Eid id id E + E1.E → E + E2.E → idContext Free Grammar:id + id + idE E + E(id + id) + idid + (id + id)



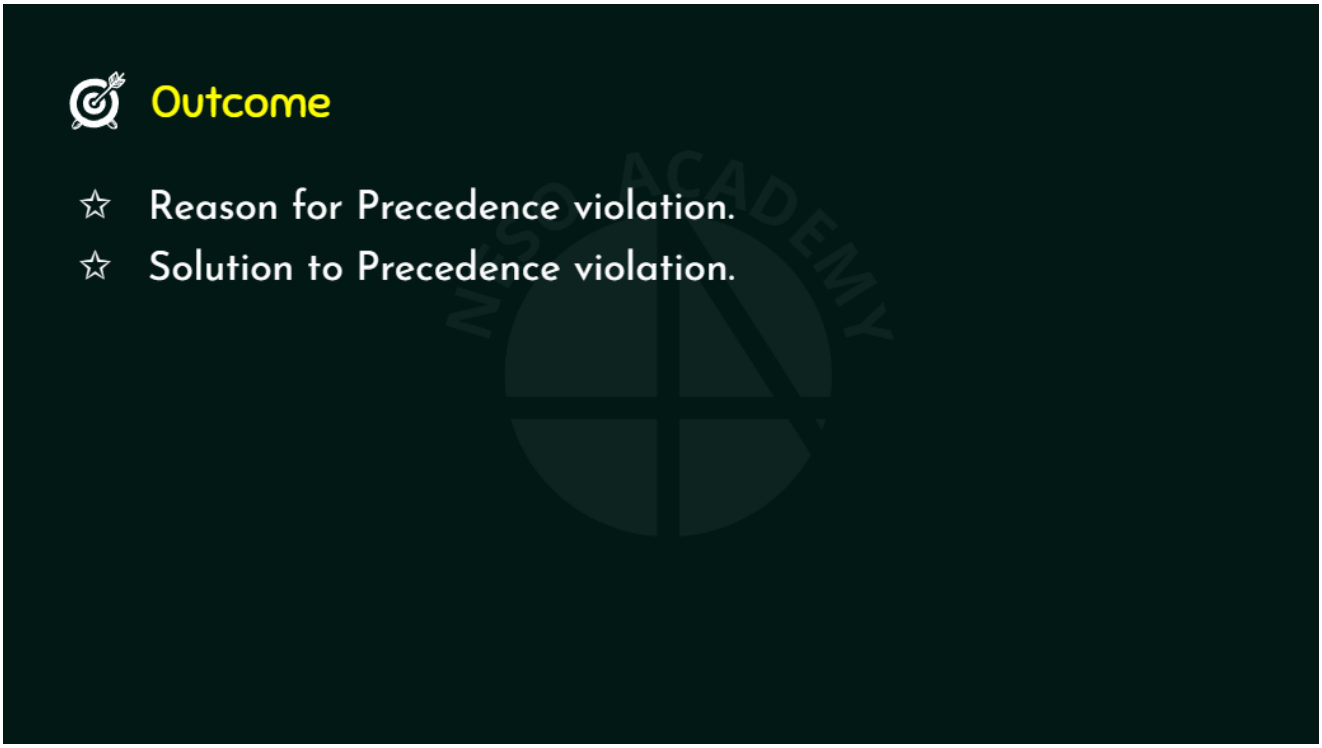
1.E → E + E2.E → idContext Free Grammar:1.E → E + id2.E → id



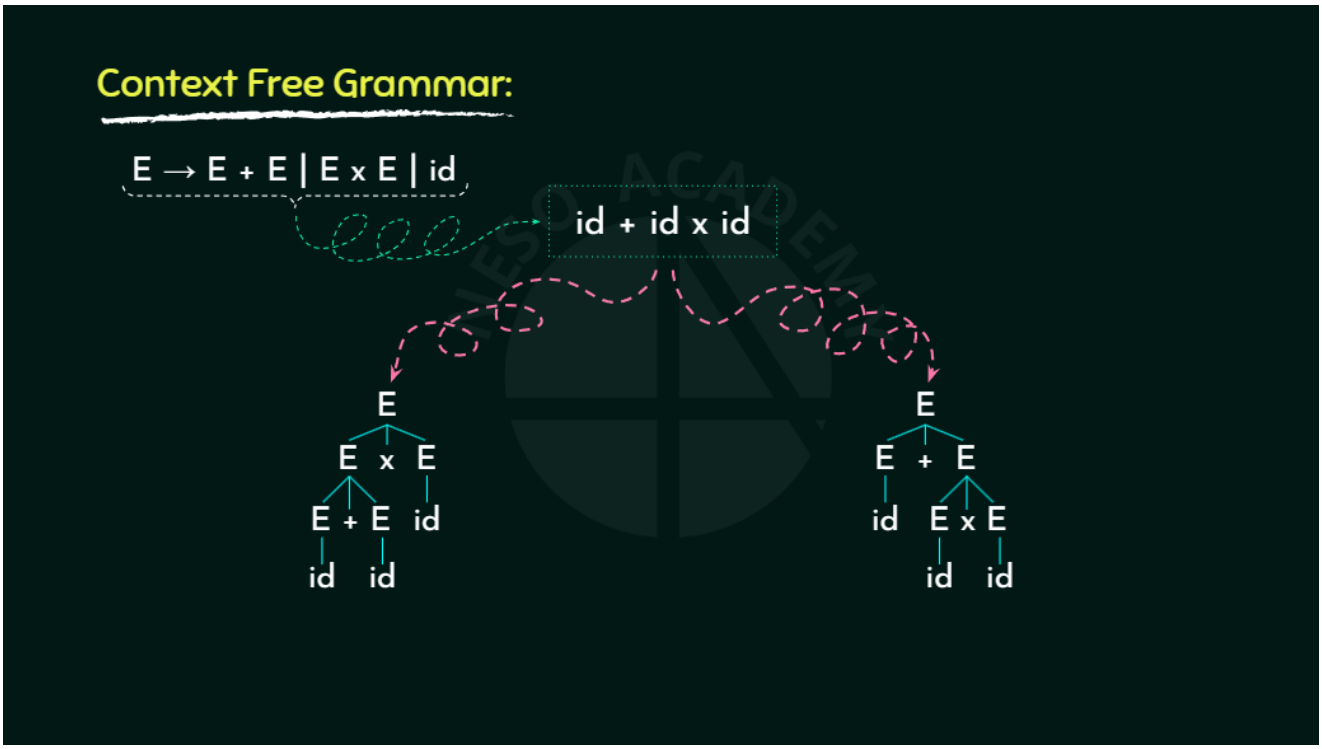
E E + idid 1.E → E + id2.E → idContext Free Grammar:id + id + id(id + id) + idE + idLeft Recursive



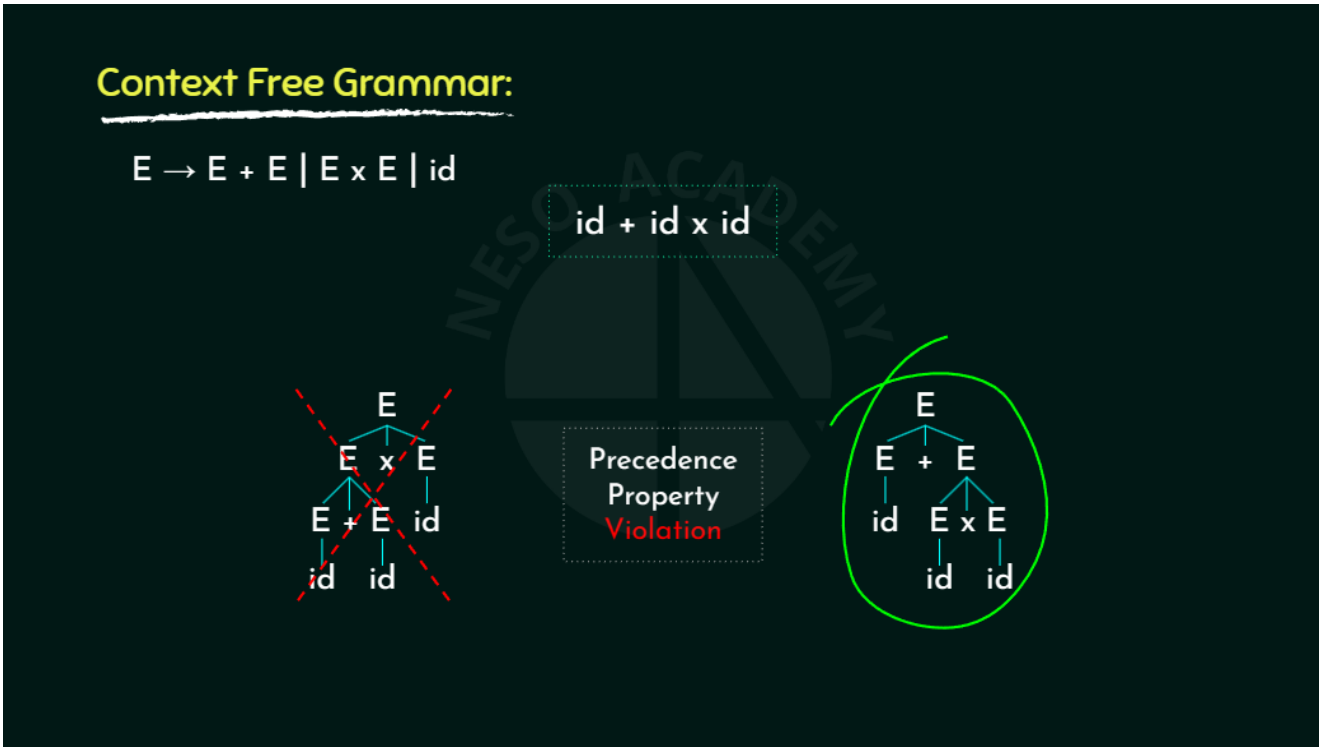
Compiler DesignPrecedence violation and solution in CFGs



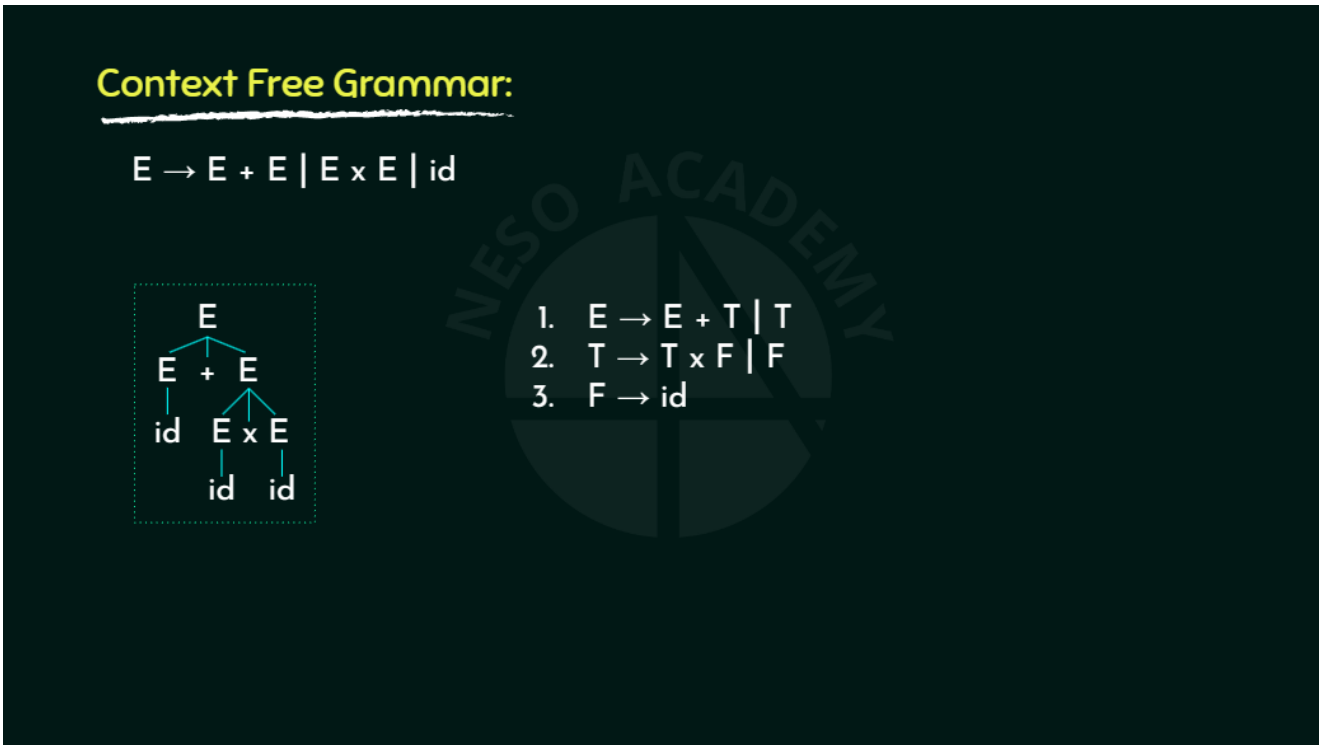
Outcome☆Reason for Precedence violation.☆Solution to Precedence violation.



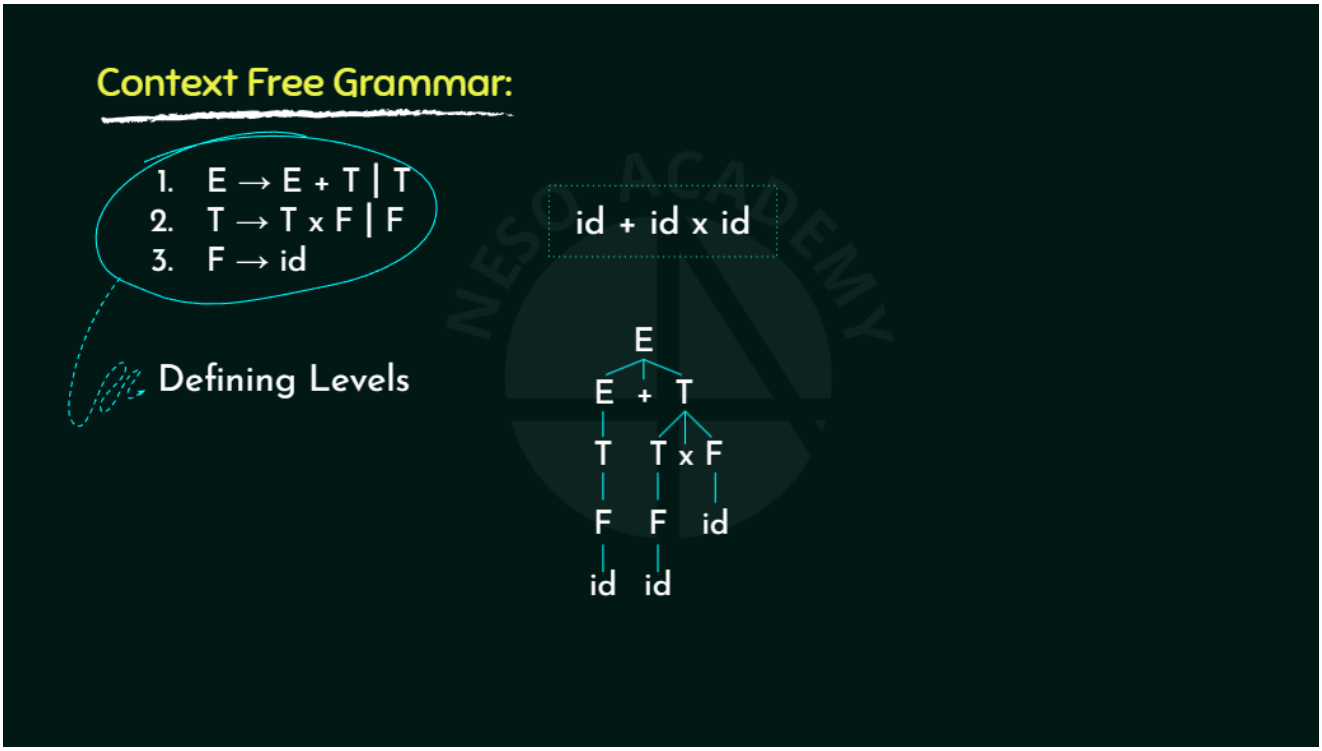
E E + Eid id id E x EE E x Eid id id E + EContext Free Grammar:id + id x idE → E + E I E x E I id||



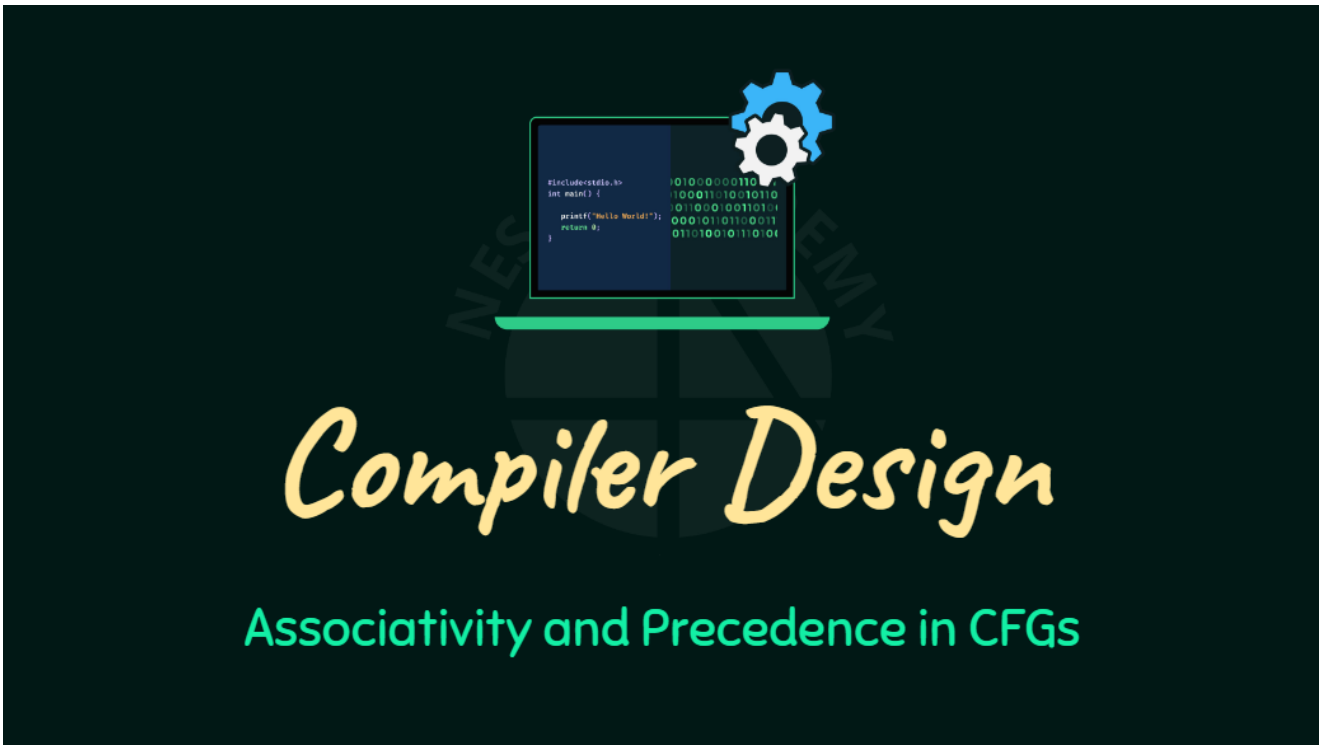
E E + Eid id id E x EE E x Eid id id E + EContext Free Grammar:id + id x idPrecedence PropertyViolation E → E + E I E x E I id||



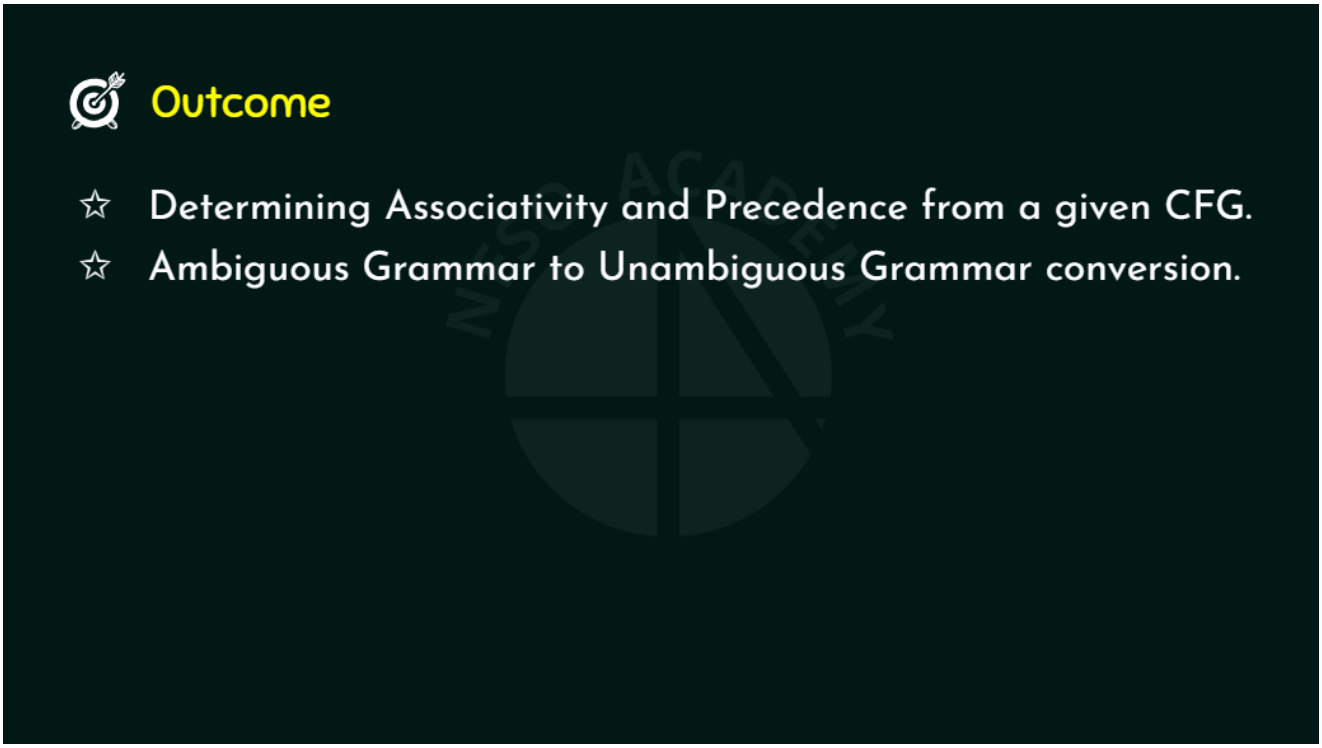
Context Free Grammar:E → E + E I E x E I id||E E + Eid id id E x E1.E → E + T2.T → T x F|T3.F → id|F



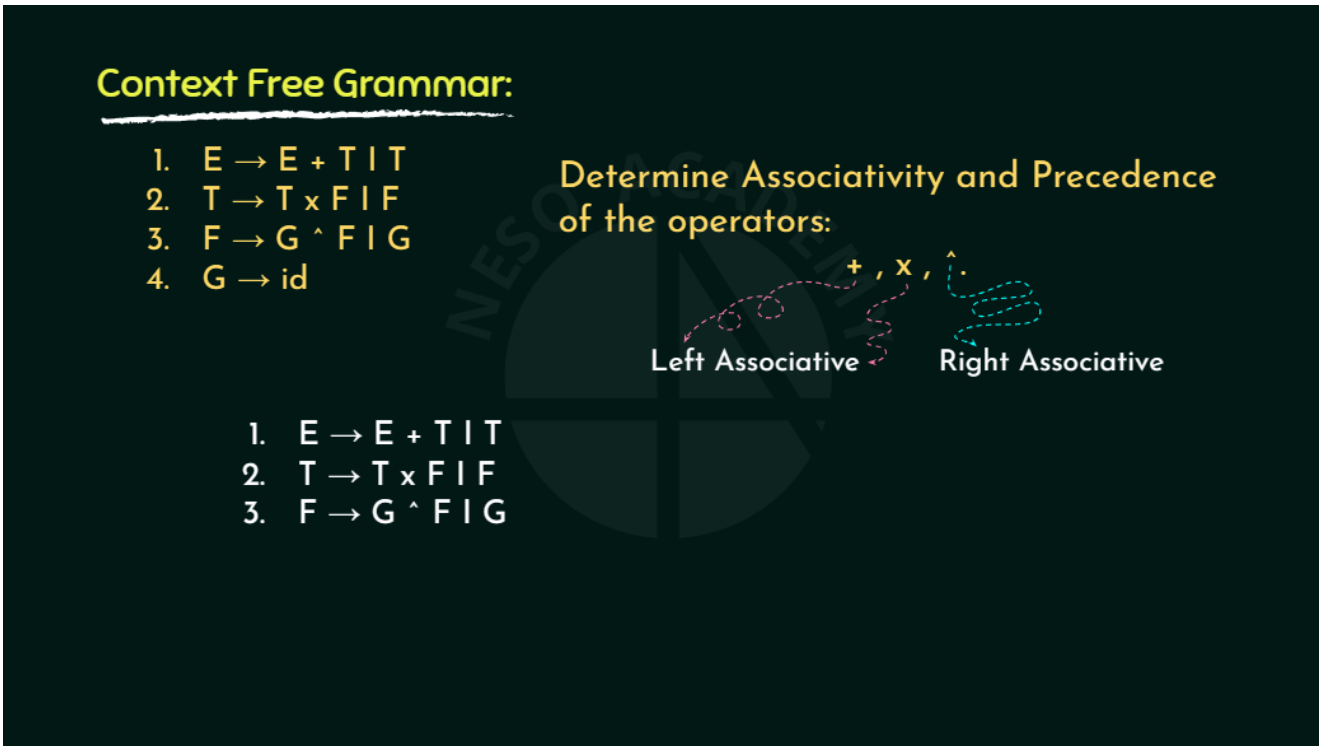
Context Free Grammar:1.E → E + T2.T → T x F|T3.F → id|Fid + id x idE E + TF id T x FT F id id Defining Levels



