more than 1 LMD, 1 RMD, 1 Parse Tree → Ambiguous

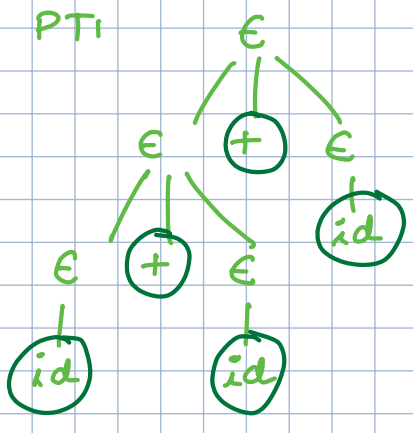
Conversion from Ambiguous Grammar to Unambiguous Grammar:

1). Associativity

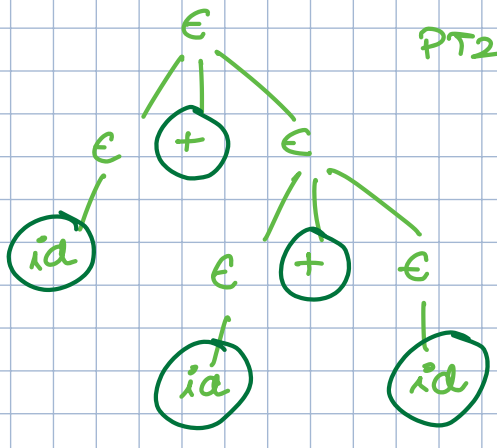
$$E \rightarrow E+E \mid E*E \mid id$$

String: id + id + id

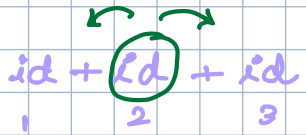
PT1



PT2



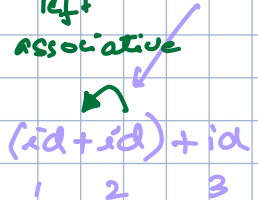
Ambiguous



id → operand

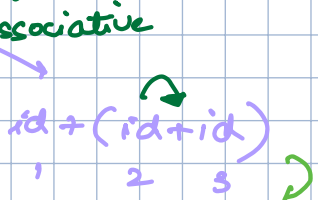
+ → operator

left associative



→ PT1

right associative

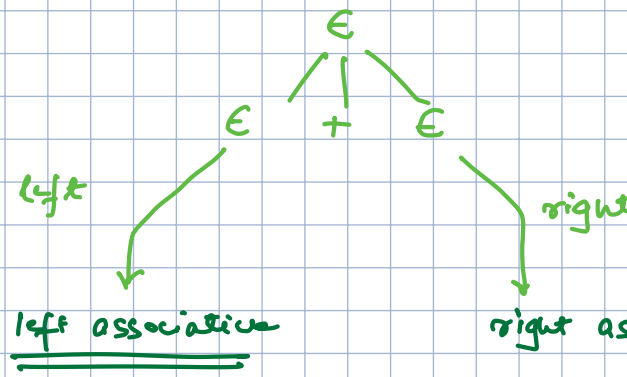


PT2

+ : Left Associativity

↳ Generate PT1

↳ shouldn't be able to generate PT2.



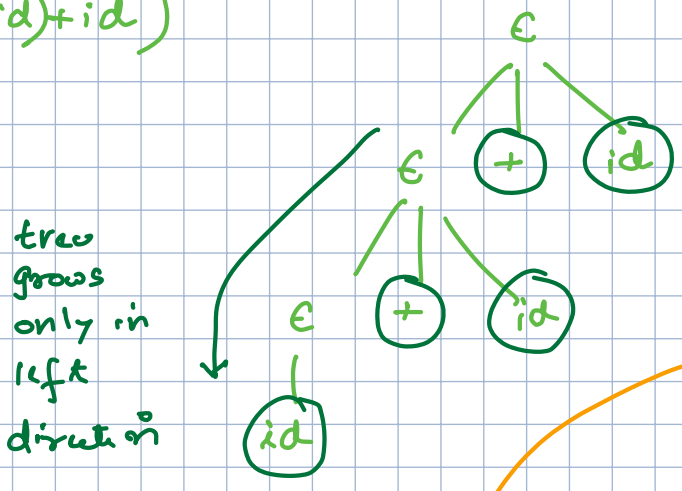
left associative

right associative

do something so that our tree grows only in the right direction.