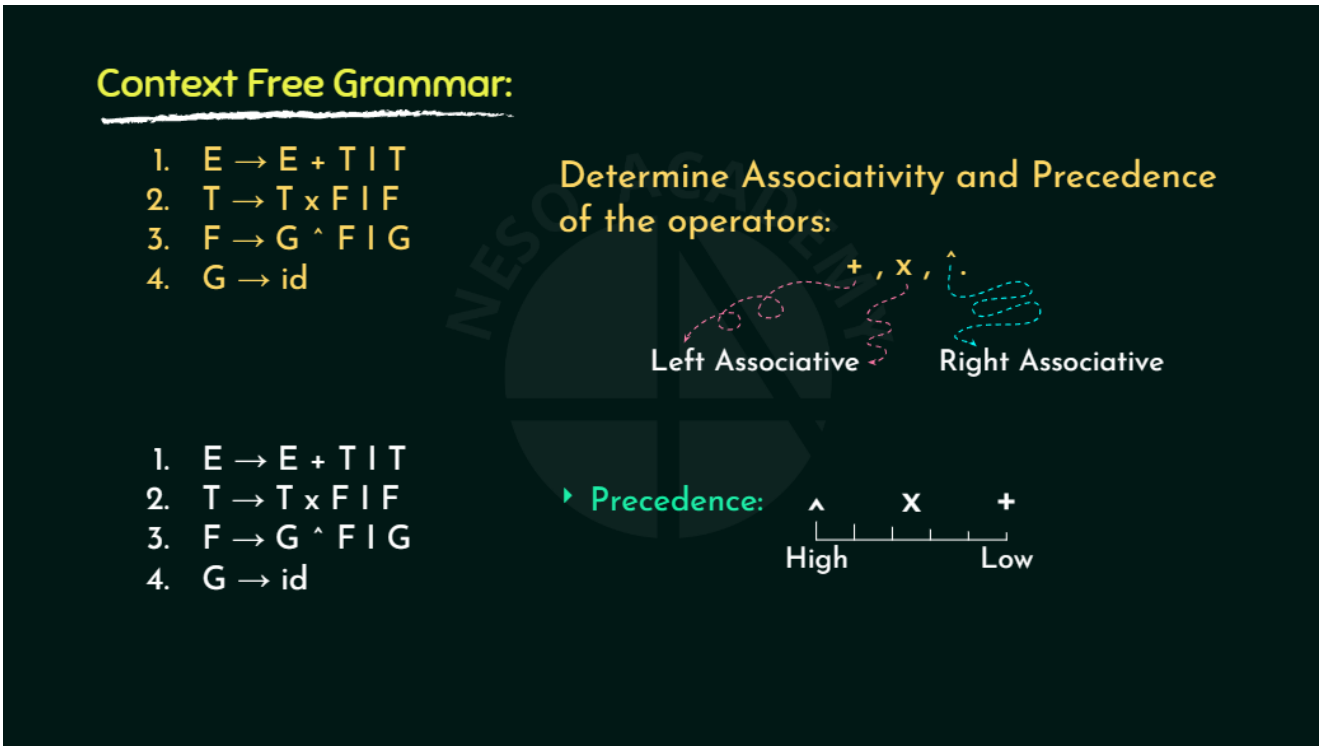
Compiler DesignAssociativity and Precedence in CFGs



Outcome☆Determining Associativity and Precedence from a given CFG.☆Ambiguous Grammar to Unambiguous Grammar conversion.



Context Free Grammar:1.E → E + T I T 2.T → T x F I F3.F → G ^ F I G4.G → idDetermine Associativity and Precedence of the operators: + , x , ^.1.E → E + T I T Left AssociativeRight Associative3.F → G ^ F I G2.T → T x F I F



1.E → E + T I T 2.T → T x F I F3.F → G ^ F I G4.G → idContext Free Grammar:1.E → E + T I T 2.T → T x F I F3.F → G ^ F I G4.G → idDetermine Associativity and Precedence of the operators: + , x , ^.Left AssociativeRight Associative‣Precedence:^x+HighLow

Context Free Grammar:

Consider the following ambiguous CFG on boolean expressions:

bExp → bExp AND bExp I bExp OR bExp I NOT bExp I True I False

The precedence of the boolean operators are NOT, AND, OR (high to low). AND, OR have left to right associativity.

1. bExp → bExp OR bExp1 I bExp1
2. bExp1 → bExp1 AND bExp2 I bExp2
3. bExp2 → NOT bExp2
4. bExp2 → True
5. bExp2 → False

1.bExp → bExp OR bExp1Context Free Grammar:Consider the following ambiguous CFG on boolean expressions: bExp → bExp AND bExp I bExp OR bExp I NOT bExp I True I FalseThe precedence of the boolean operators are NOT, AND, OR (high to low).AND, OR have left to right associativity. 2.bExp1 → bExp1 AND bExp23.bExp2 → NOT bExp24.bExp2 → True5.bExp2 → FalseI bExp1I bExp2

Context Free Grammar:

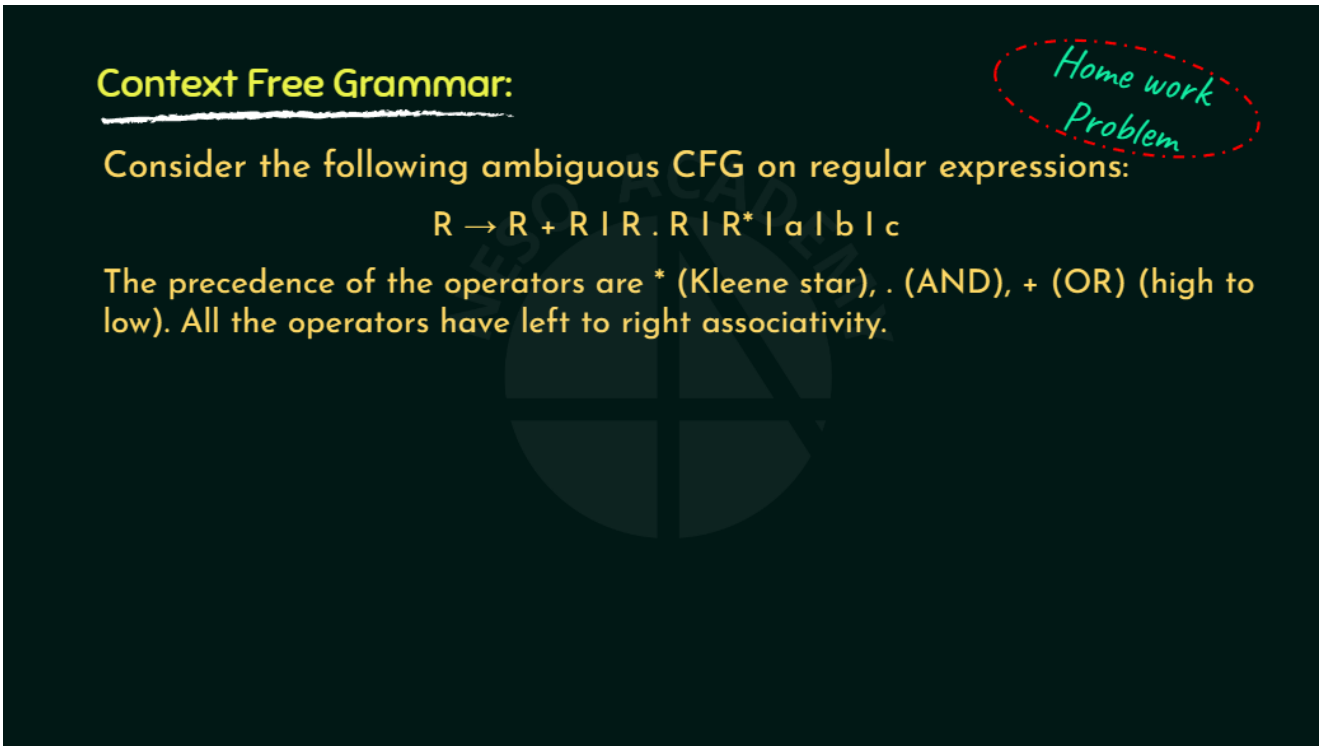
Consider the following ambiguous CFG on boolean expressions:

bExp → bExp AND bExp I bExp OR bExp I NOT bExp I True I False

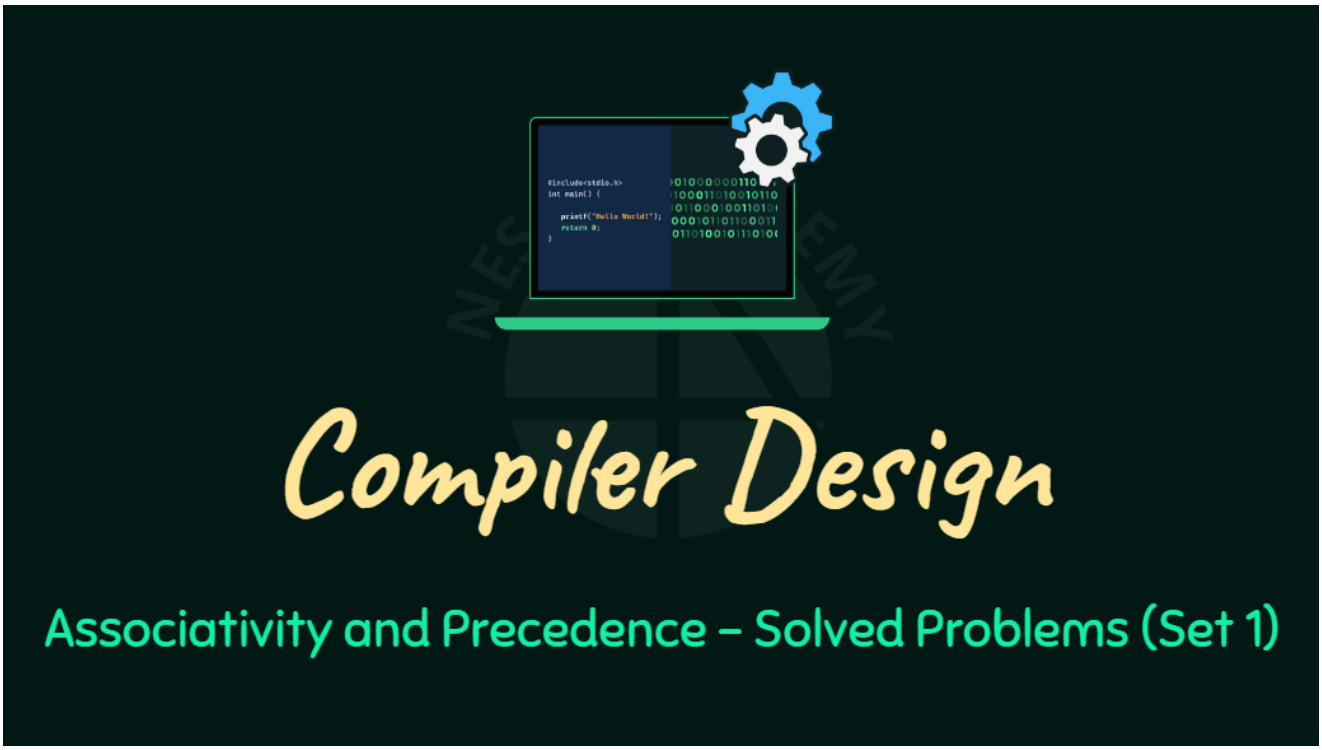
The precedence of the boolean operators are NOT, AND, OR (high to low). AND, OR have left to right associativity.

1. bExp → bExp OR bExp1 I bExp1
2. bExp1 → bExp1 AND bExp2 I bExp2
3. bExp2 → NOT bExp2 I True I False

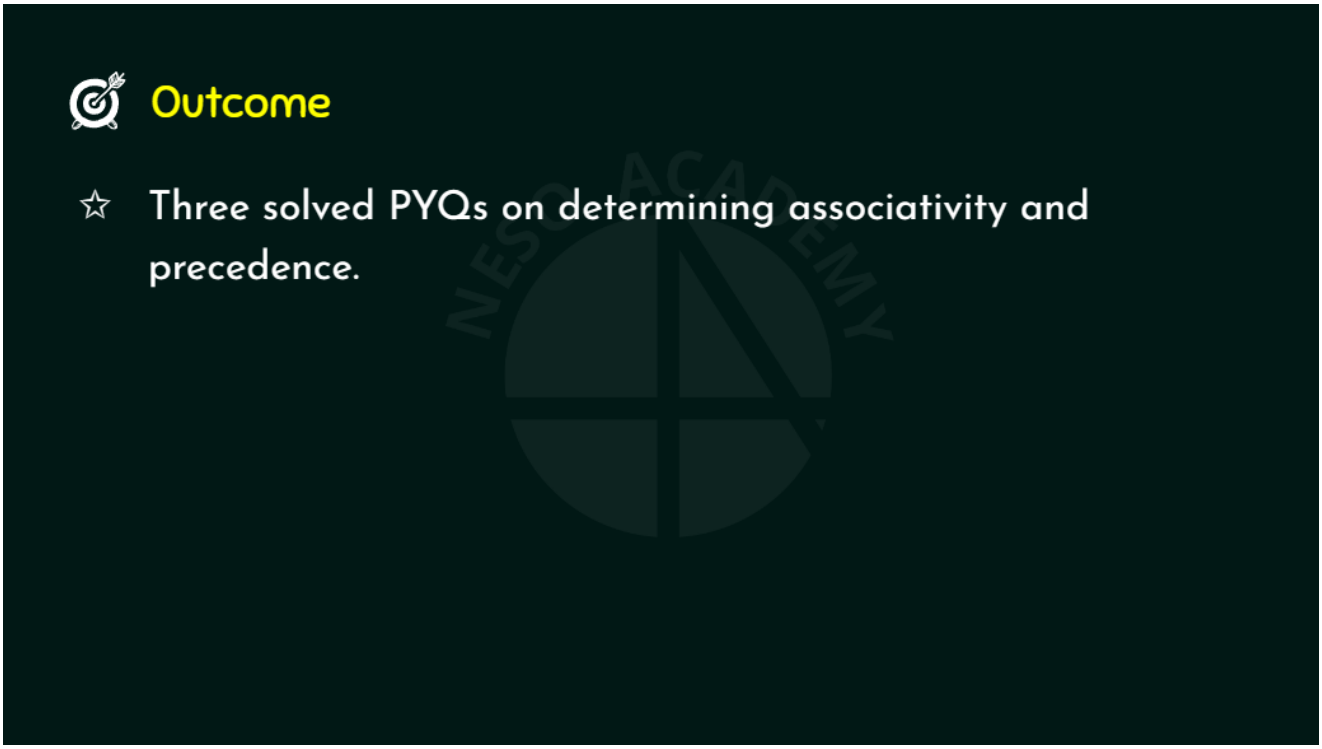
Context Free Grammar:Consider the following ambiguous CFG on boolean expressions: bExp → bExp AND bExp I bExp OR bExp I NOT bExp I True I FalseThe precedence of the boolean operators are NOT, AND, OR (high to low).AND, OR have left to right associativity. 1.bExp → bExp OR bExp12.bExp1 → bExp1 AND bExp23.bExp2 → NOT bExp2 I True I FalseI bExp1I bExp2



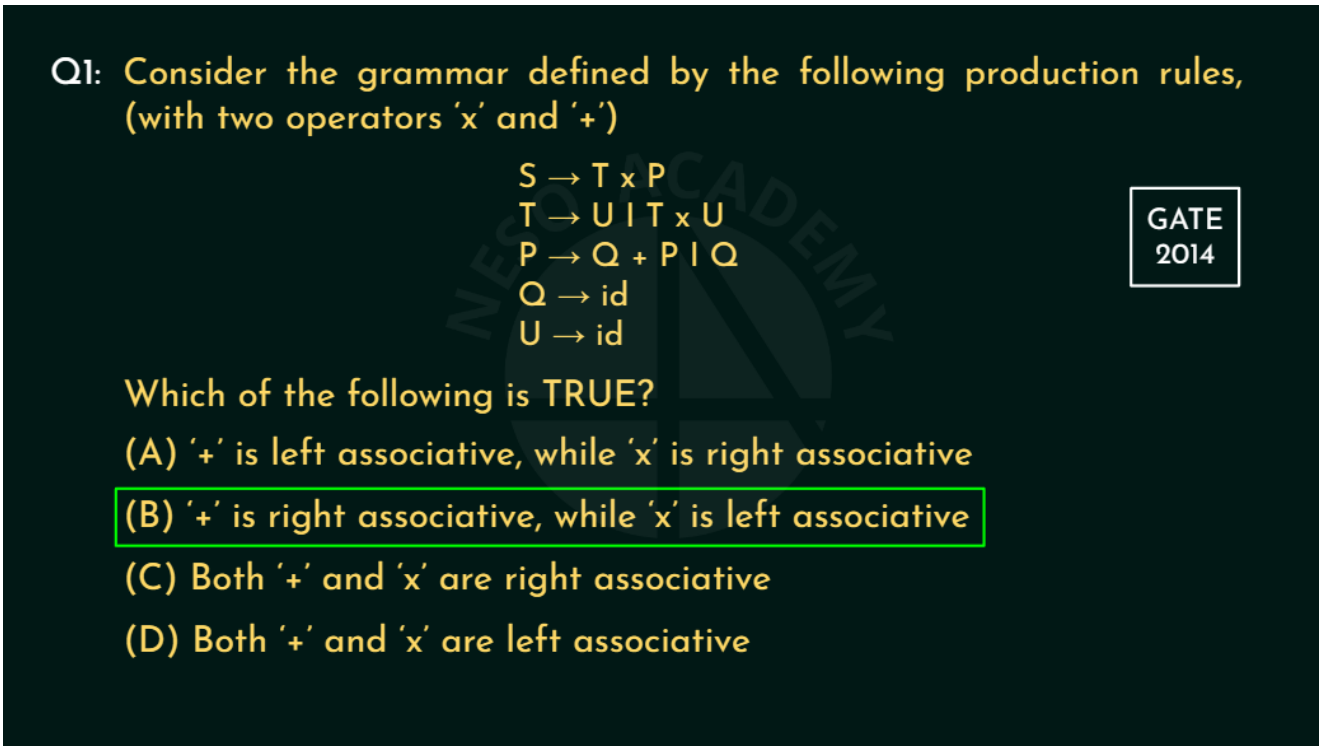
Context Free Grammar:Consider the following ambiguous CFG on regular expressions: R → R + R I R . R I R* I a I b I cThe precedence of the operators are * (Kleene star), . (AND), + (OR) (high to low). All the operators have left to right associativity. Home workProblem



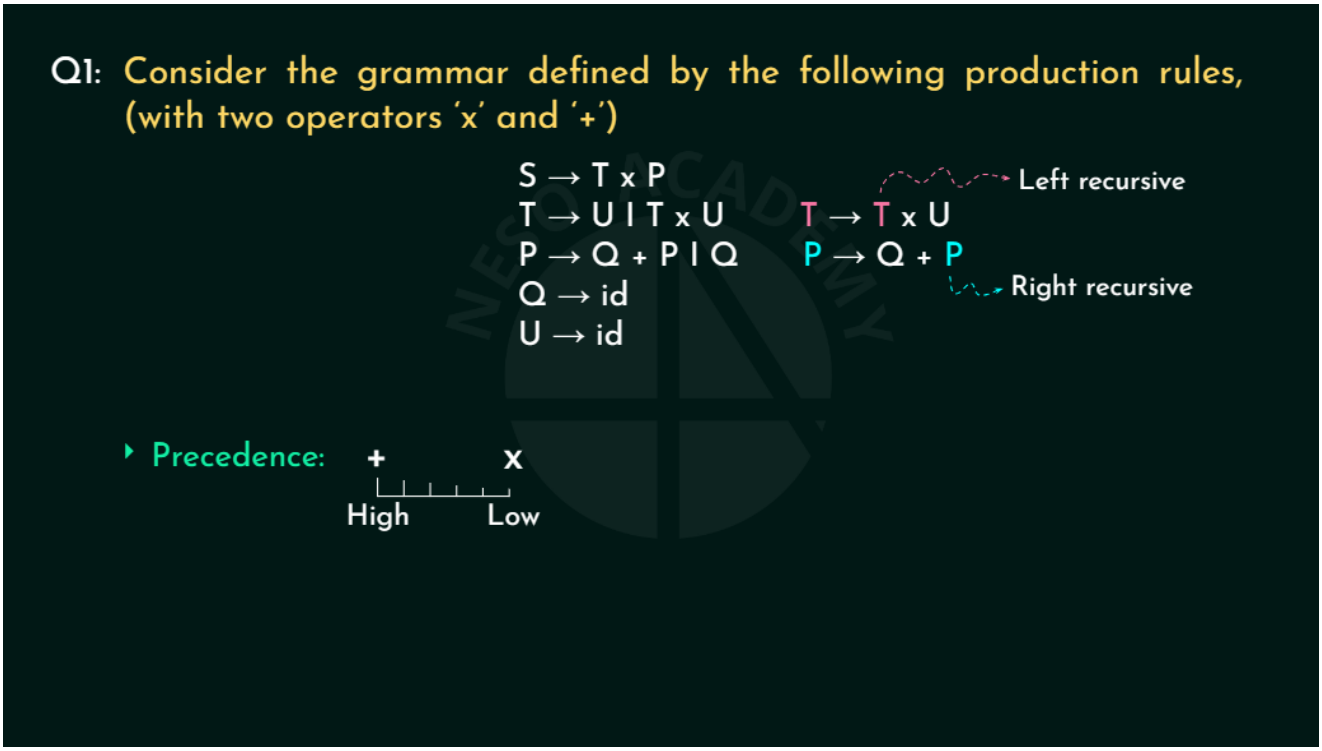
Compiler DesignAssociativity and Precedence - Solved Problems (Set 1)



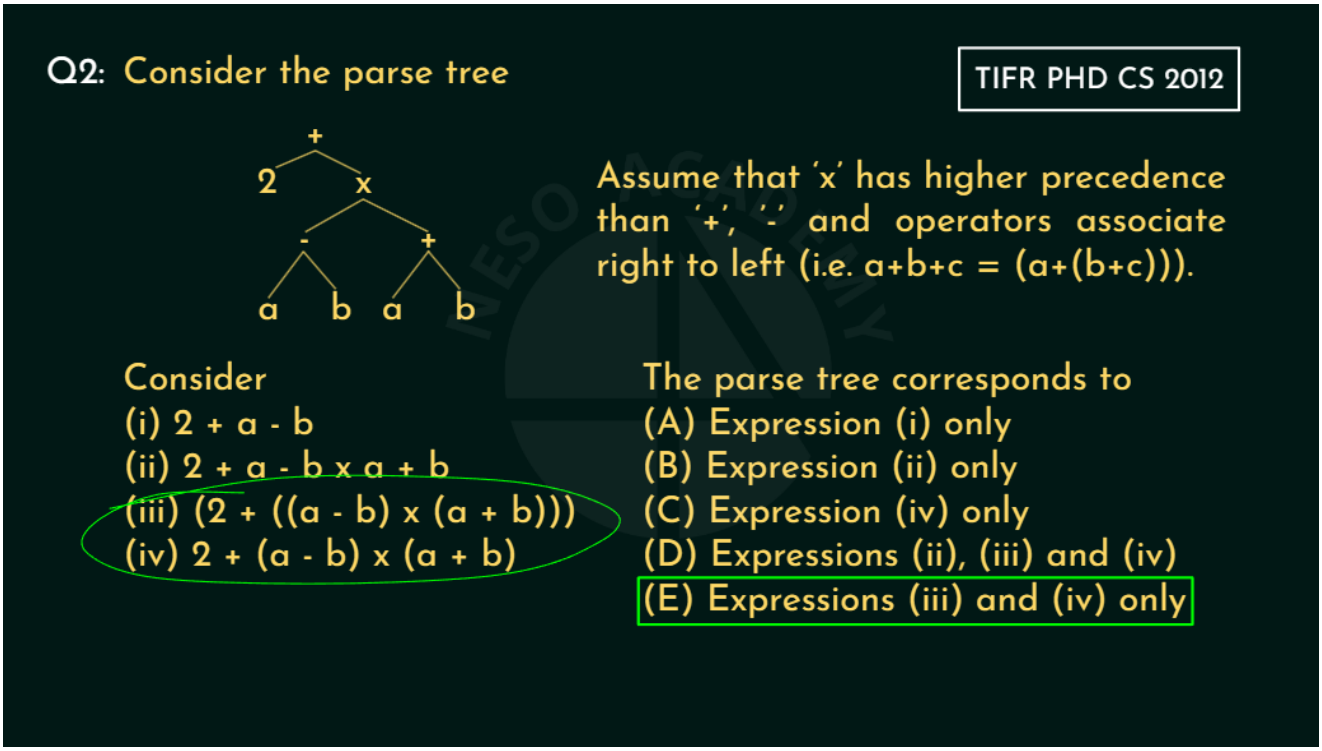
Outcome☆Three solved PYQs on determining associativity and precedence.



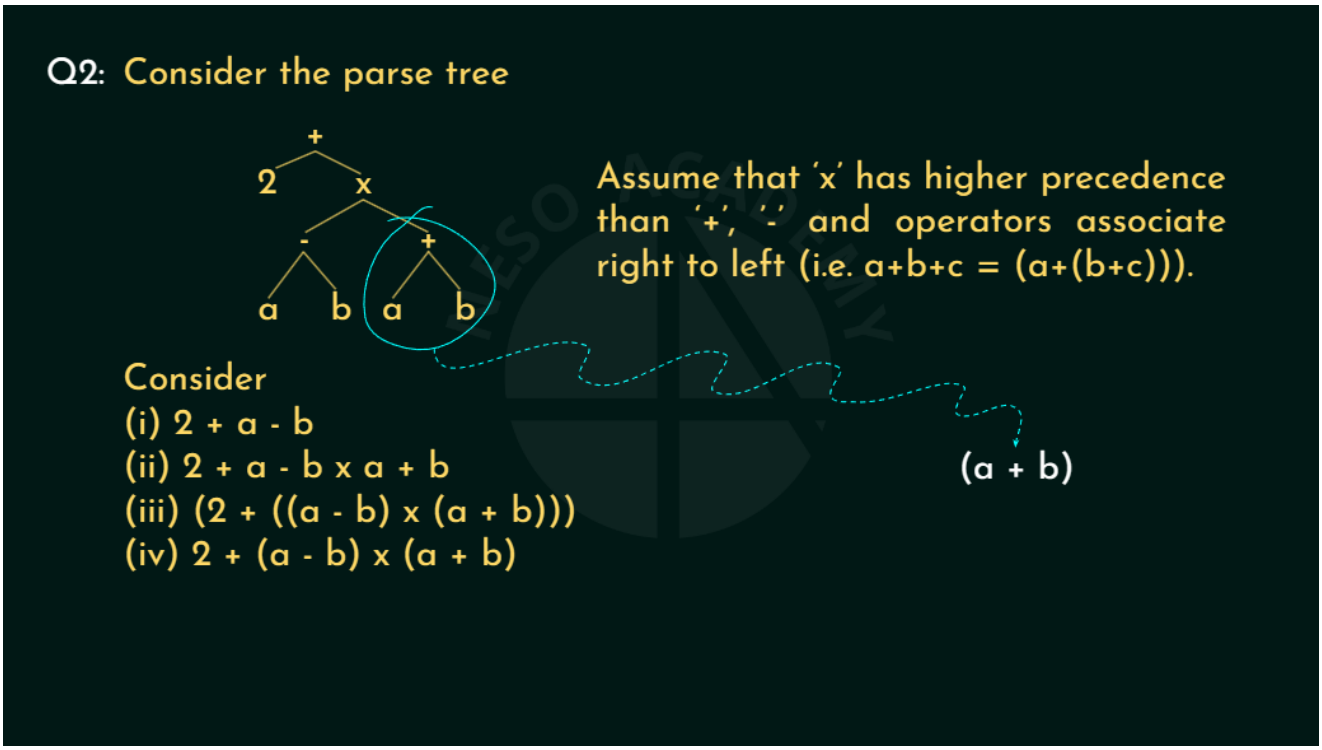
Q1:Consider the grammar defined by the following production rules, (with two operators 'x' and '+')S → T x PT → U | T x UP → Q + P | QQ → idU → idWhich of the following is TRUE? GATE 2014(A) '+' is left associative, while 'x' is right associative(B) '+' is right associative, while 'x' is left associative(C) Both '+' and 'x' are right associative(D) Both '+' and 'x' are left associative



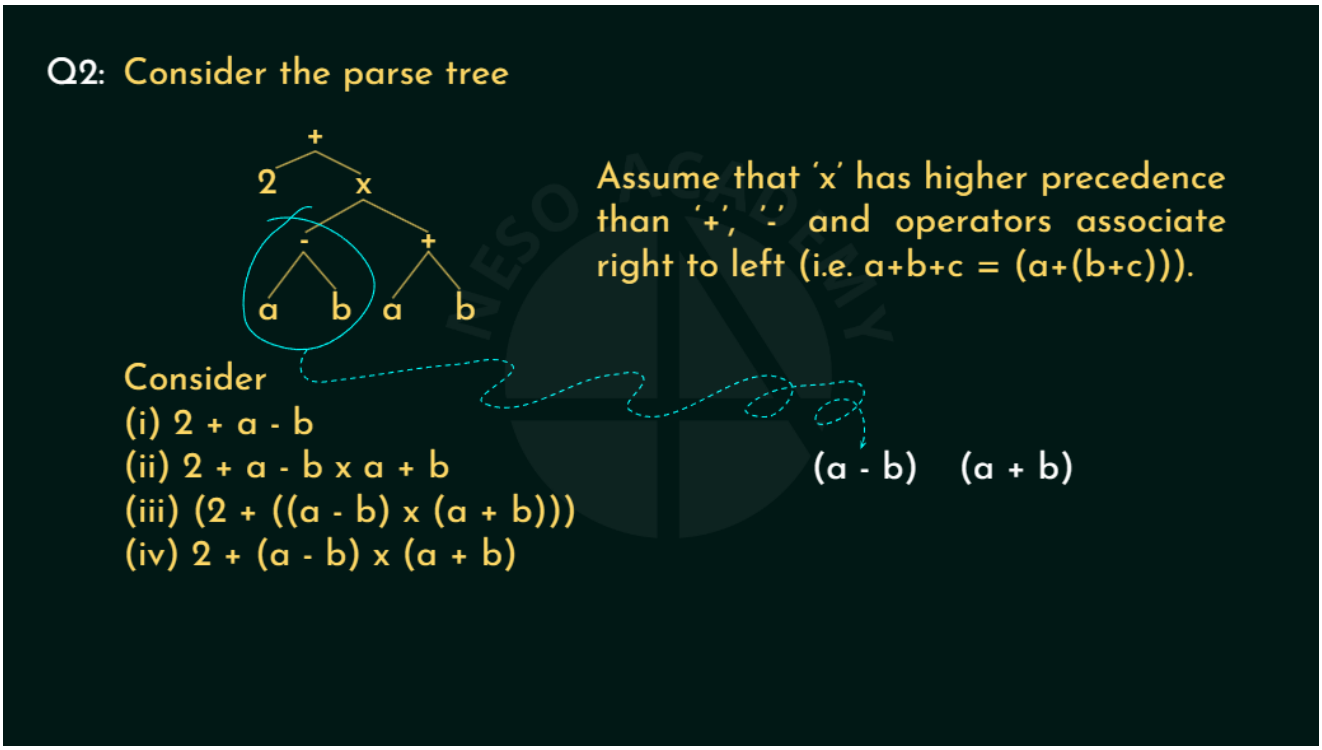
Q1:Consider the grammar defined by the following production rules, (with two operators ‘x’ and ‘+’)S → T x PT → U I T x UP → Q + P I QQ → idU → id‣Precedence:+xHighLowT → T x ULeft recursiveP → Q + PRight recursive



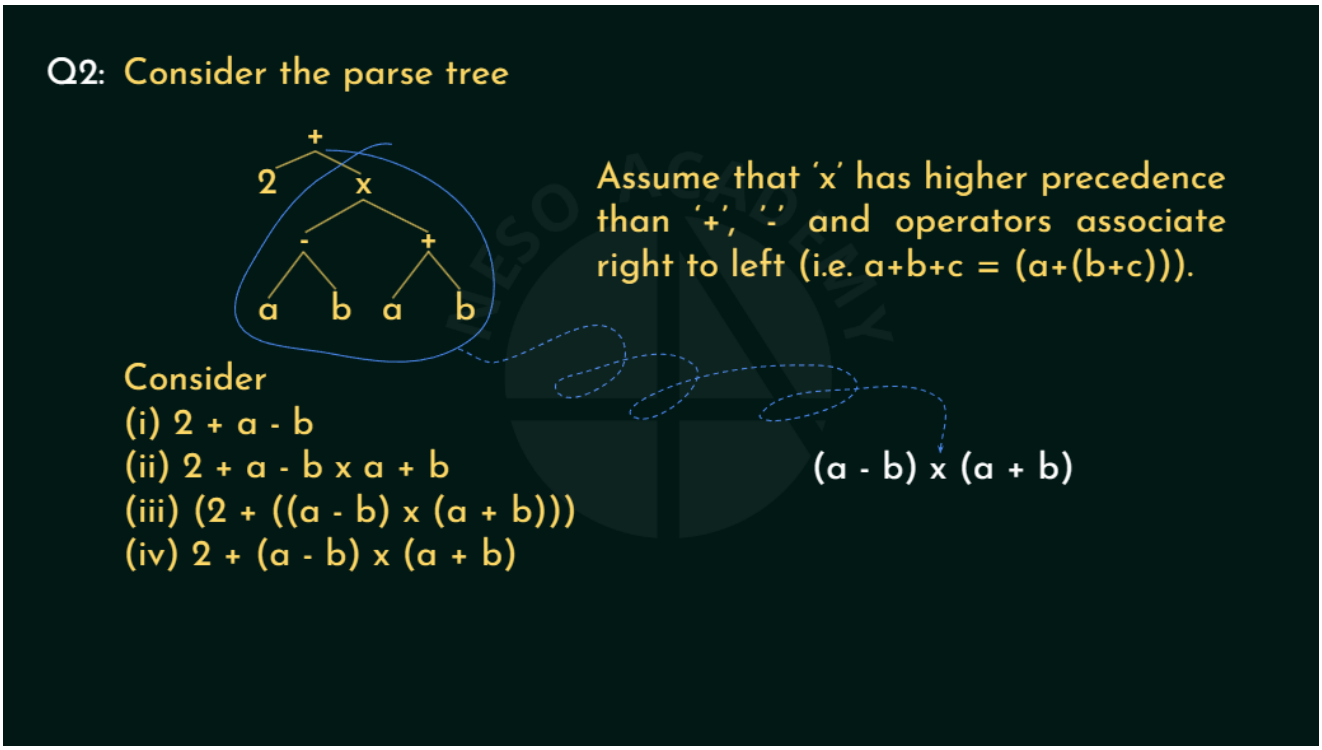
Q2:Consider the parse treeTIFR PHD CS 2012+-+x2ababAssume that ‘x’ has higher precedence than ‘+’, ’-’ and operators associate right to left (i.e. a+b+c = (a+(b+c))).Consider(i) 2 + a - b(ii) 2 + a - b x a + b(iii) (2 + ((a - b) x (a + b)))(iv) 2 + (a - b) x (a + b)The parse tree corresponds to(A) Expression (i) only(B) Expression (ii) only(C) Expression (iv) only(D) Expressions (ii), (iii) and (iv)(E) Expressions (iii) and (iv) only



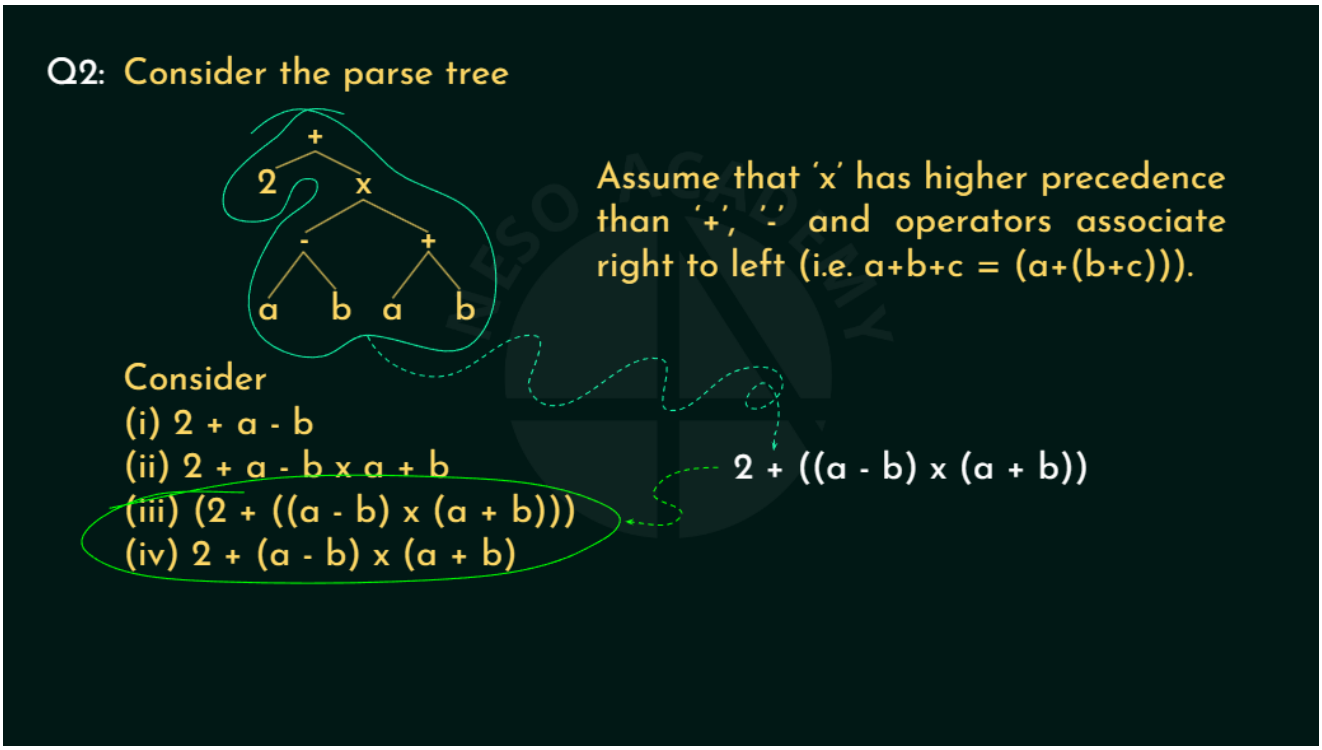
Q2:Consider the parse tree+-+x2ababAssume that 'x' has higher precedence than '+', '-' and operators associate right to left (i.e. a+b+c = (a+(b+c))). Consider(i) 2 + a - b(ii) 2 + a - b x a + b(iii) (2 + ((a - b) x (a + b)))(iv) 2 + (a - b) x (a + b)(a + b)



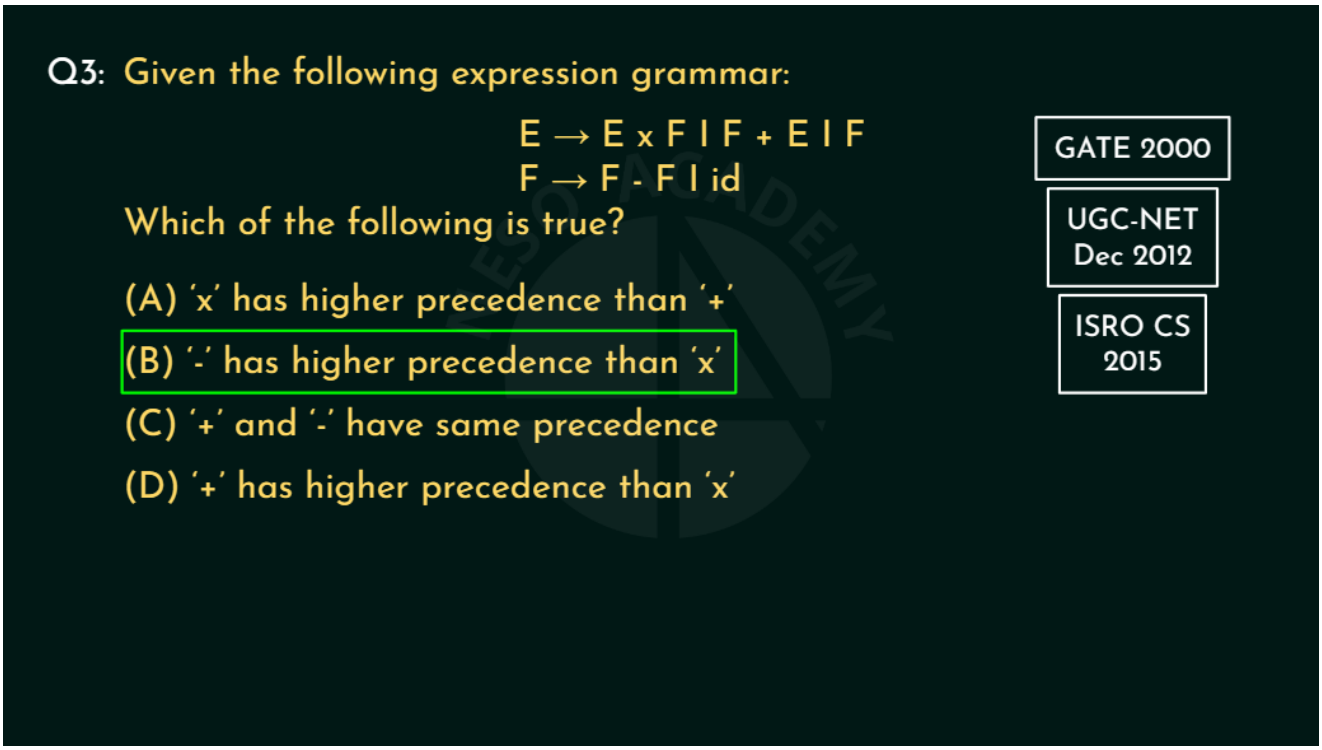
Q2:Consider the parse tree+-+x2ababAssume that 'x' has higher precedence than '+', '-' and operators associate right to left (i.e. a+b+c = (a+(b+c))). Consider(i) 2 + a - b(ii) 2 + a - b x a + b(iii) (2 + ((a - b) x (a + b)))(iv) 2 + (a - b) x (a + b)(a - b)(a + b)



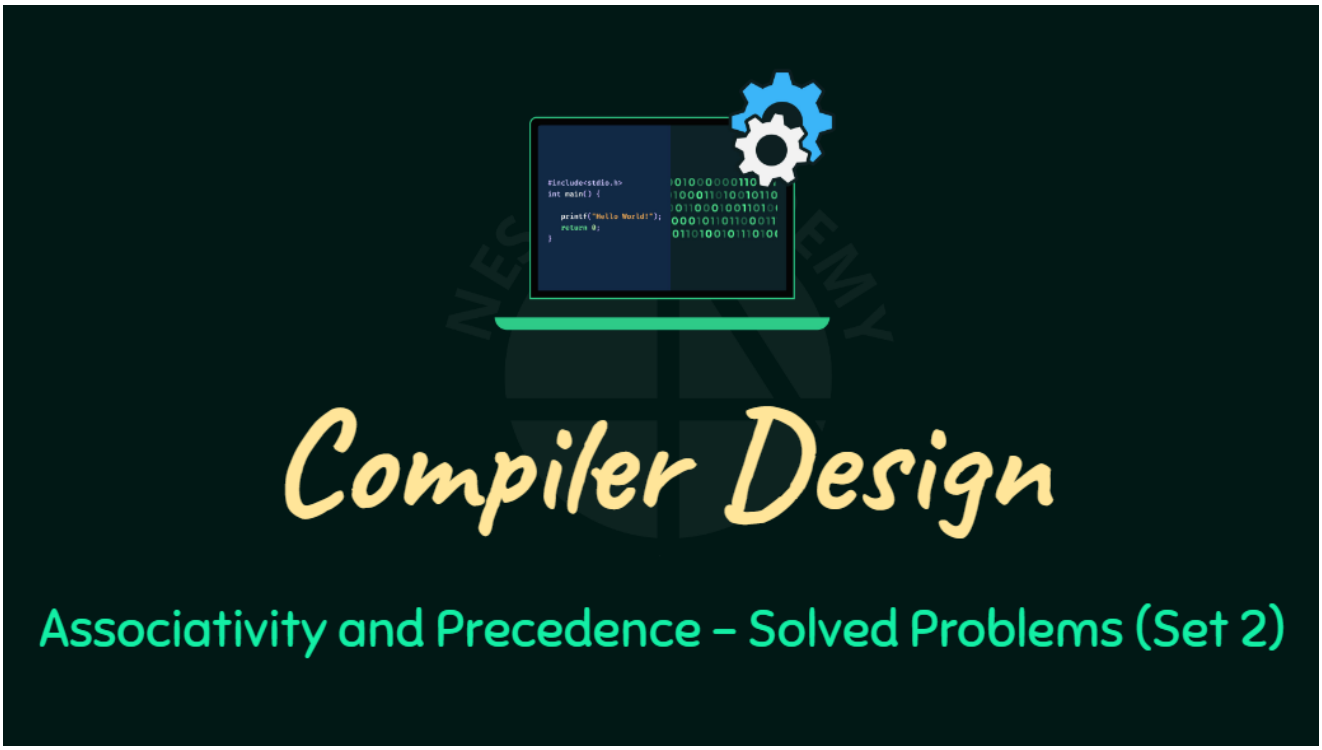
Q2:Consider the parse tree+-+x2ababAssume that 'x' has higher precedence than '+', '-' and operators associate right to left (i.e. a+b+c = (a+(b+c))). Consider(i) 2 + a - b(ii) 2 + a - b x a + b(iii) (2 + ((a - b) x (a + b)))(iv) 2 + (a - b) x (a + b)(a - b)(a + b)x



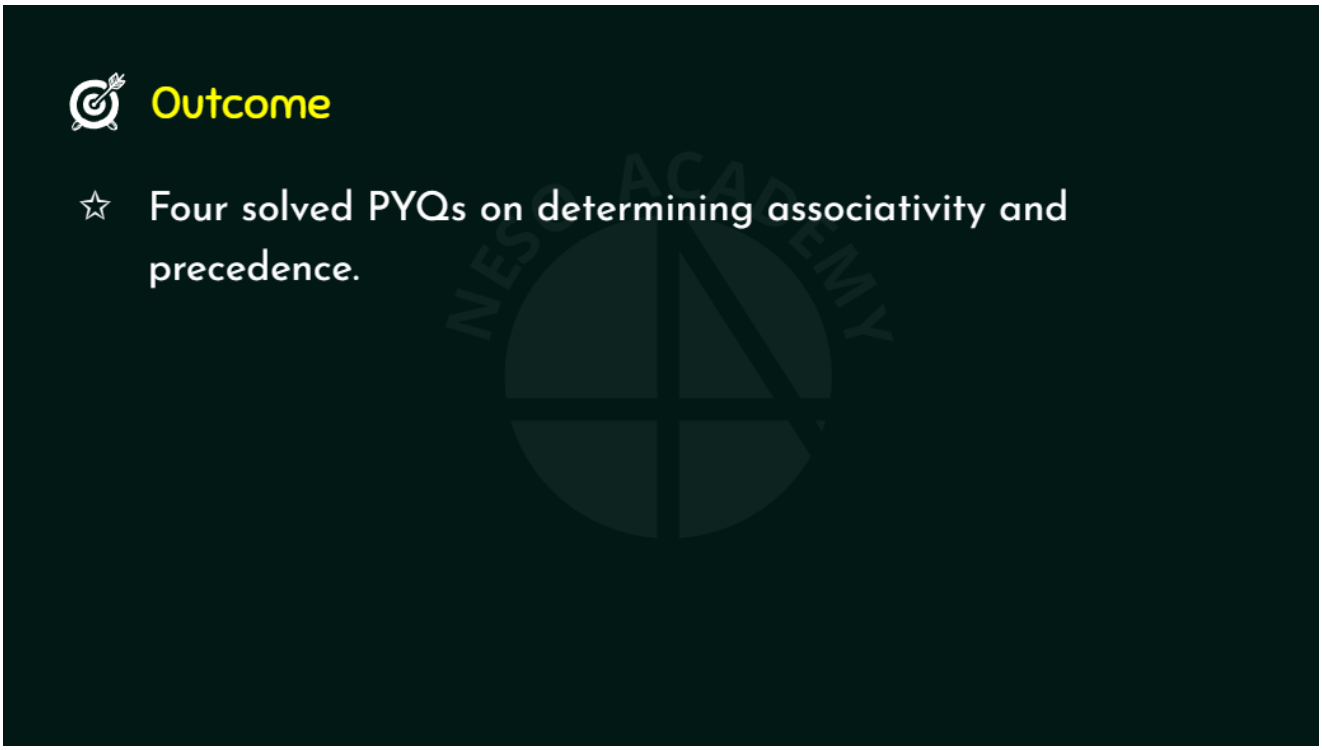
Q2:Consider the parse tree+-+x2ababAssume that 'x' has higher precedence than '+', '-' and operators associate right to left (i.e. a+b+c = (a+(b+c))). Consider(i) 2 + a - b(ii) 2 + a - b x a + b(iii) (2 + ((a - b) x (a + b)))(iv) 2 + (a - b) x (a + b)((a - b) x (a + b))2 +