$E \longrightarrow E + id \mid id$

((id + id) + id)



tree grows only in left direction

left most symbol of RHS = LHS

If grammar is left recursive then the operator will be left associative.

1 case where our grammar is ambiguous

↳ Rules of associativity are not defined properly.

$E \rightarrow E+E \mid id$

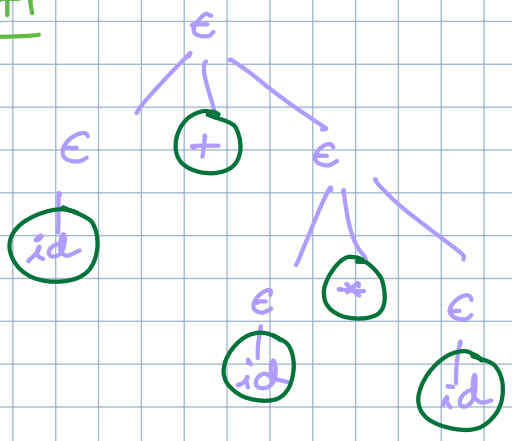
$E \rightarrow E+id \mid id$

2. Precedence

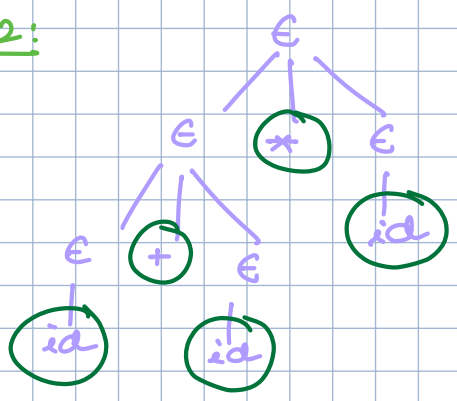
$E \rightarrow E+E \mid E*E \mid id$

String: id + id * id

PT1



PT2:



2 Parse Trees

↳ Grammar Ambiguous

2 + 3 * 4

2 + (3 * 4)

2 + 12

14

PT1 ✓

(2+3) * 4

5 * 4

20

PT2

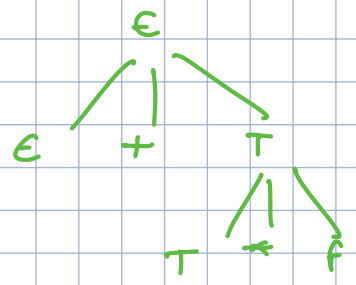
2 operators, higher precedence operator should be at a lower level, far away from the start symbol.

$E \rightarrow E+E \mid E*E \mid id$

$E \rightarrow E+T \mid T$ (first deriving +)

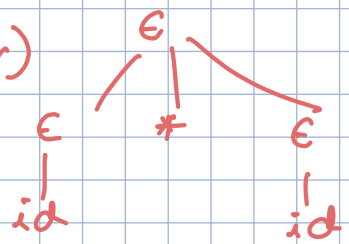
$T \rightarrow T*f \mid f$ (then derive *)

$f \rightarrow id$

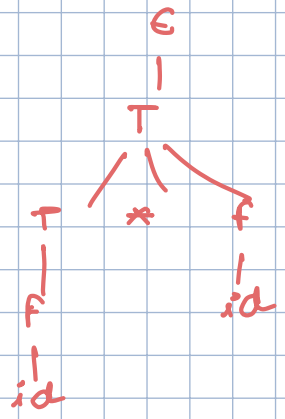


String which u were able to derive earlier, u should be able to derive the same strings now with the new grammar.