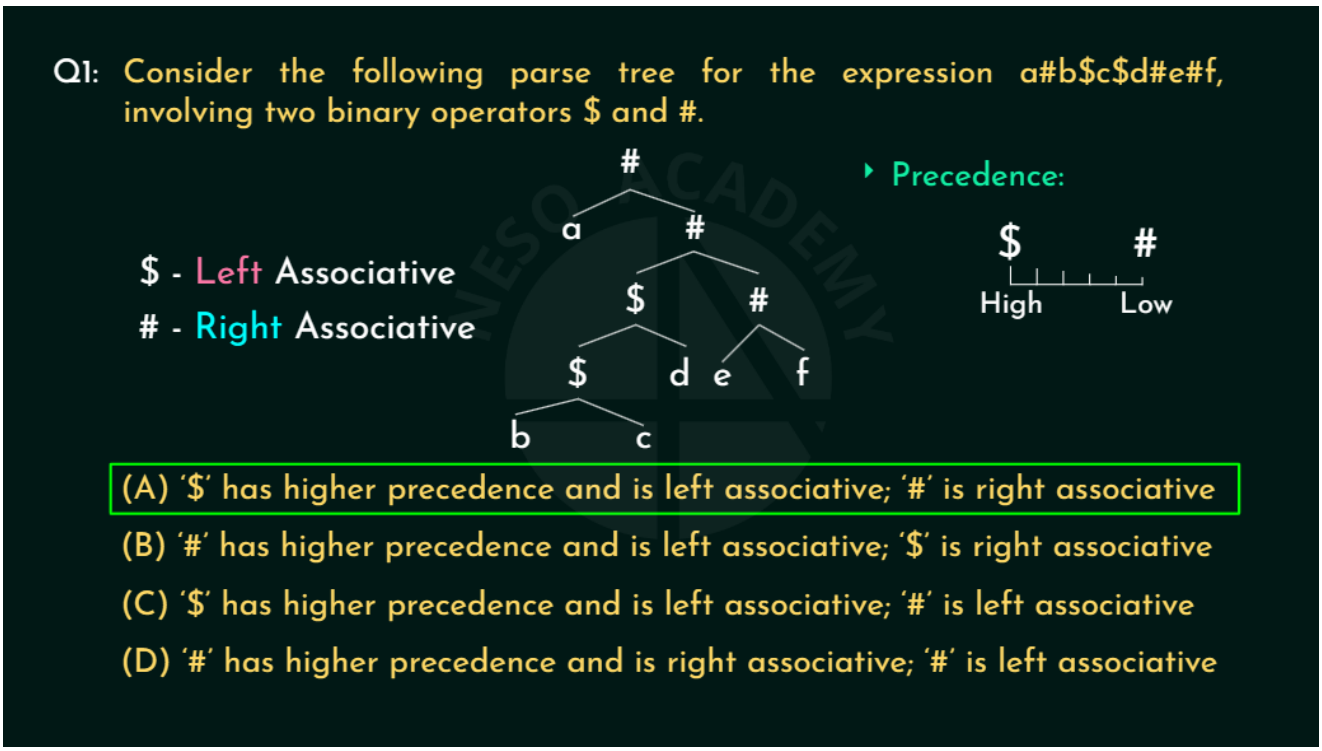
(A) 'x' has higher precedence than '+'(B) '-' has higher precedence than 'x'(C) '+' and '-' have same precedence(D) '+' has higher precedence than 'x'Q3:Given the following expression grammar:E → E x F I F + E I FF → F - F I idWhich of the following is true?GATE 2000ISRO CS 2015UGC-NET Dec 2012



Compiler DesignAssociativity and Precedence - Solved Problems (Set 2)

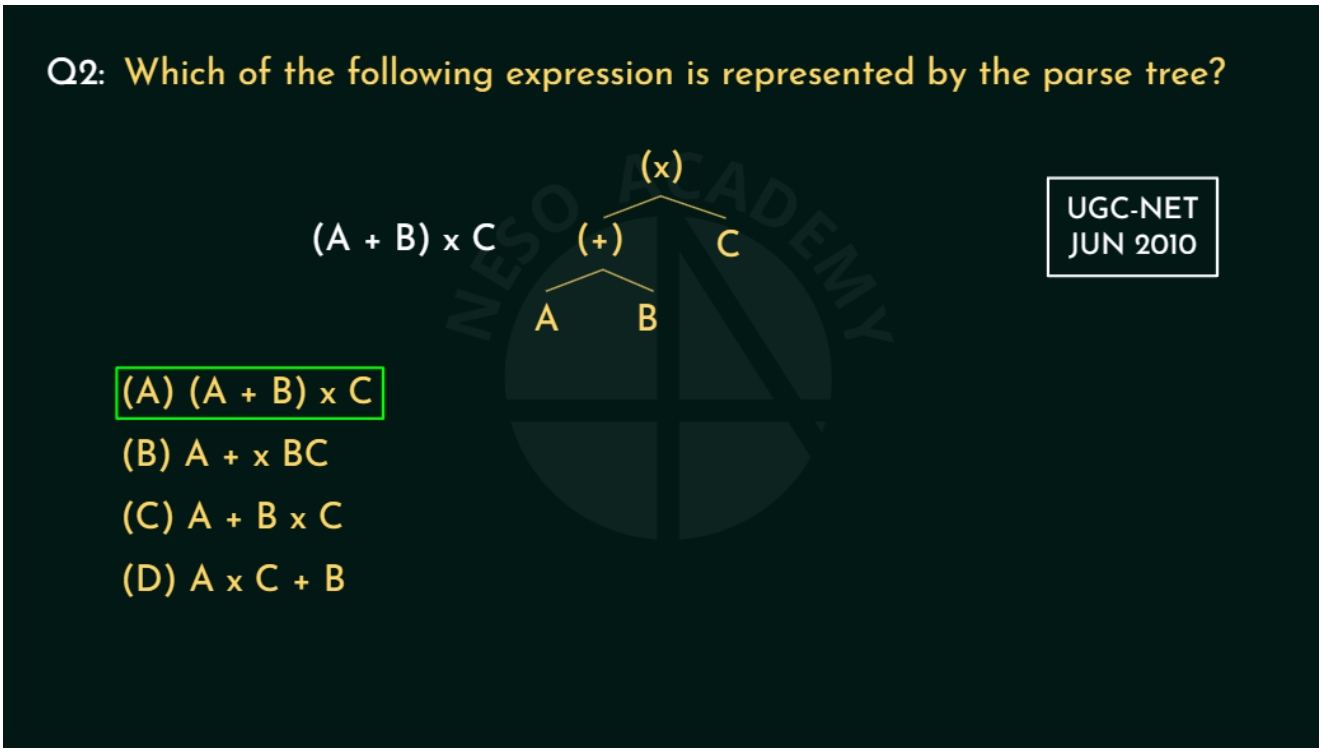


Outcome☆Four solved PYQs on determining associativity and precedence.

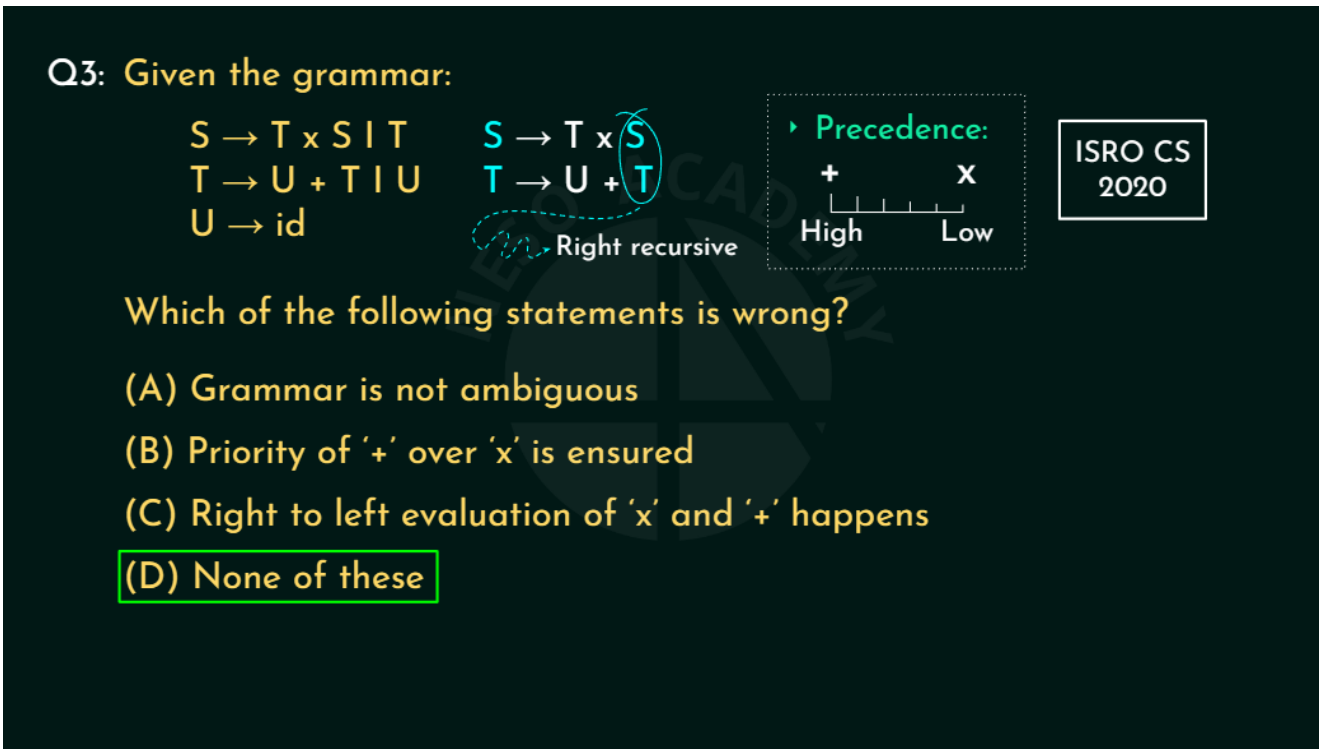


Q1:Consider the following parse tree for the expression a#bcd#e#f, involving two binary operators $ and #.a##$#$dbcef‣Precedence:$#HighLow$- Left Associative#- Right Associative(A) '$' has higher precedence and is left associative; '#' is right associative(B) '#'

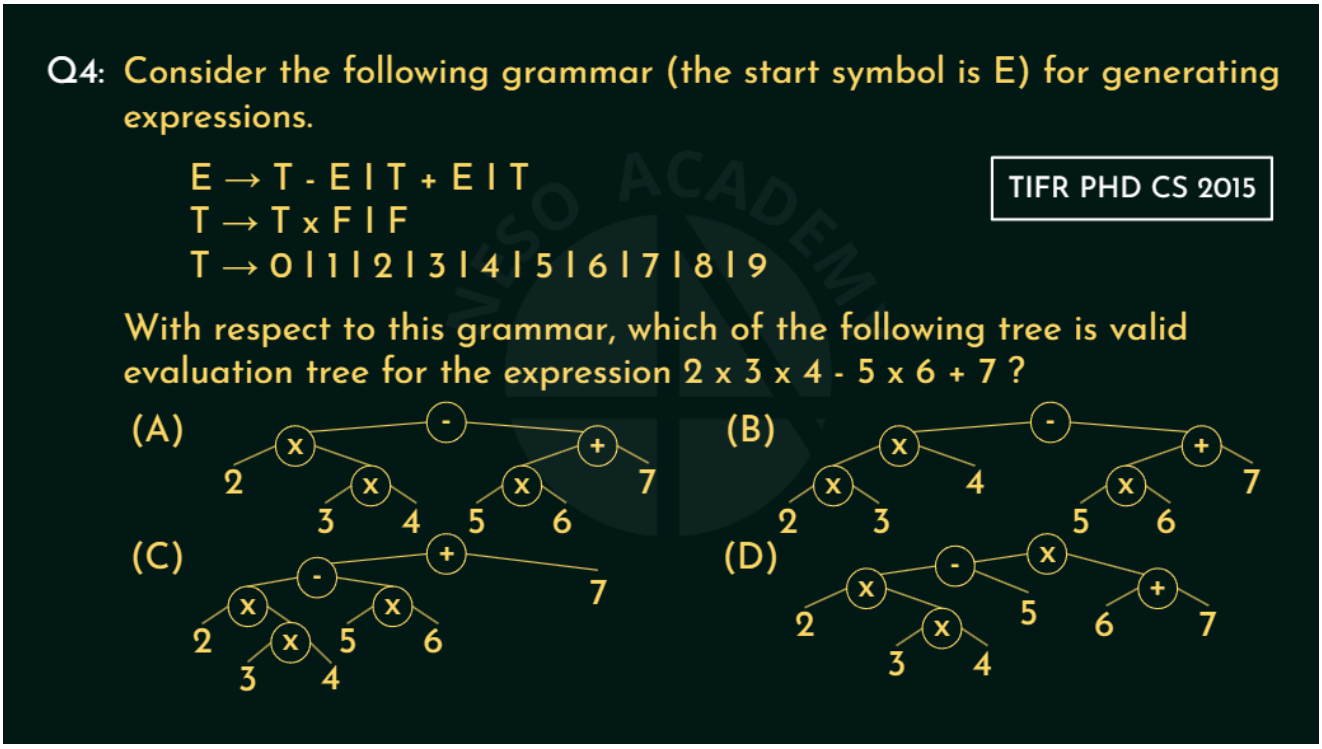
has higher precedence and is left associative; '$' is right associative(C) '$' has higher precedence and is left associative; '#' is left associative(D) '#' has higher precedence and is right associative; '#' is left associative



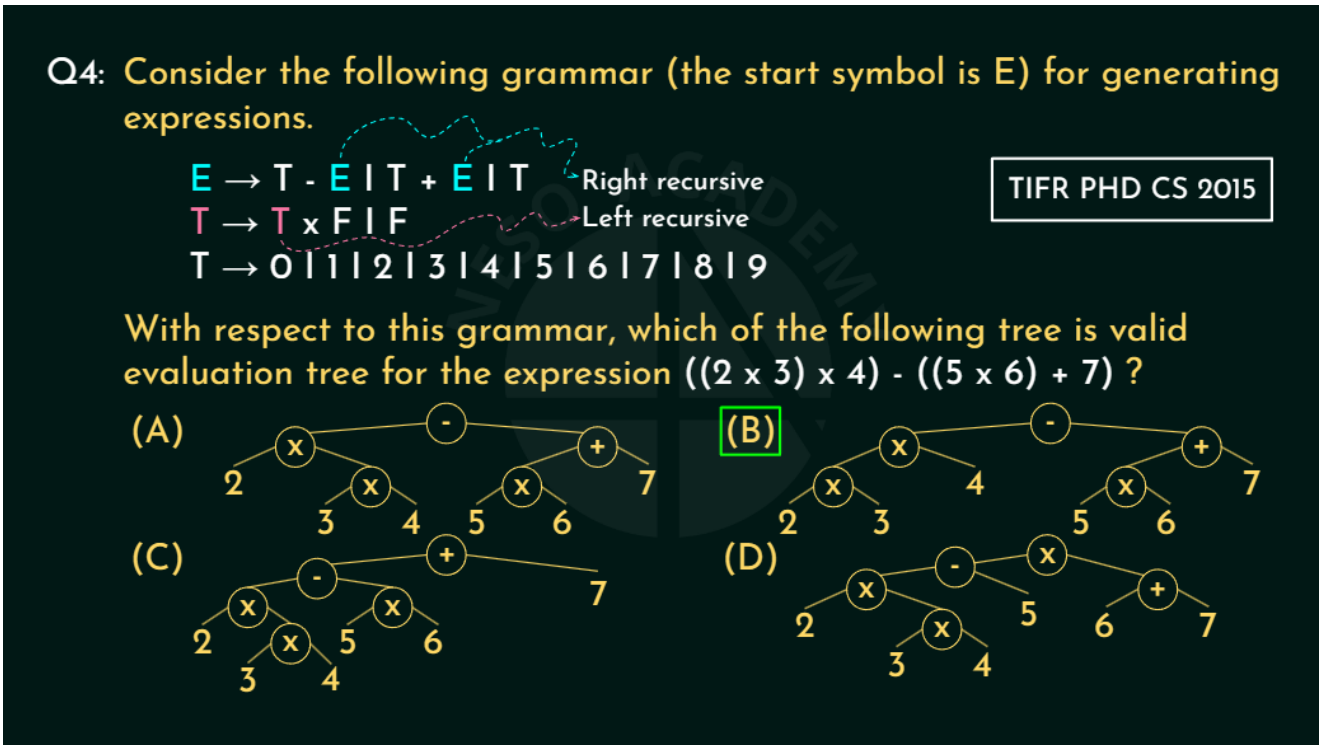
Q2:(x)CAB(A) (A + B) x C(B) A + x BC(C) A + B x C(D) A x C + B(+)UGC-NET JUN 2010(A + B) x CWhich of the following expression is represented by the parse tree?



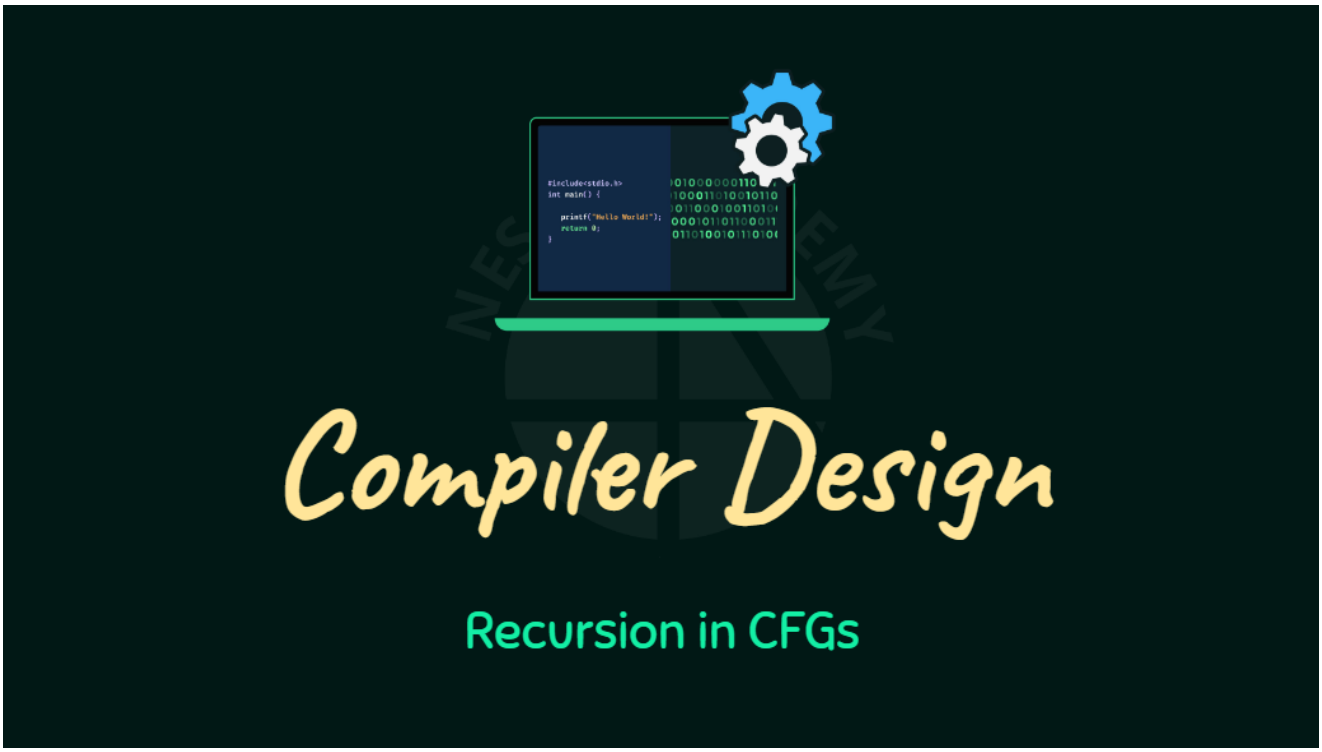
(A) Grammar is not ambiguous(B) Priority of ‘+’ over ‘x’ is ensured(C) Right to left evaluation of ‘x’ and ‘+’ happens(D) None of theseQ3:Given the grammar:S → T x S I TT → U + T I UU → idWhich of the following statements is wrong?ISRO CS 2020S → T x ST → U + TRight recursive+xHighLow‣Precedence:



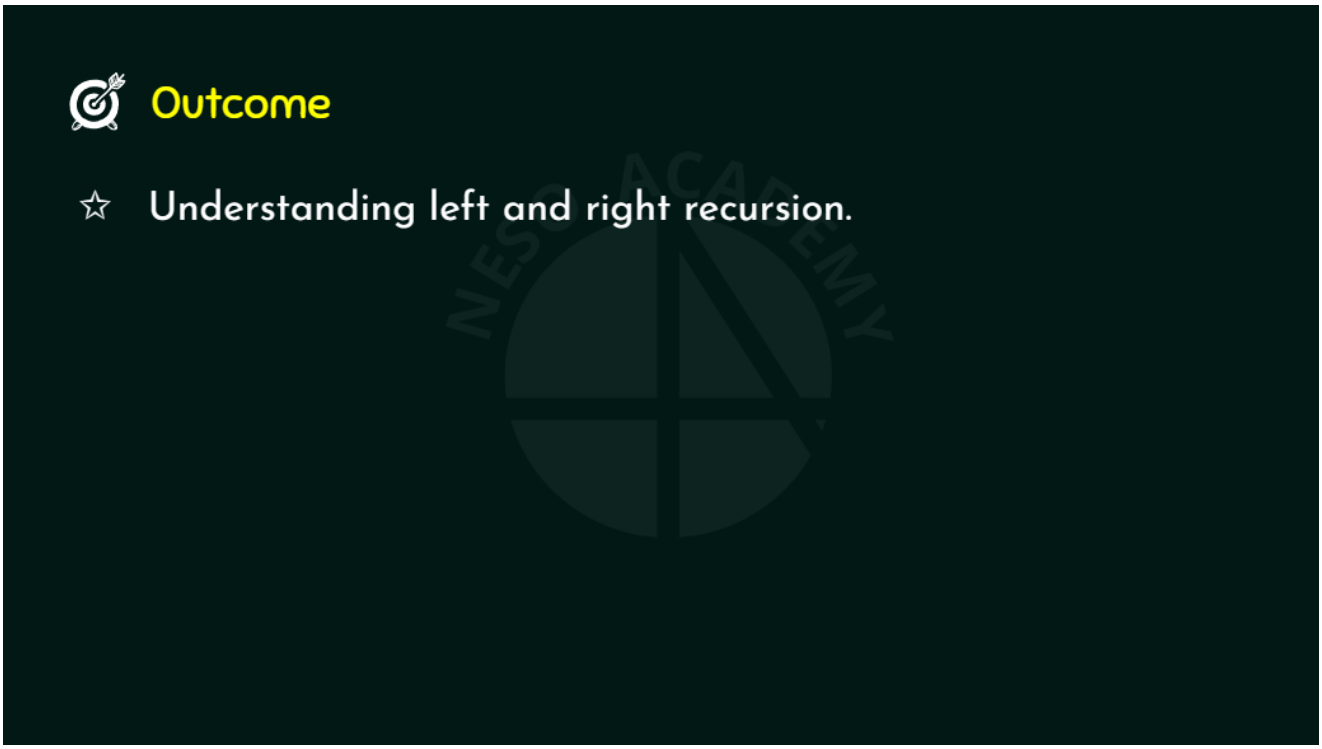
3x4732x(A)Q4:Consider the following grammar (the start symbol is E) for generating expressions.E → T - E I T + E I TT → T x F I FT → 0 I 1 I 2 I 3 I 4 I 5 I 6 I 7 I 8 I 9With respect to this grammar, which of the following tree is validevaluation tree for the expression 2 x 3 x 4 - 5 x 6 + 7 ?TIFR PHD CS 2015(B)(C)(D)-x+x276543x-x+x7654+-65xx2-x243xx5+76



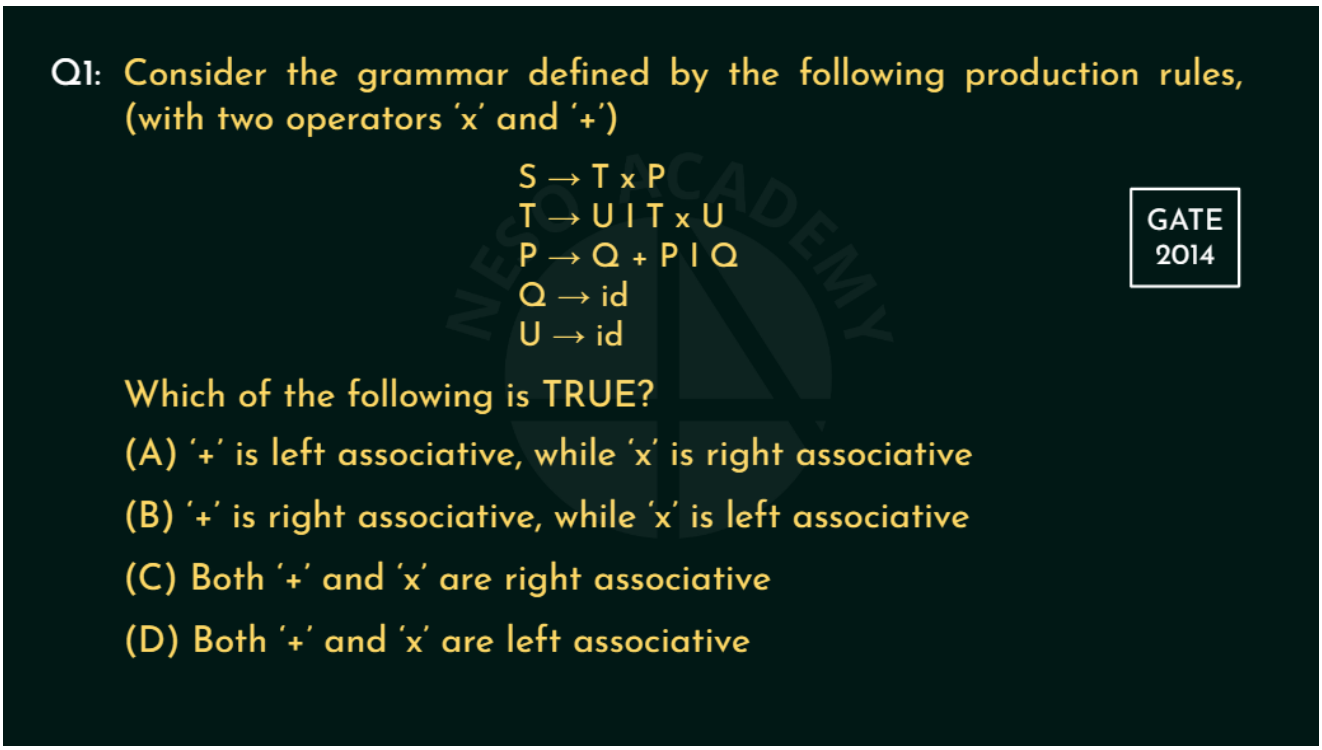
3x4732x(A)Q4:Consider the following grammar (the start symbol is E) for generating expressions.E → T - E I T + E I TT → T x F I FT → 0 I 1 I 2 I 3 I 4 I 5 I 6 I 7 I 8 I 9With respect to this grammar, which of the following tree is validevaluation tree for the expression ((2 x 3) x 4) - ((5 x 6) + 7) ?TIFR PHD CS 2015(B)(C)(D)-x+x276543x-x+x7654+-65xx2-x243xx5+76Left recursiveRight recursive



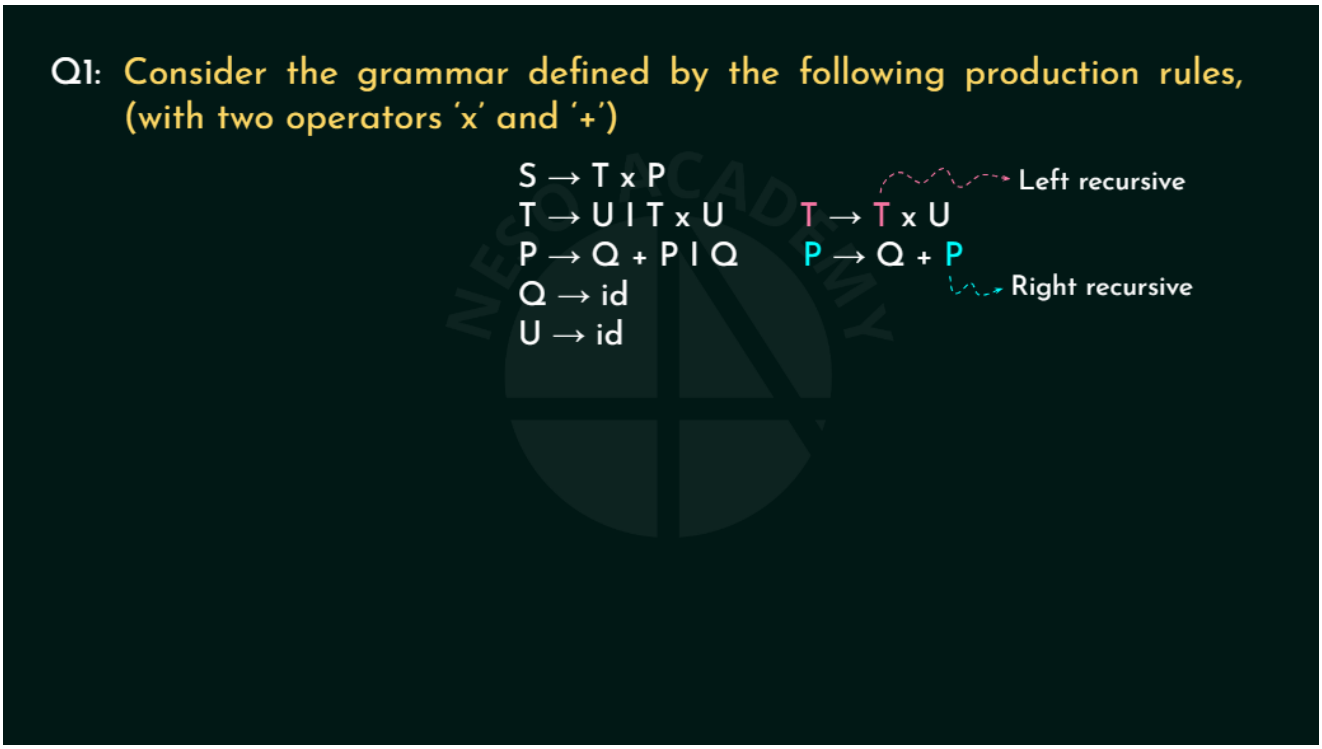
Compiler DesignRecursion in CFGs



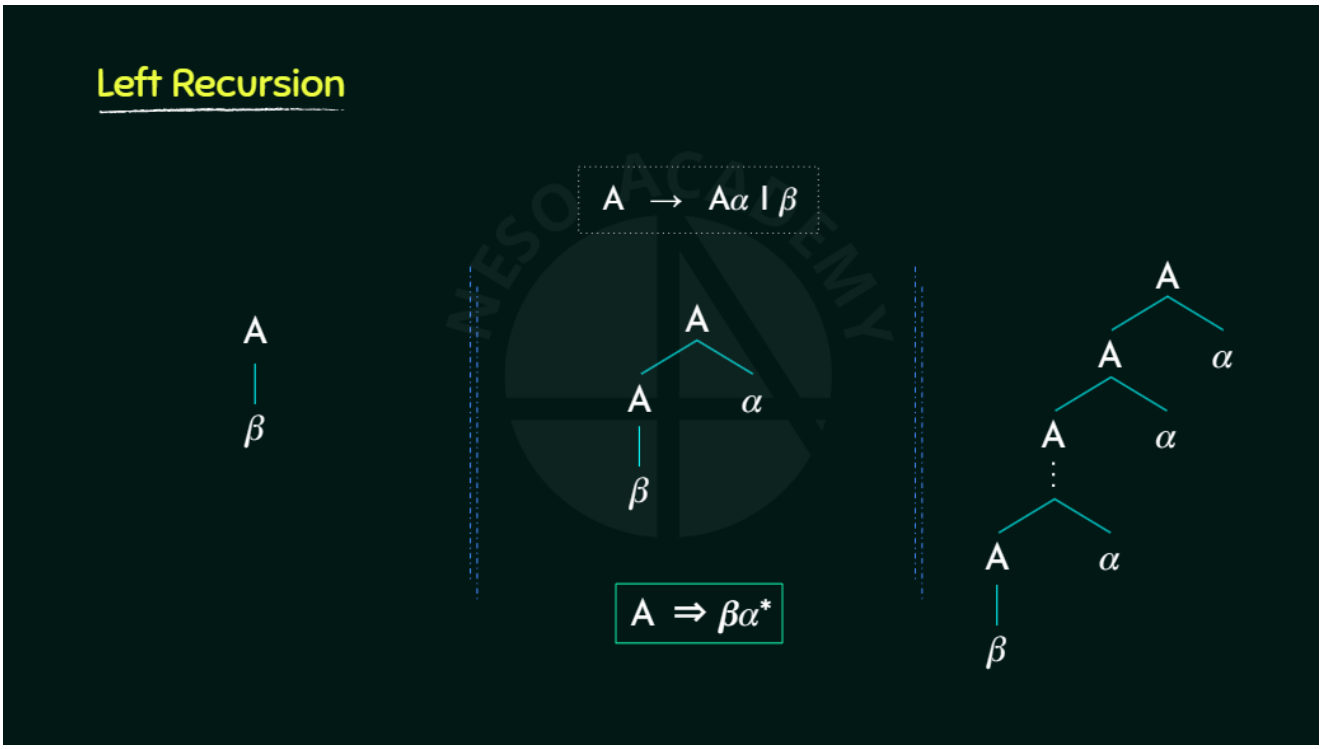
Outcome☆Understanding left and right recursion.



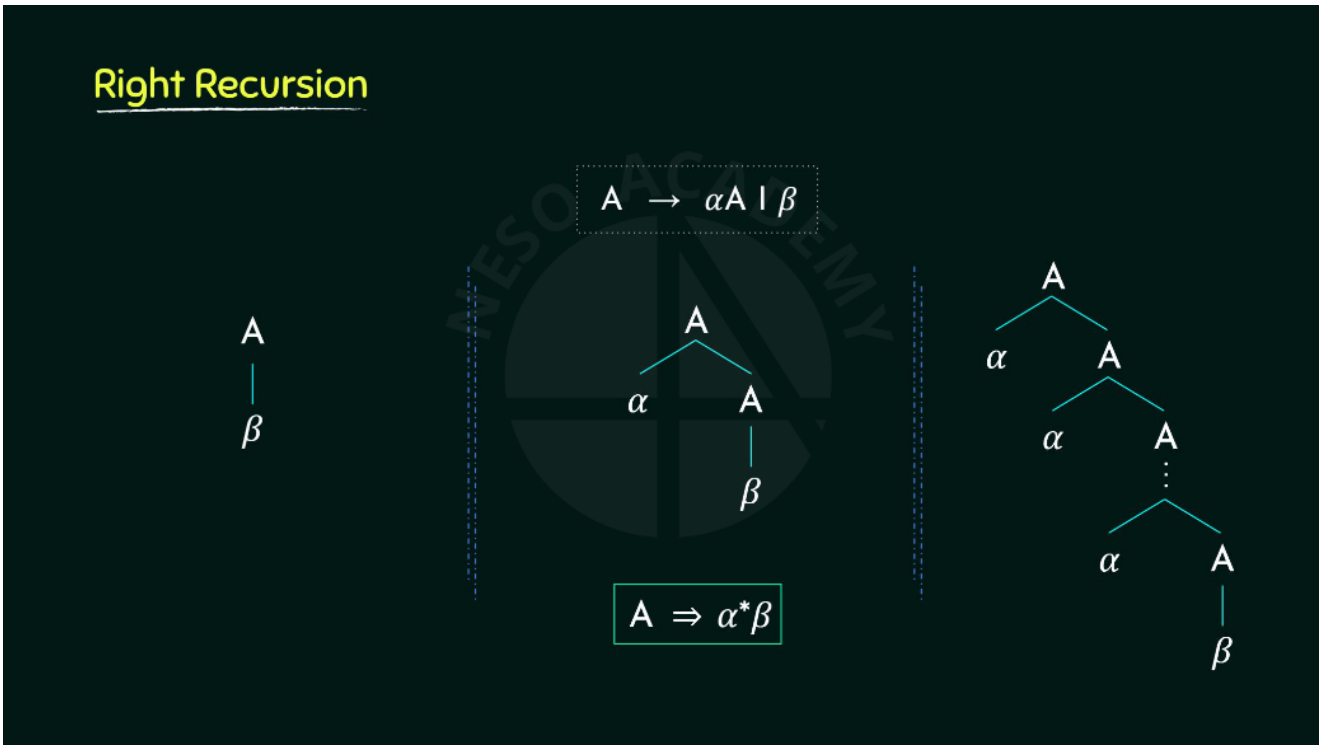
Q1:Consider the grammar defined by the following production rules, (with two operators ‘x’ and ‘+’)S → T x PT → U | T x UP → Q + P | QQ → idU → idWhich of the following is TRUE? GATE 2014(A) ‘+’ is left associative, while ‘x’ is right associative(B) ‘+’ is right associative, while ‘x’ is left associative(C) Both ‘+’ and ‘x’ are right associative(D) Both ‘+’ and ‘x’ are left associative

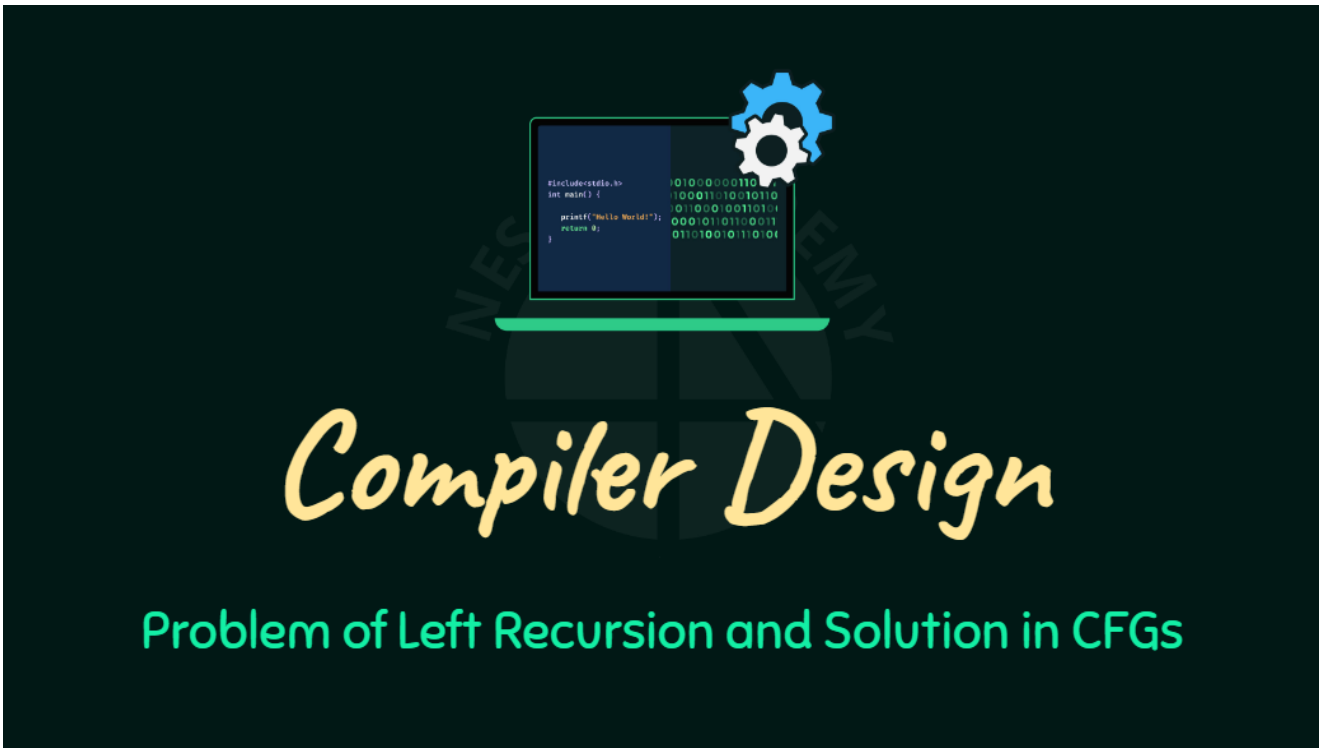


Q1:Consider the grammar defined by the following production rules, (with two operators 'x' and '+')S → T x PT → U | T x UP → Q + P | QQ → idU → idT → T x ULeft recursiveP → Q + PRight recursive



Aβα*⇒ Left RecursionA→Aα | ββAAβAαAAαAαβAαβ





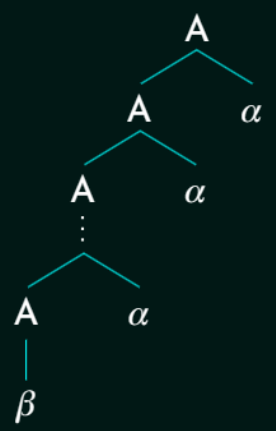
Compiler DesignProblem of Left Recursion and Solution in CFGs

Outcome

- ☆ Problem due to Left recursion.
- ☆ Solution provided by Right recursion.
- ☆ Conversion of Left recursion to Right recursion.

Outcome☆Problem due to Left recursion. ☆Solution provided by Right recursion. ☆Conversion of Left recursion to Right recursion.

Left Recursion

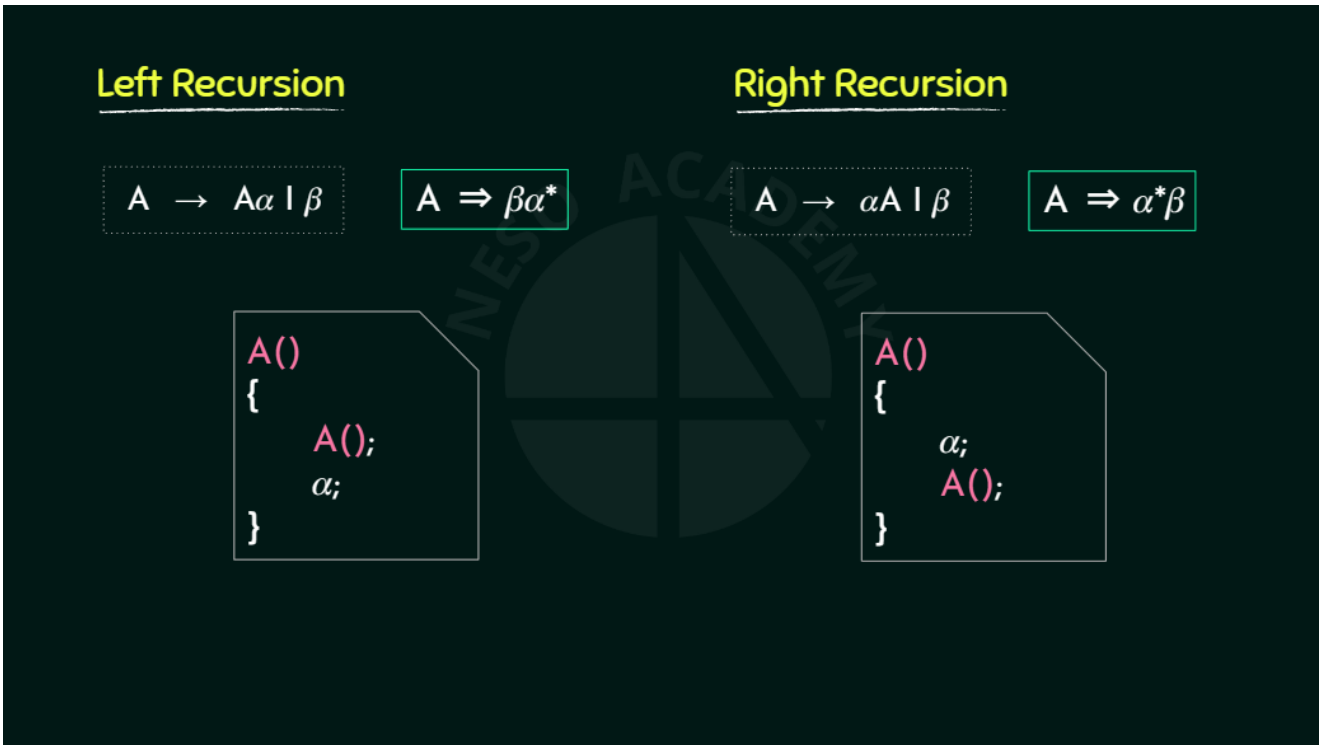


$A \rightarrow A\alpha \mid \beta$

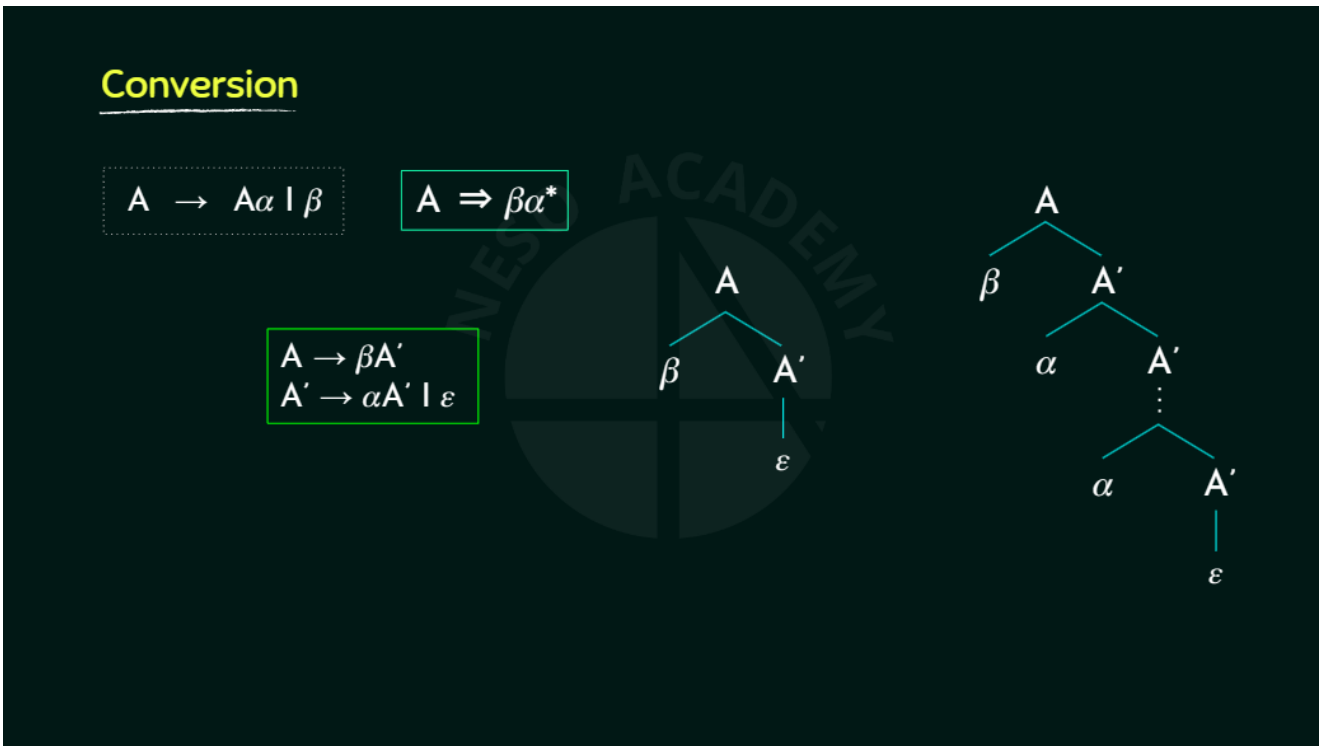
$A \Rightarrow \beta\alpha^*$

```
A()
{
    A();
    α;
}
```

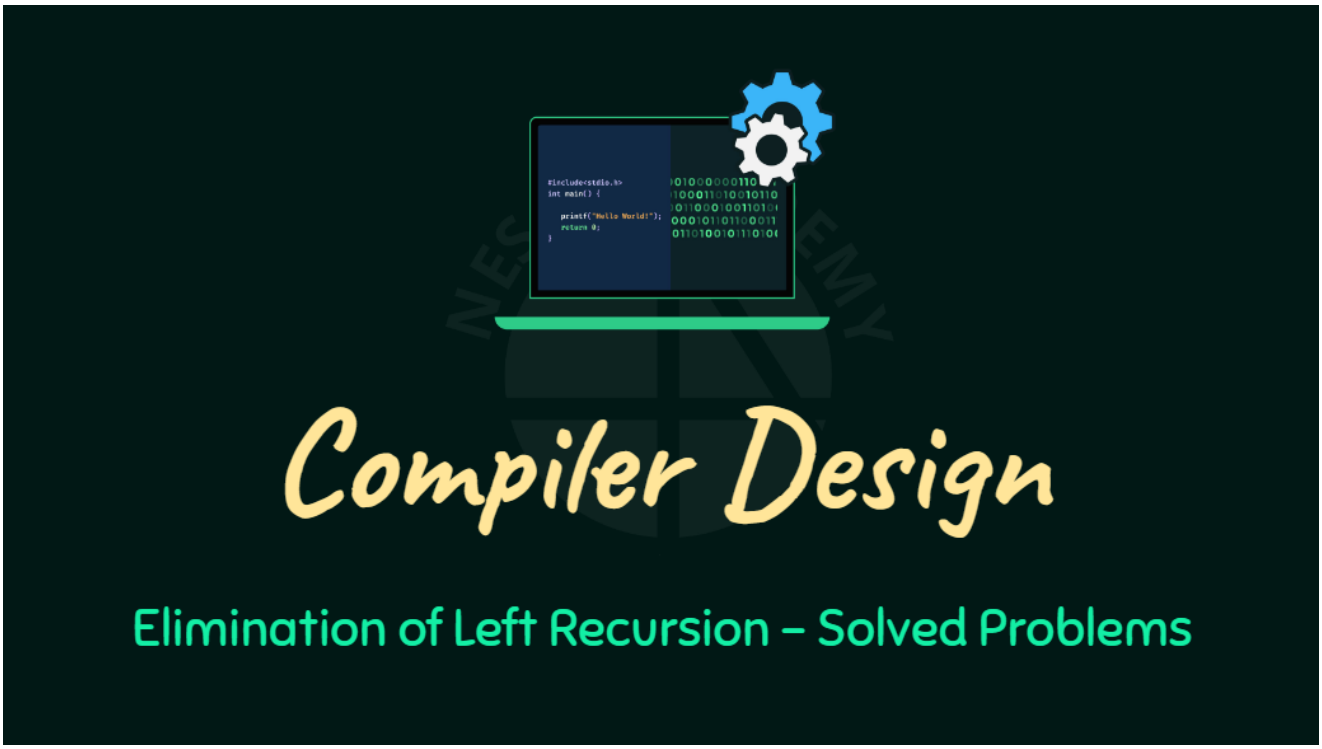
Aβα*⇒ Left RecursionA→Aα I βAAαAαβAαβA(){A();α;}



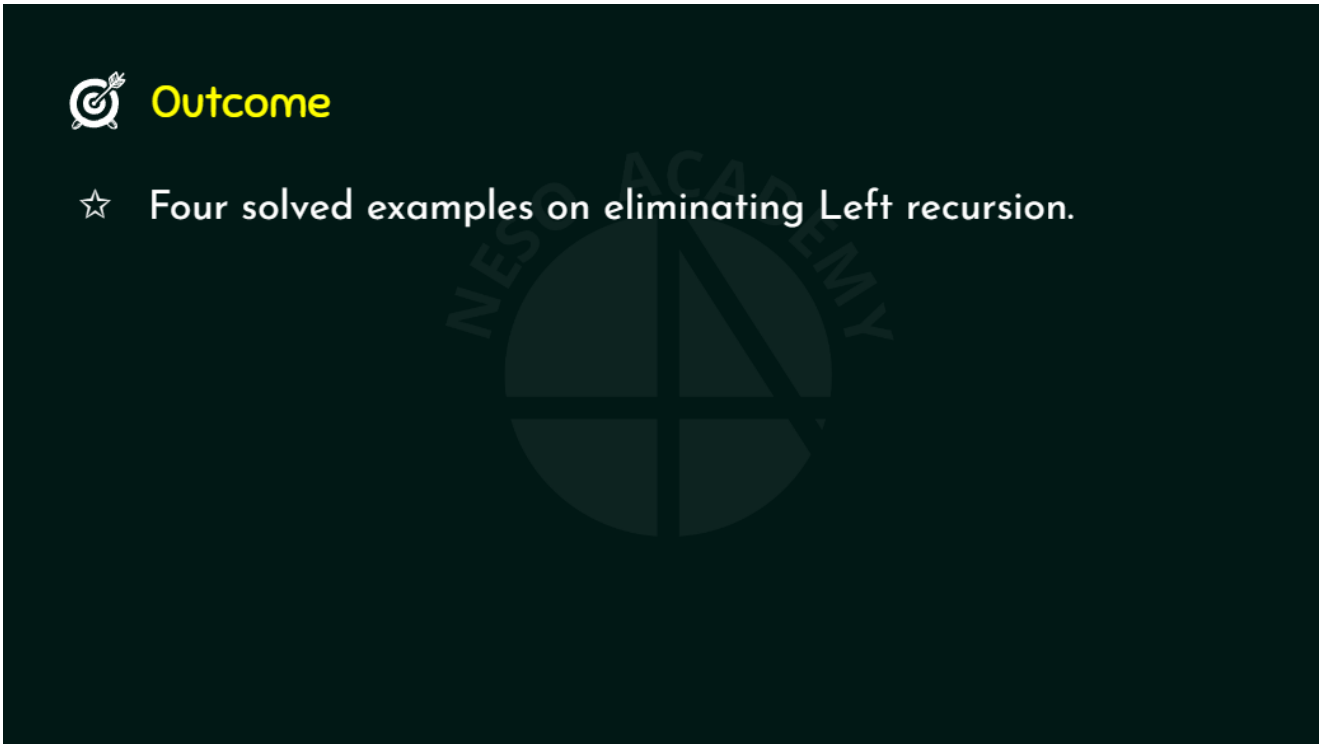
Left RecursionA→Aα I βAβα*⇒ A(){A();α;}Right RecursionA→αA I βAα*β⇒ A(){α;A();}



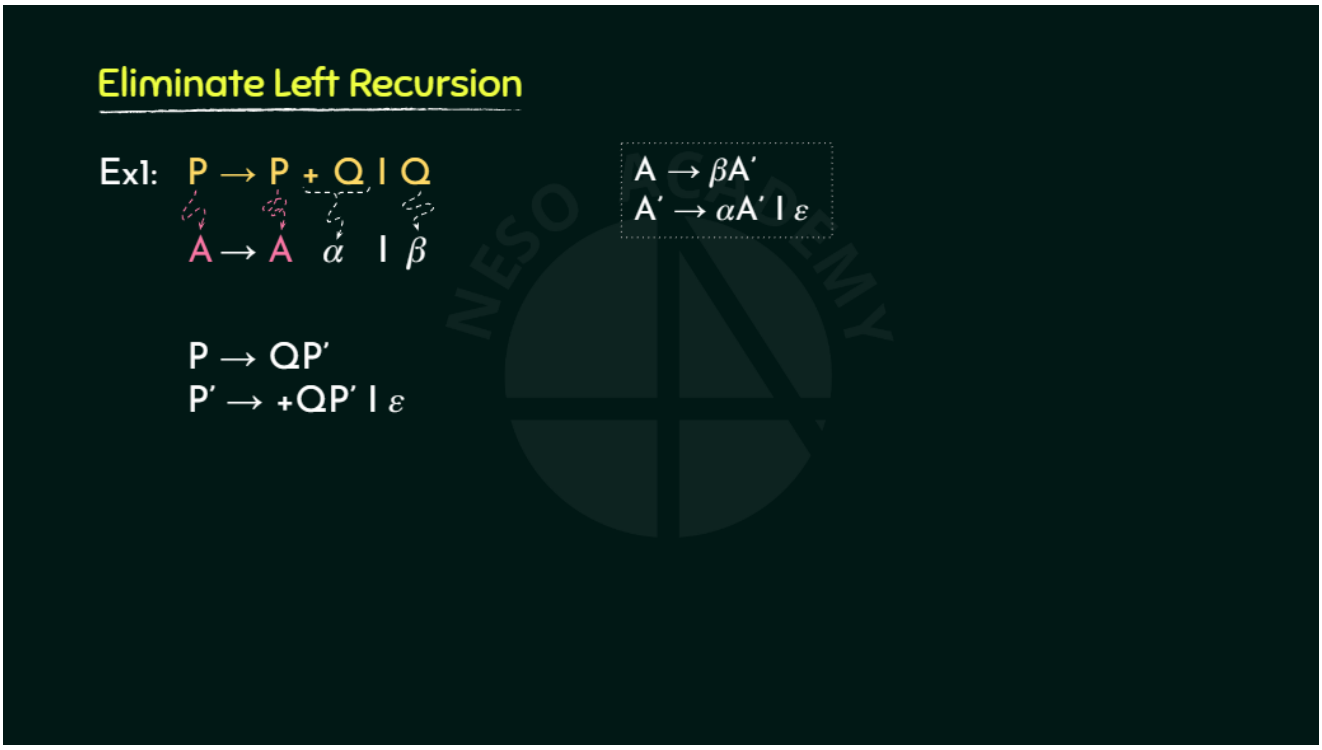
ConversionA→Aα I βAβα*⇒ A → βA'A' → αA' I εAA'βA'αεA'αεA'βA



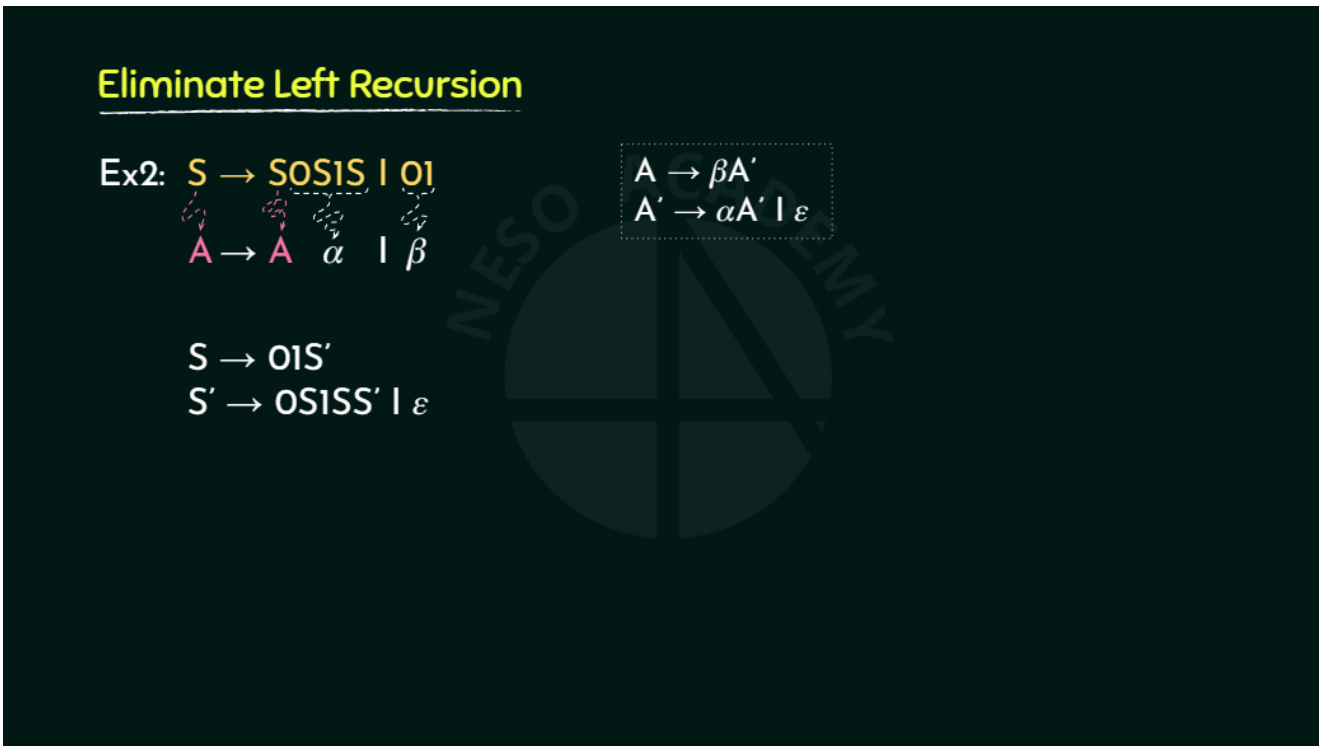
Compiler DesignElimination of Left Recursion - Solved Problems



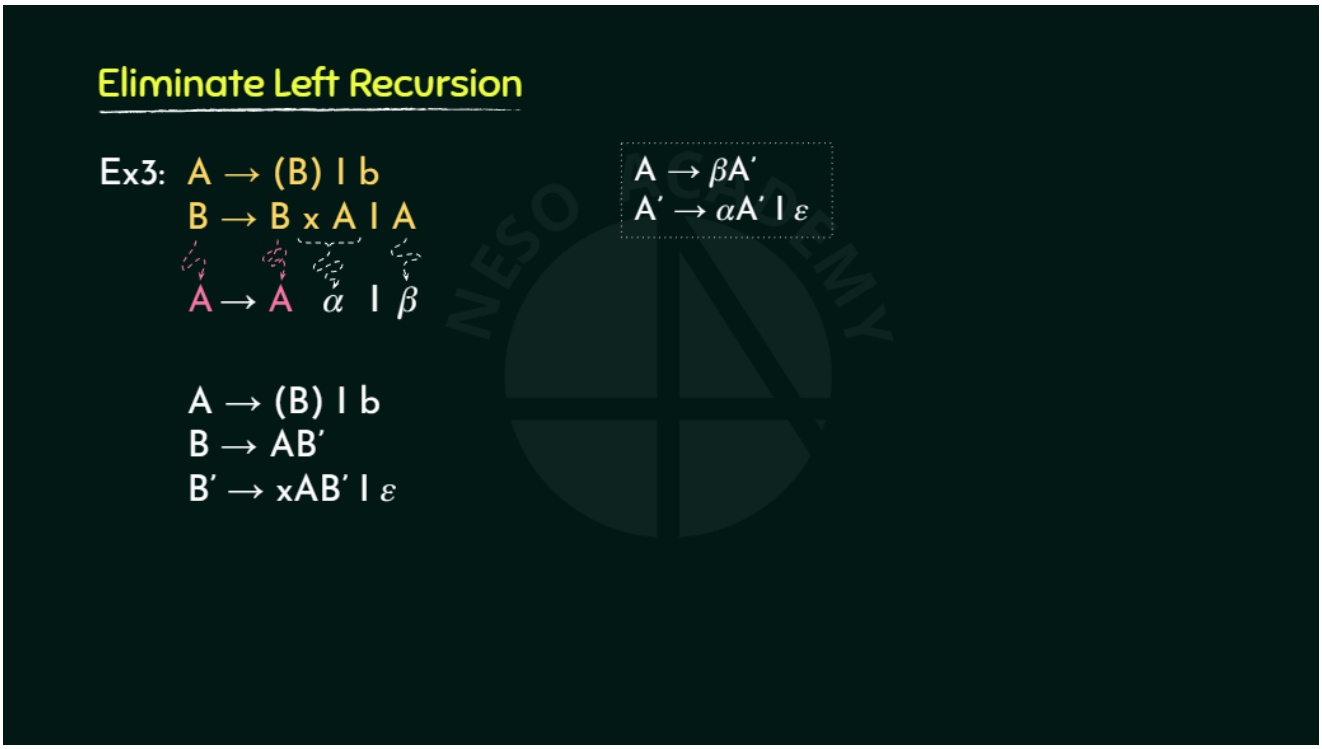
Outcome☆Four solved examples on eliminating Left recursion.



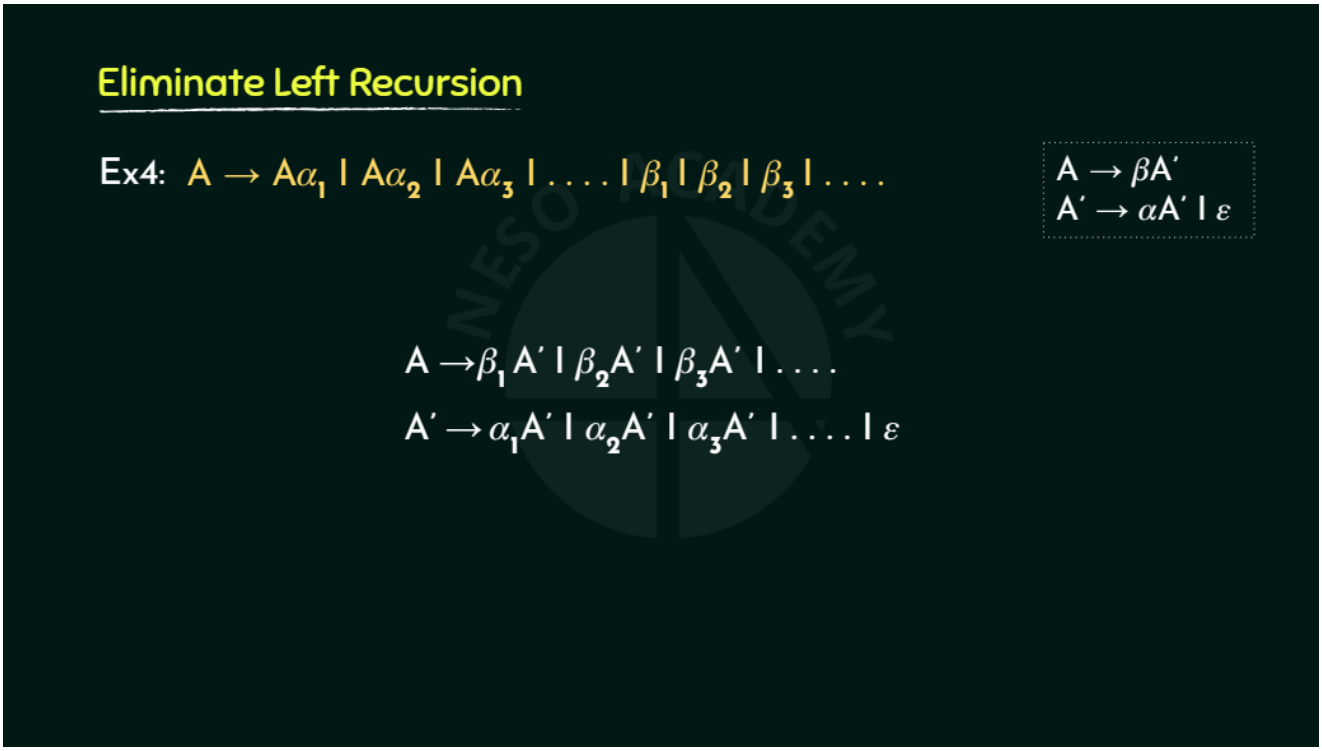
+QP'Ex1:P → P + Q I QEliminate Left RecursionAAα→βIA → βA'A' → αA' I εP→QP'→I εP'



Ex2:S → S0S1S I 01Eliminate Left RecursionAAα→βIA → βA'A' → αA' I εS→01S'S'→0S1SI εS'

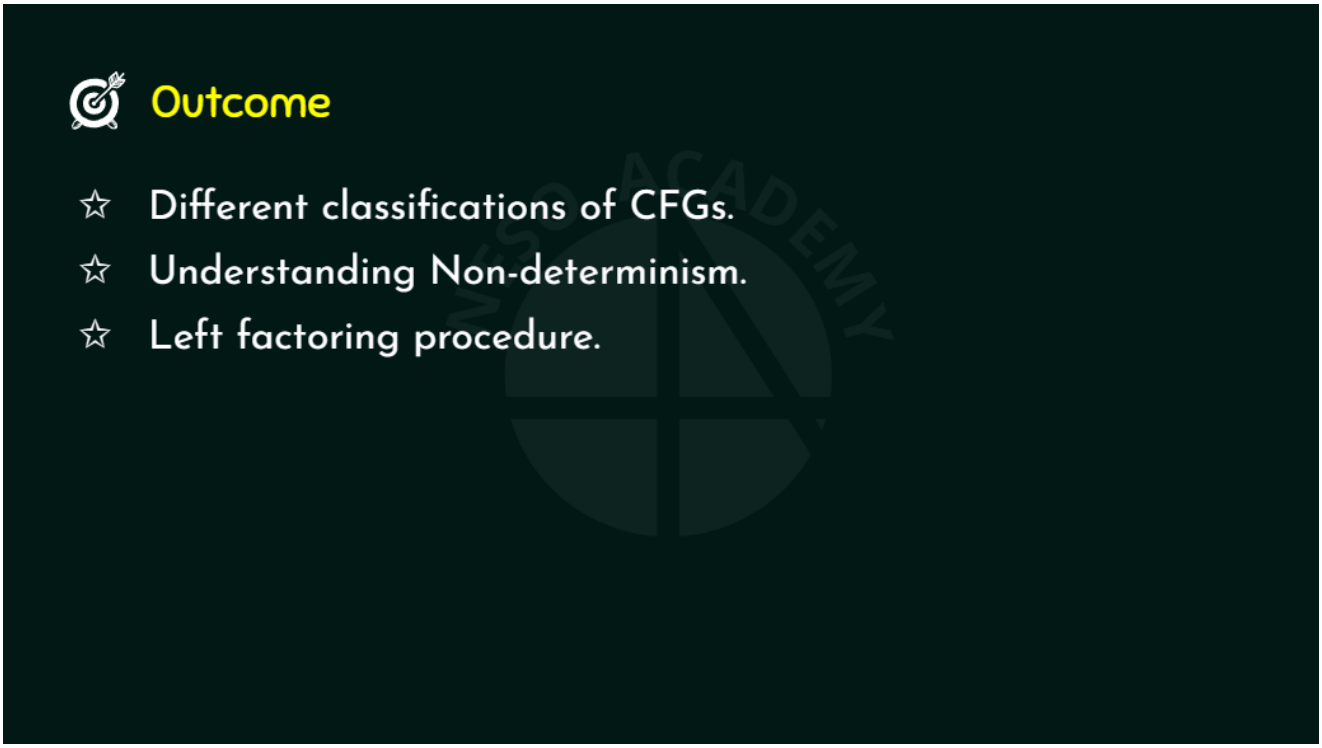


A → (B) I bEx3:A → (B) I bB → B x A I AEliminate Left RecursionA → βA'A' → αA' I εB→AB'B'→xAI εAAα→βIB'

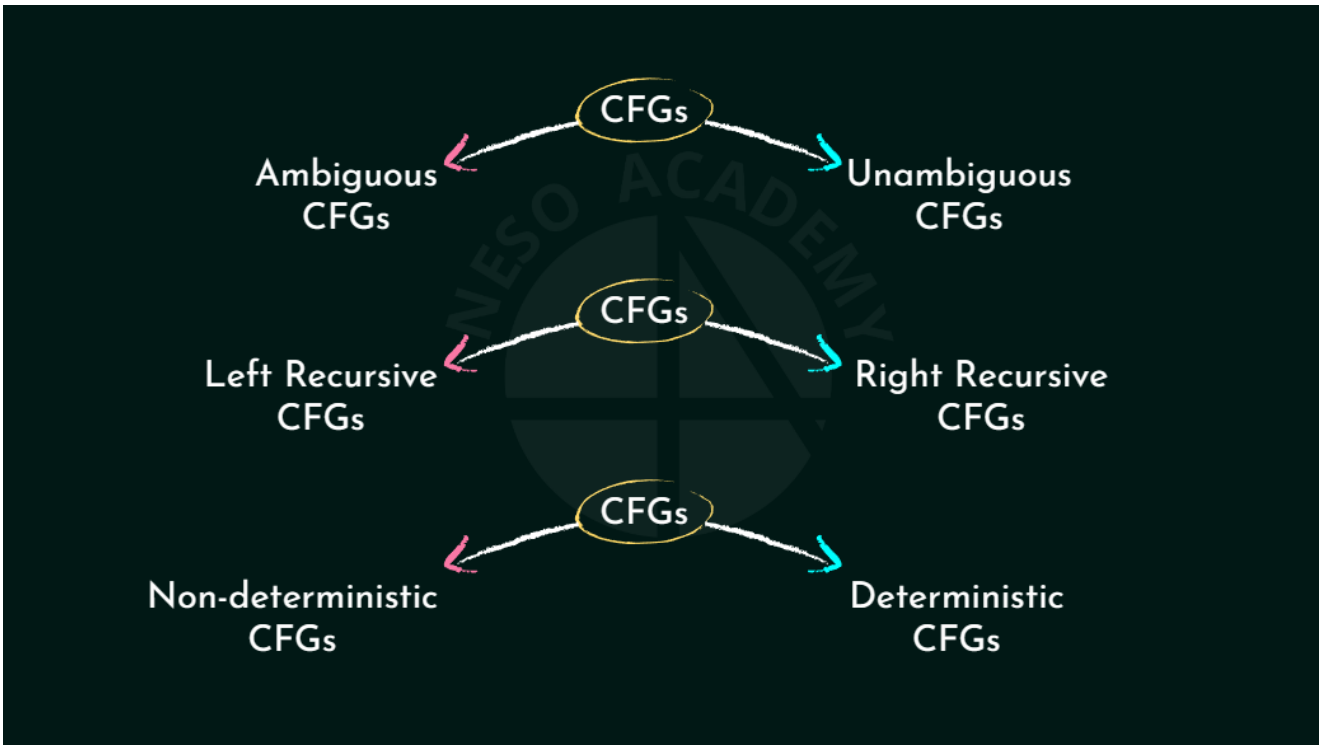


α1A'Ex4:A → Aα1 I Aα2 I Aα3 I . . . . I β1 I β2 I β3 I . . . .Eliminate Left RecursionA → βA'A' → αA' I εA→β1A'A'→I β2A'I β3A'I . . . .I α2A'I α3A'I . . . . I ε

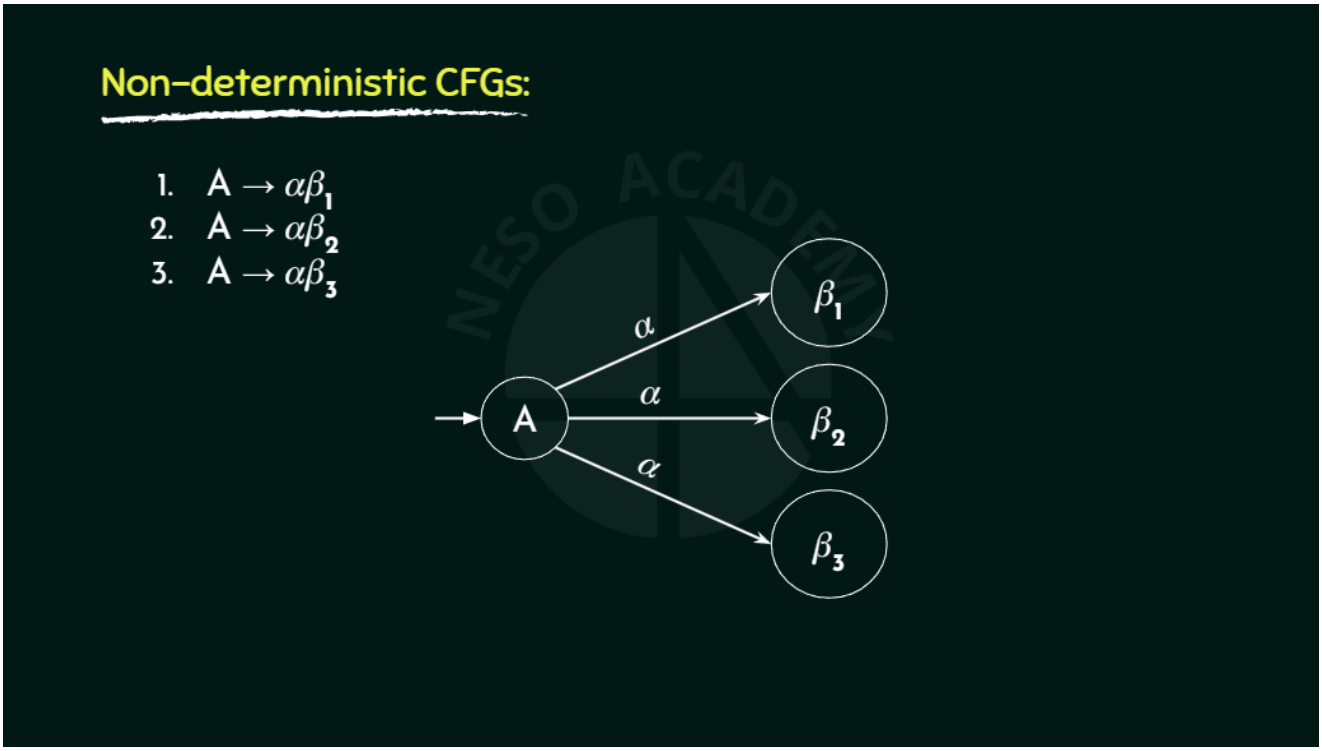
Compiler DesignNon-deterministic CFGs



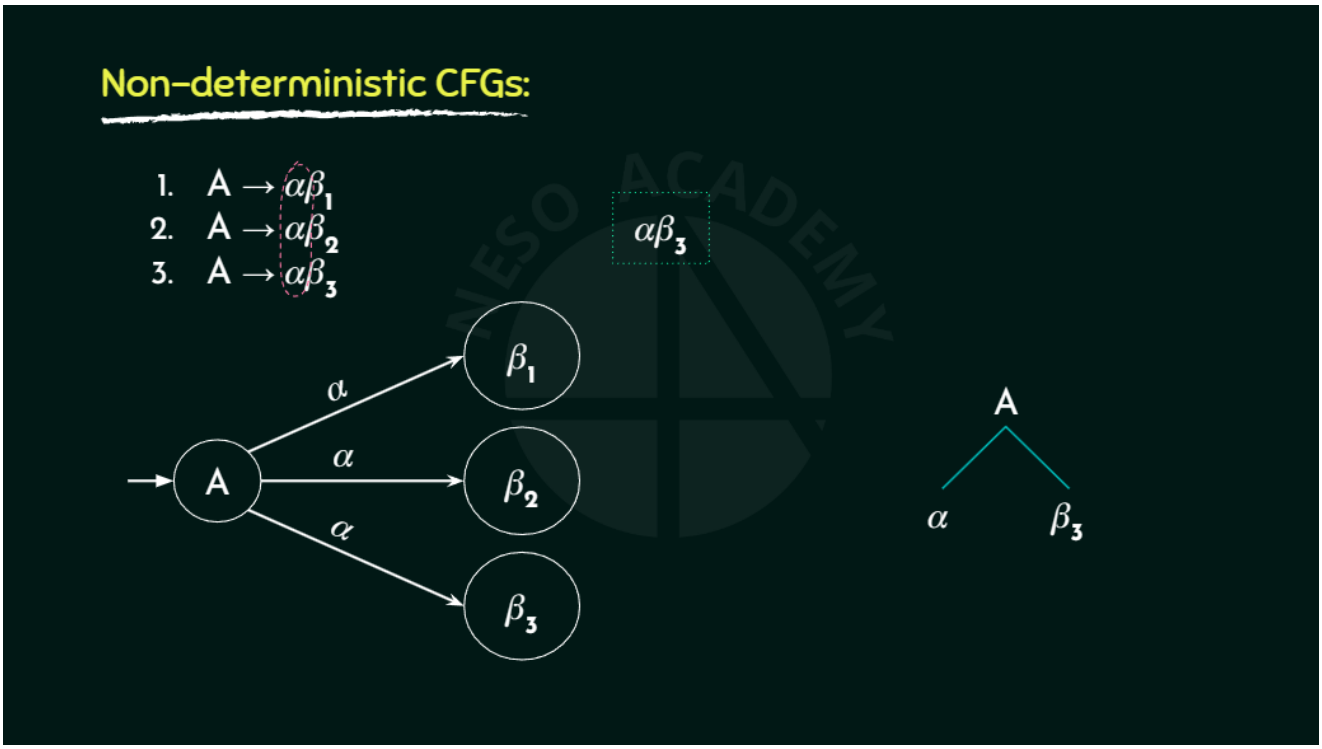
Outcome☆Different classifications of CFGs. ☆Understanding Non-determinism. ☆Left factoring procedure.



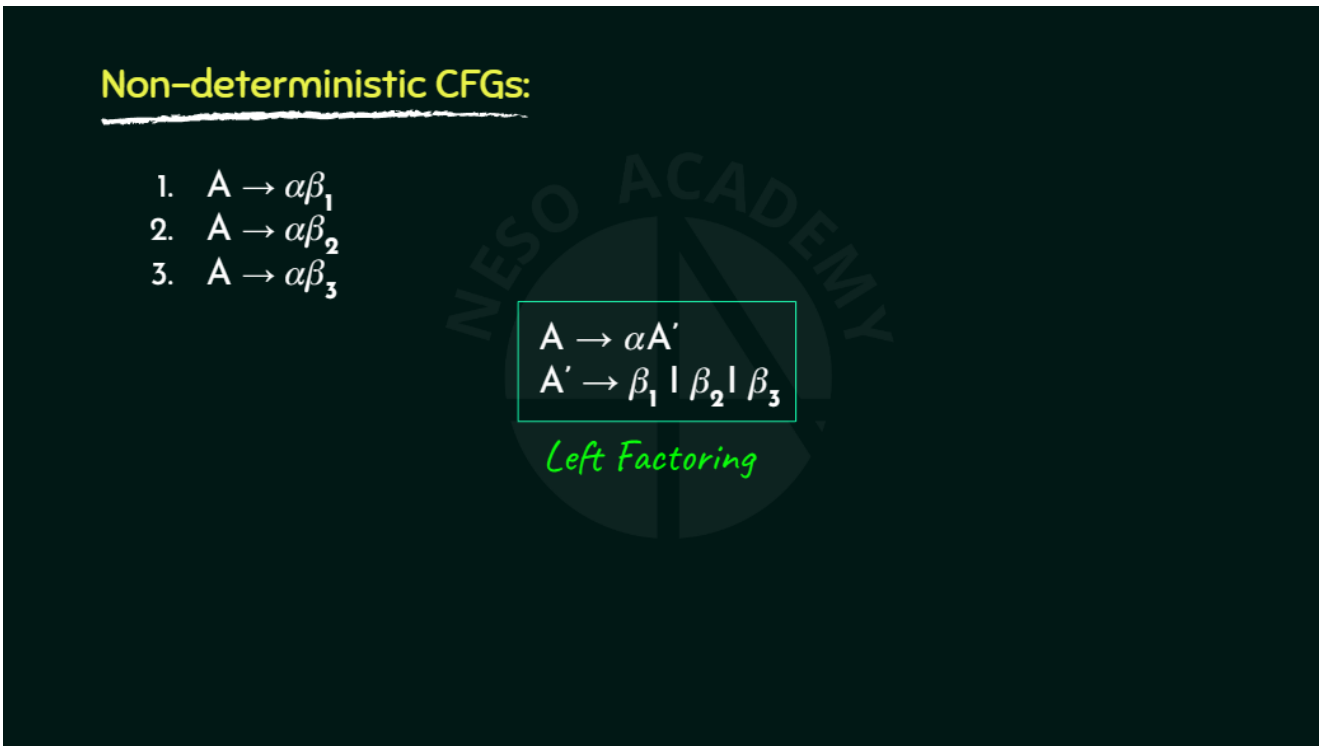
CFGsAmbiguousCFGsUnambiguousCFGsCFGsLeft RecursiveCFGsRight RecursiveCFGsCFGsNon-deterministicCFGsDeterministicCFGs



Non-deterministic CFGs:1.A → $\alpha\beta$12.A → $\alpha\beta$23.A → $\alpha\beta$3Aβ1$\alpha\beta$2β3$\alpha\alpha$

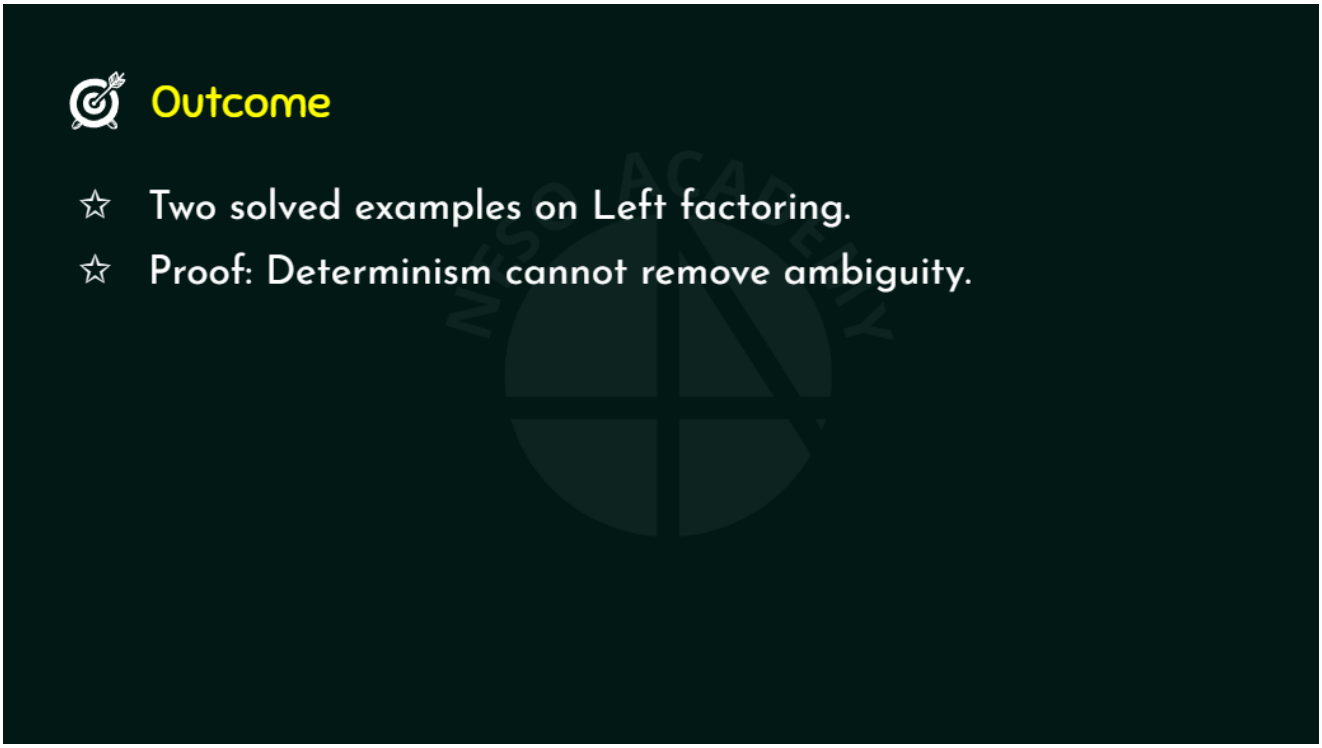


Non-deterministic CFGs:1.A → $\alpha\beta$12.A → $\alpha\beta$23.A → $\alpha\beta$3$\alpha\beta$3β3αAAβ1$\alpha\beta$2β3$\alpha\alpha$

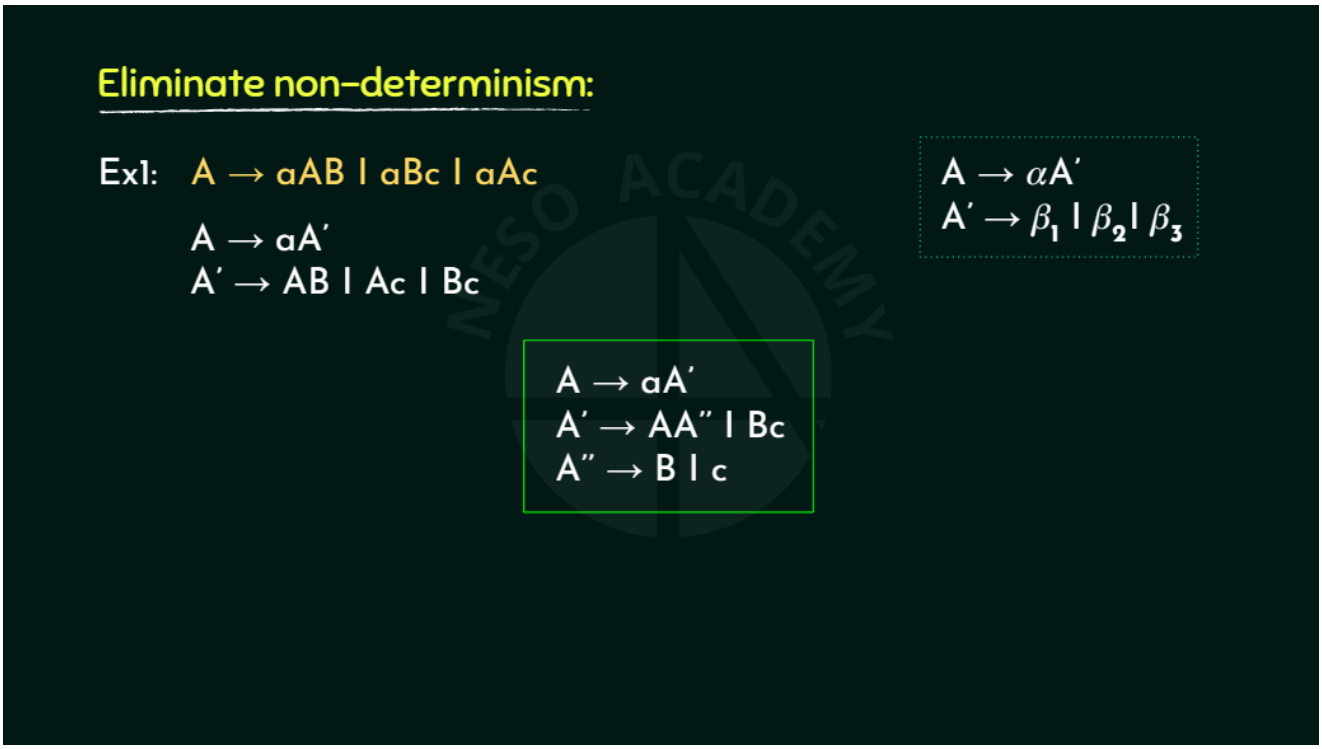


Non-deterministic CFGs:1.A → $\alpha\beta$12.A → $\alpha\beta$23.A → $\alpha\beta$3I β2β1A→αA'→A'I β3Left Factoring

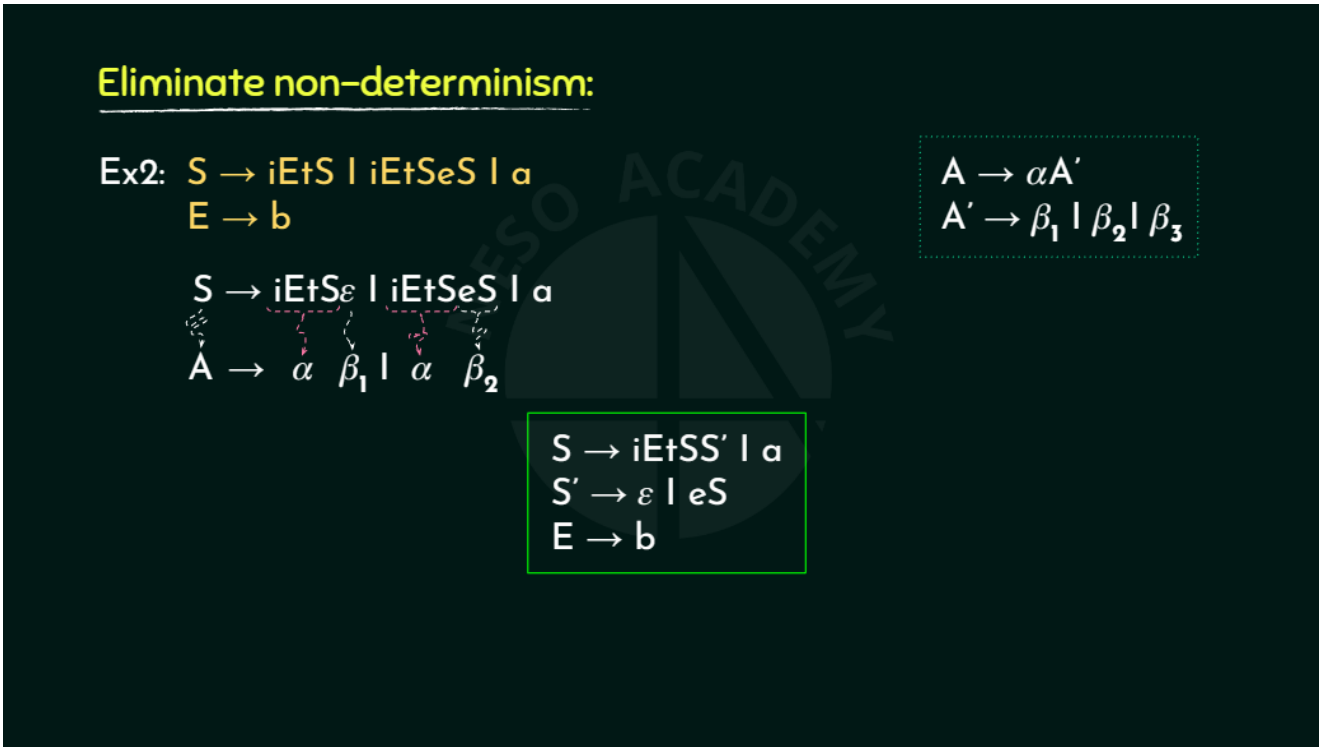
Compiler DesignNon-deterministic CFGs - Solved Problems (Set 1)



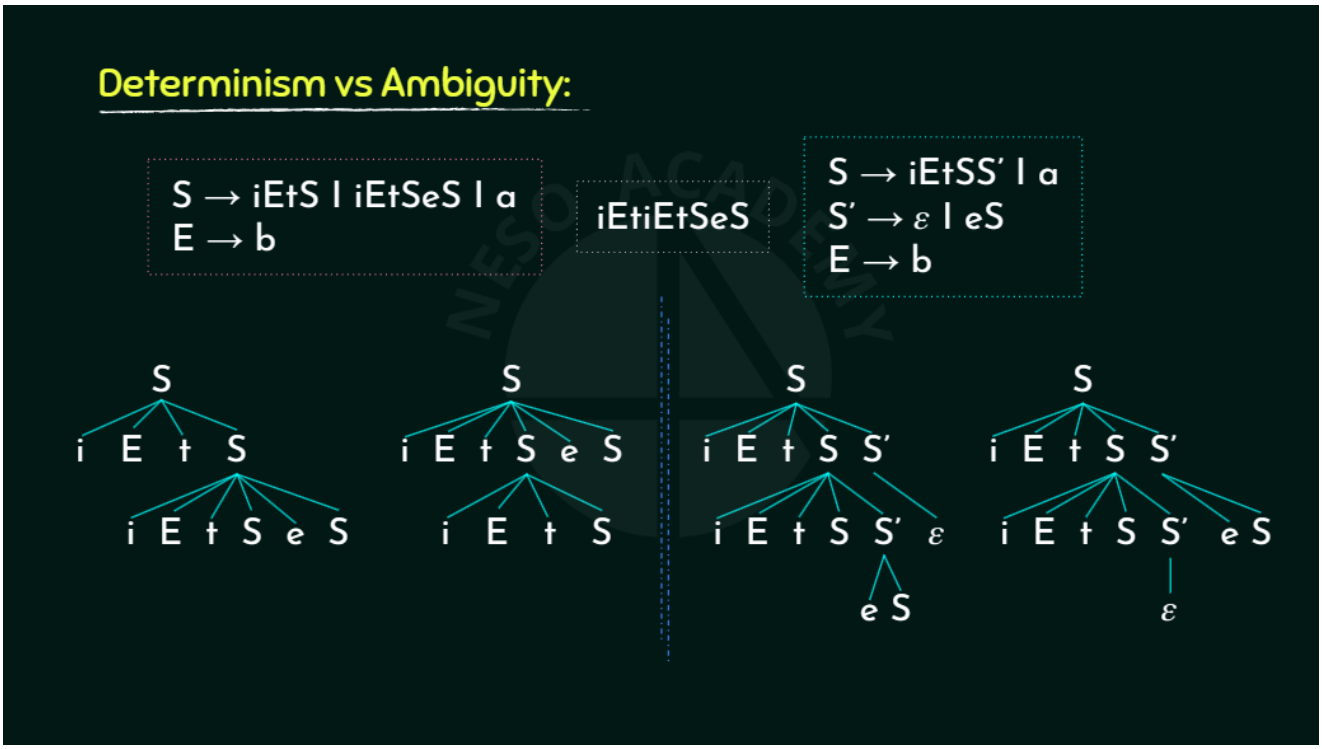
Outcome☆Two solved examples on Left factoring. ☆Proof: Determinism cannot remove ambiguity.



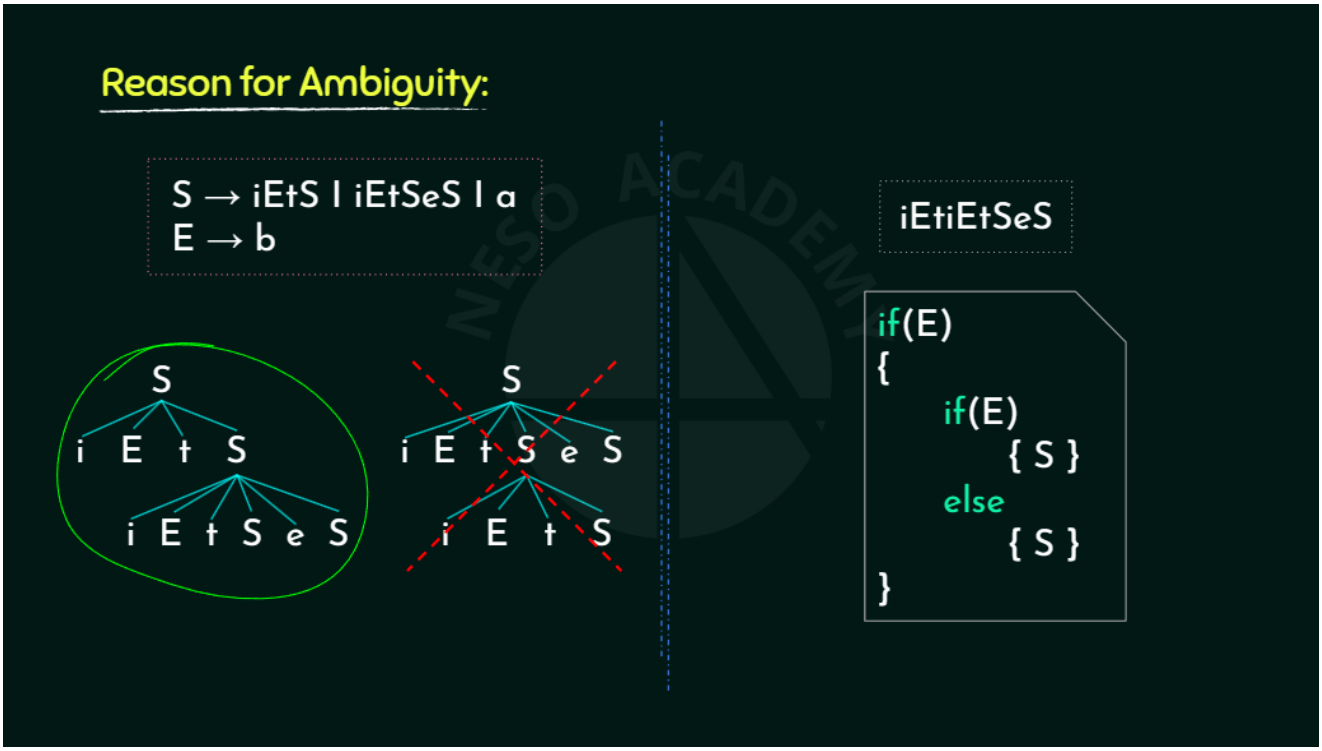
A → aA'A' → AA'' I BcA'' → B I c A → aAB I aBc I aAcEx1:Eliminate non-determinism:I β2β1A→αA'→A'I β3A → aA'A' → AB I Ac I Bc



Ex2:Eliminate non-determinism:S → iEtS I iEtSeS I aE → bI β2β1A→αA'→A'I β3S → iEtSS' I aS' → ε I eSE → bS → iEtSε I iEtSeS I a→A$\alpha\beta$1I$\alpha\beta$2

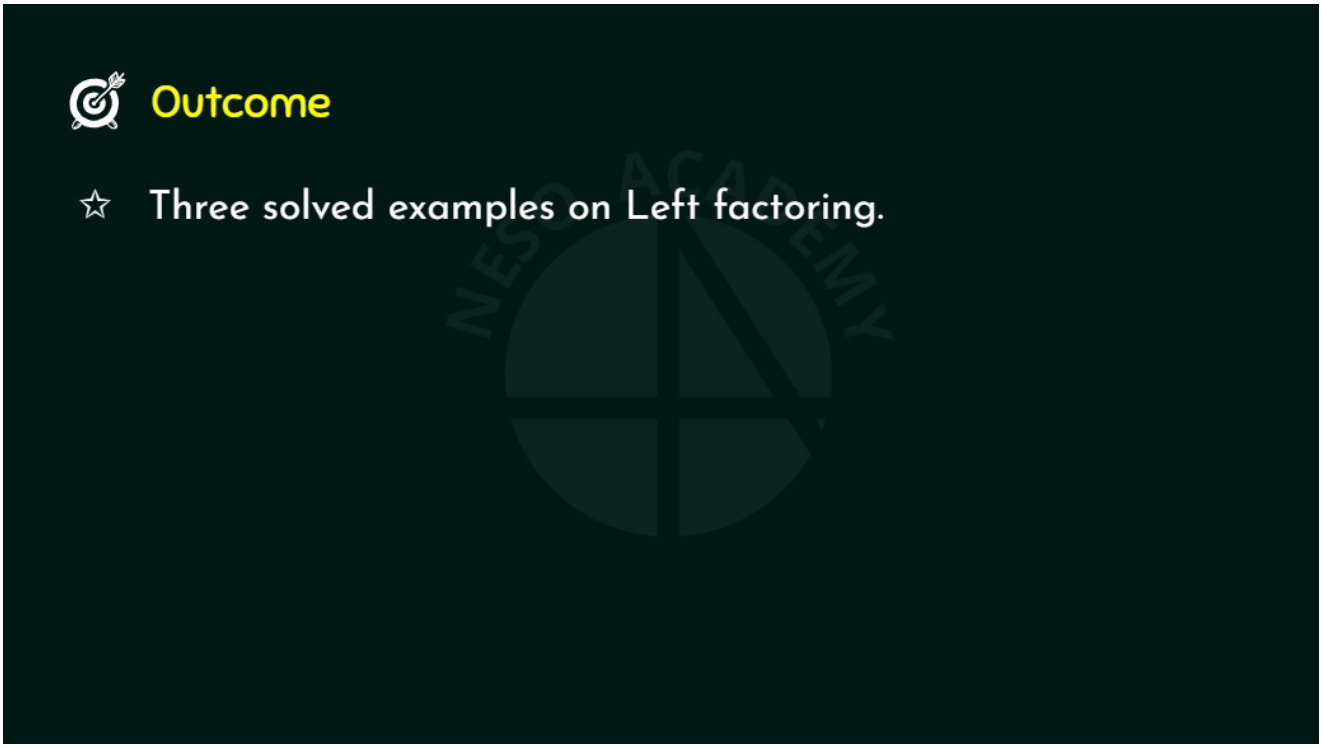


Determinism vs Ambiguity:S → iEtSS' I aS' → ε I eSE → bS → iEtS I iEtSeS I aE → bSi E t Si E t S e SSi E t Si E t S e SSi E t S S'i E t S S'εSi E t S S'i E t S S'e Sεe SiEtiEtSeS

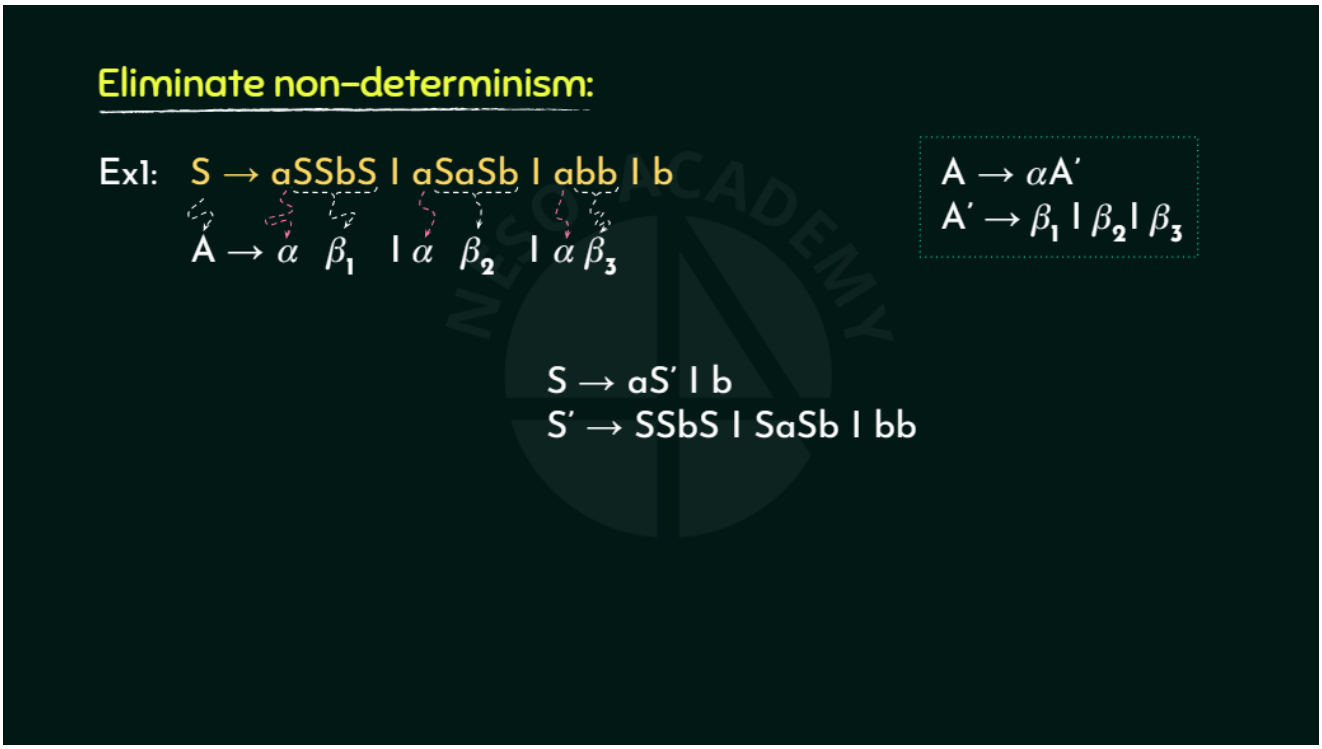


S → iEtS I iEtSeS I aE → bSi E t S e Si E t SSi E t Si E t S e SiEtiEtSeSif(E){if(E){ S }else{ S }}Reason for Ambiguity:

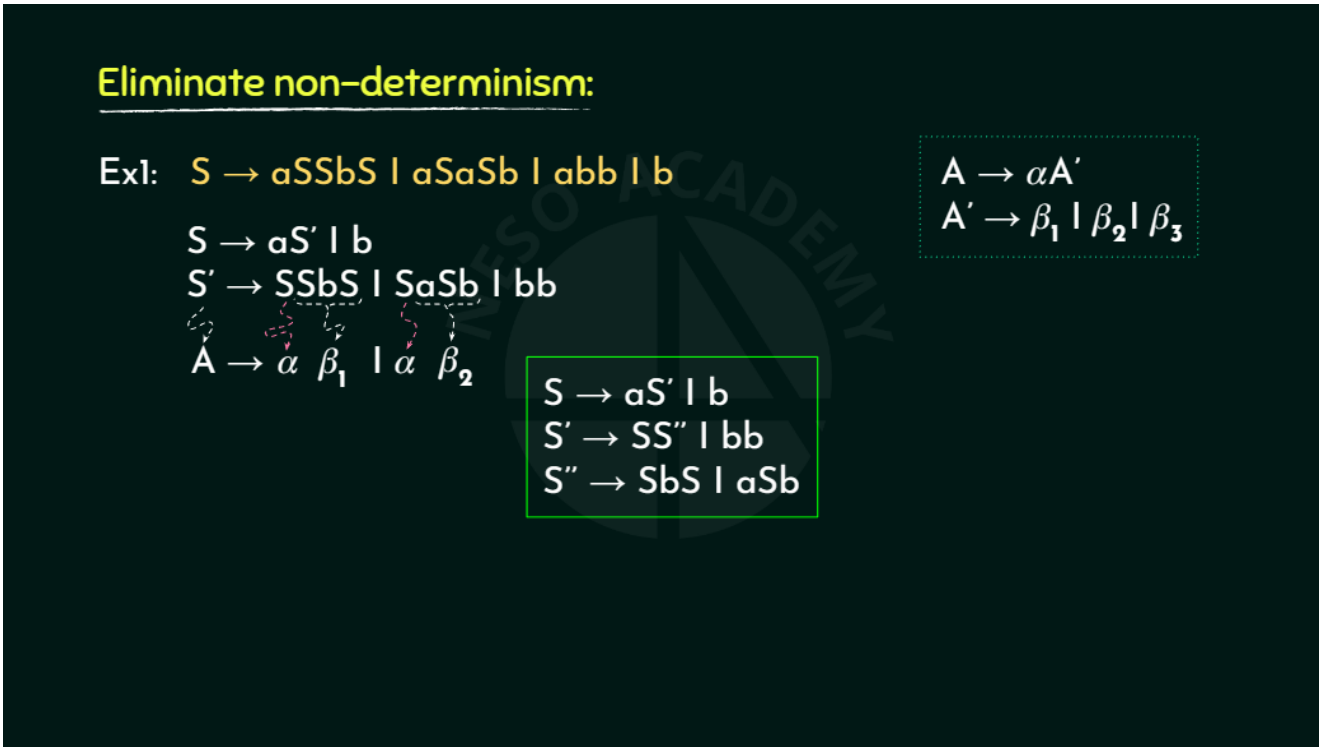
Compiler DesignNon-deterministic CFGs - Solved Problems (Set 2)



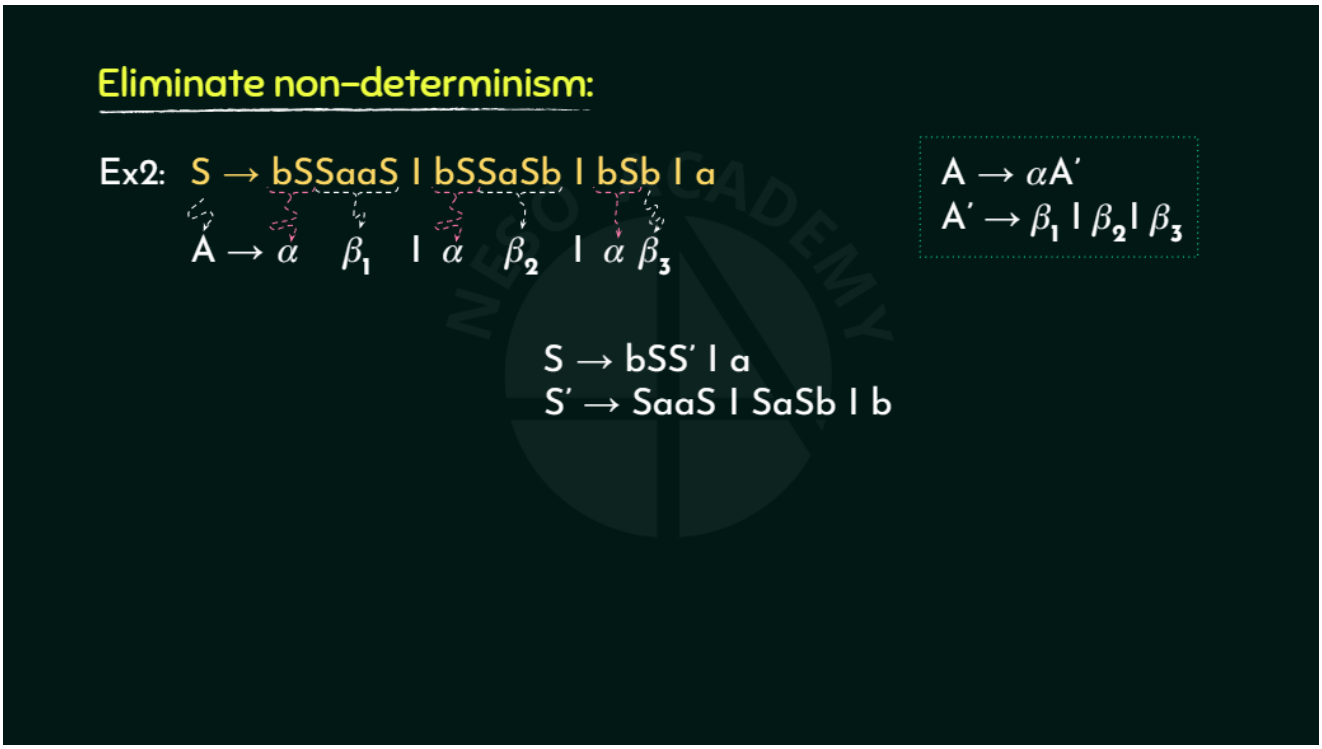
Outcome☆Three solved examples on Left factoring.



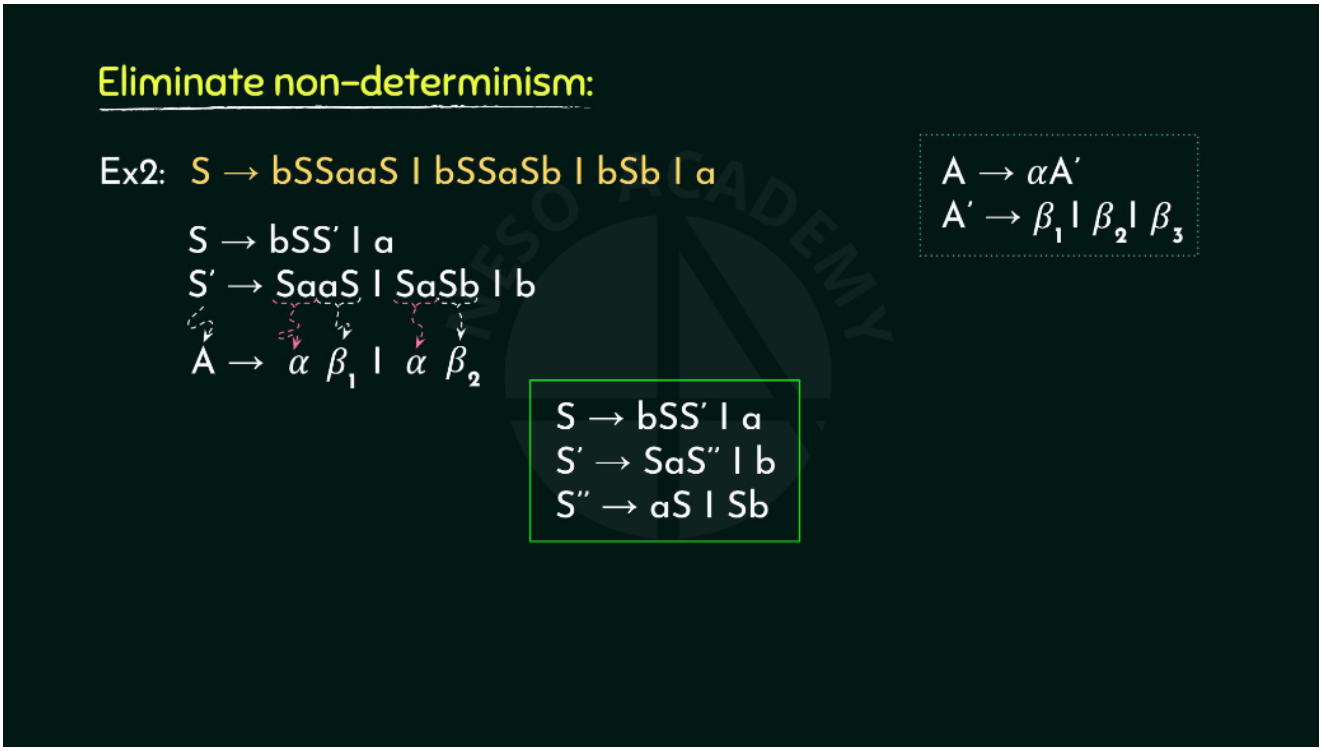
S → aSSbS I aSaSb I abb I bEx1:Eliminate non-determinism:→AIαIααβ1β2β3I β2β1A→αA'→A'I β3S→aS'I bS'→ SSbSI SaSbI bb

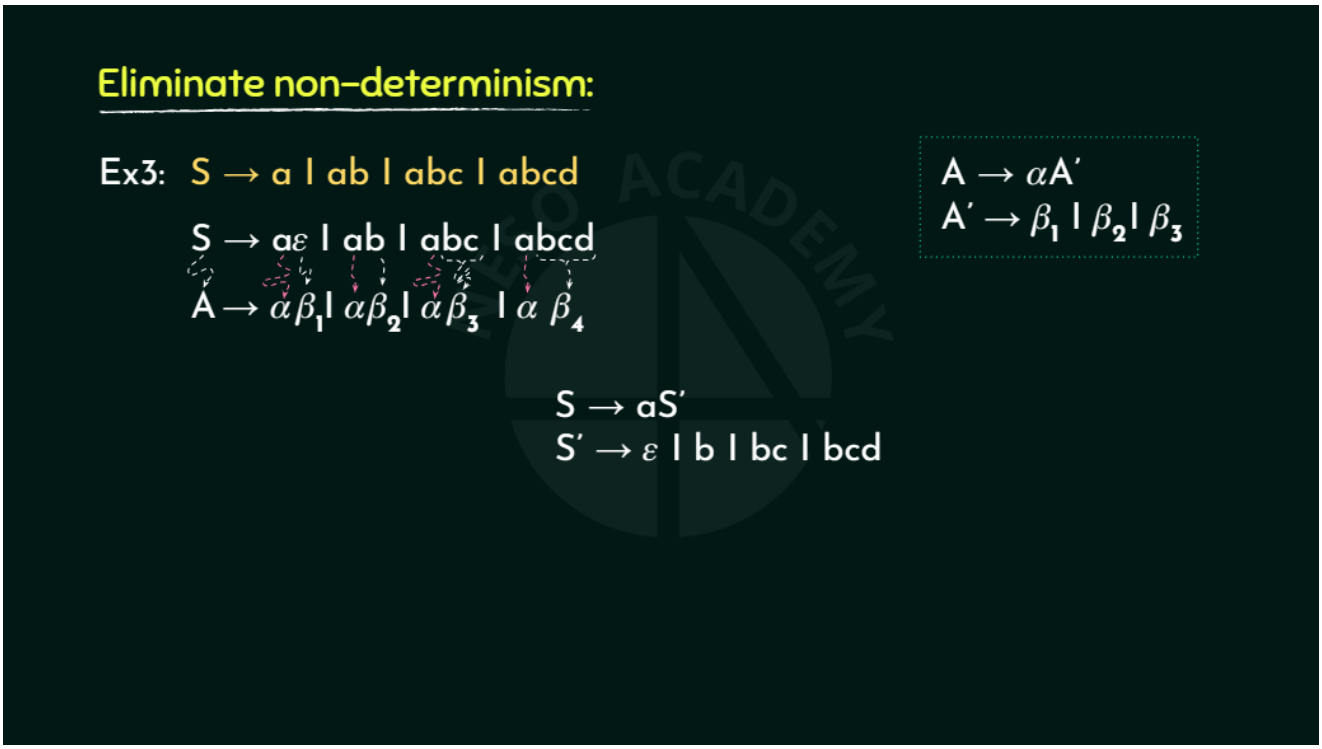


S → aS' I bS' → SS'' I bbS'' → SbS I aSbS → aS' I bS' → SSbS I SaSb I bbS → aSSbS I aSaSb I abb I bEx1:Eliminate non-determinism:→Aαβ1β2I β2β1A→αA'→A'I β3Iα

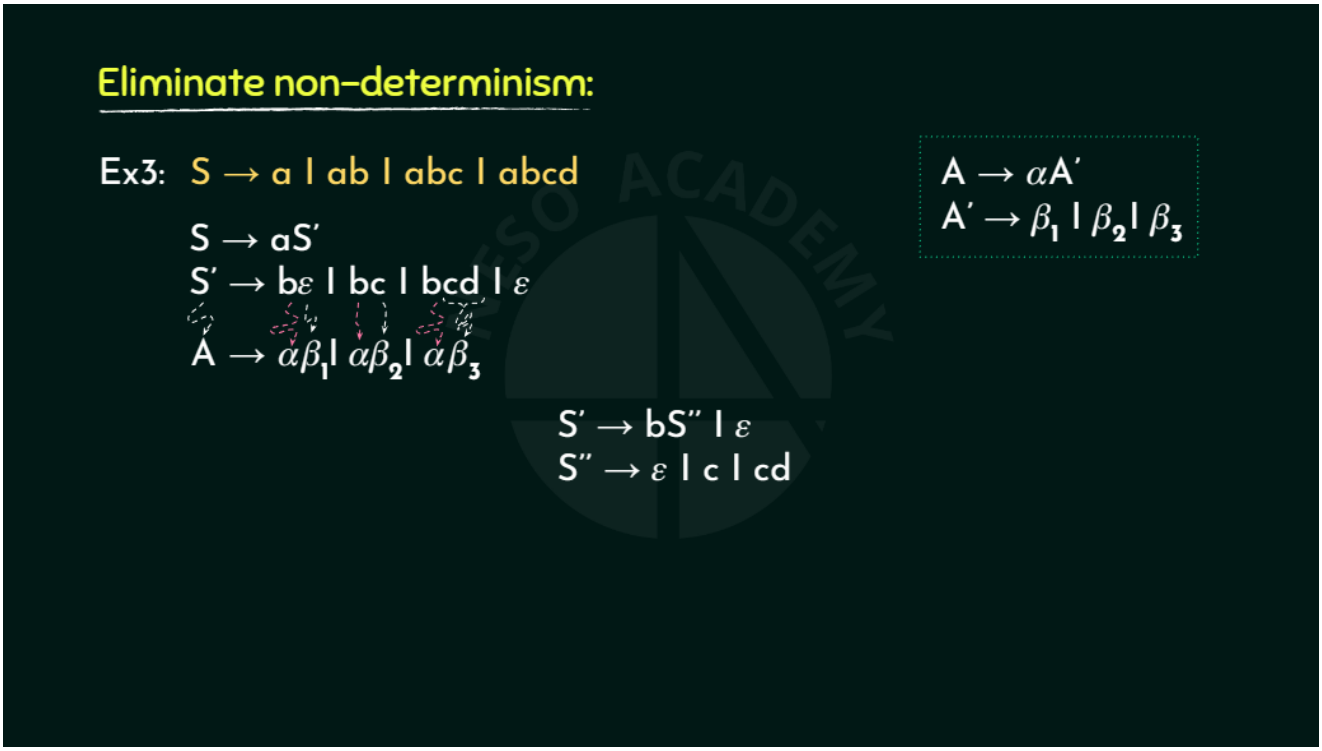


Ex2:Eliminate non-determinism:I β2β1A→αA'→A'I β3S → bSSaaS I bSSaSb I bSb I a→Aβ1β2β3αIαIαS→ bSS'I aS'→ SaaSI SaSbI b

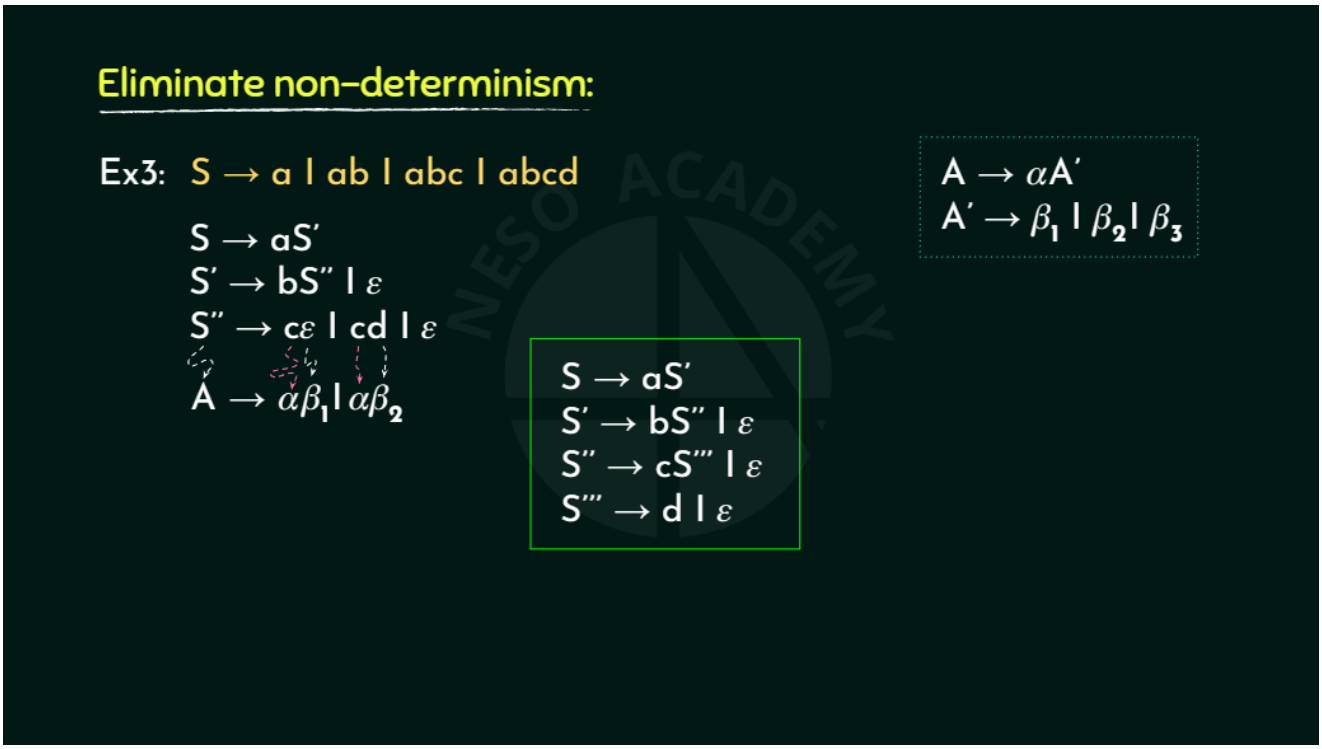




IααS → aε I ab I abc I abcdEx3:Eliminate non-determinism:I β2β1A→αA'→A'I β3S → a I ab I abc I abcd→Aβ1β2IαIαβ4β3SS'→ aS'→ εI b I bcI bcd



Ex3:Eliminate non-determinism:I β2β1A→αA'→A'I β3S → a I ab I abc I abcdS → aS' S' → bε I bc I bcd I εAαIαIα→β1β2β3S'→ bS''I εS''→ εI cI cd



S → aS'S' → bS'' I εS'' → cS''' I εS''' → d I εβ1Ex3:Eliminate non-determinism:I β2β1A→αA'→A'I β3S → a I ab I abc I abcdAαIα→β2S → aS'S' → bS'' I εS'' → cε I cd I ε