(old grammar)



(new grammar)



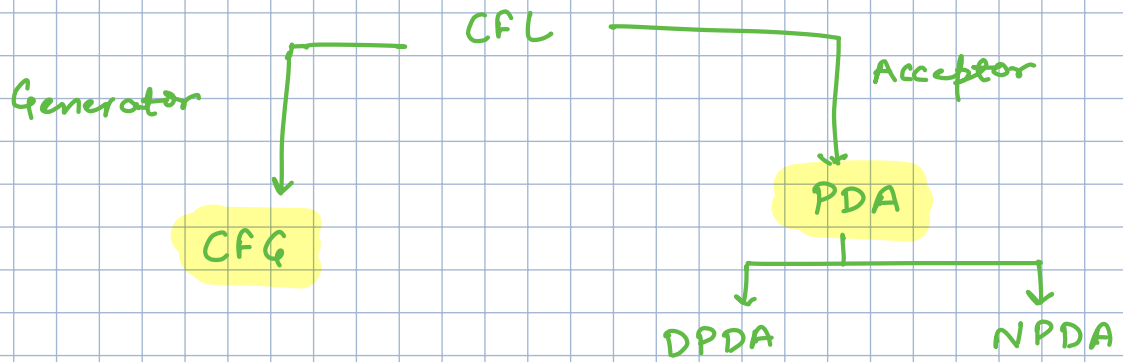
Grammar may be ambiguous bcz:

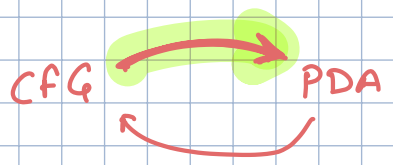
1) Associativity is not defined properly: left Recursive

2) Precedence Rules are not defined properly:

operator with higher precedence at lower level.

Equivalence of PDA & CFG:





Both are equivalent in power.

CFG to PDA:

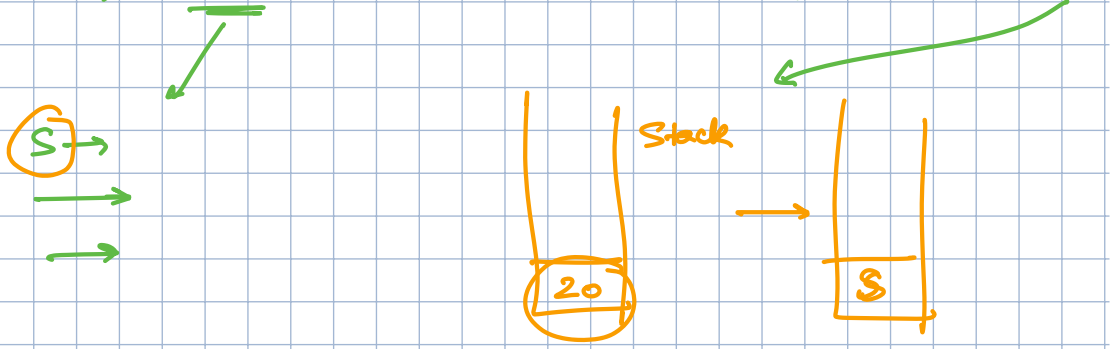
1. Convert CFG productions to GNF

$NT \rightarrow T$

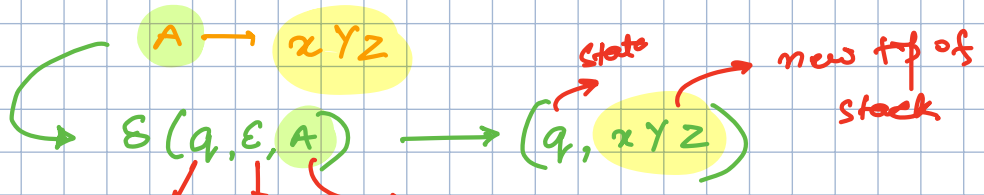
$NT \rightarrow T\,(NT)^*$

2. PDA will have only 1 state $\{q\}$

3. Start Symbol of CFG will be initial symbol in PDA



4. for non terminal symbols (variables), add the following rule



State Input Alphabet Top of Stack

5. for each terminal symbol, add the following rule:

'x'

$\delta(q, x, x) \rightarrow (q, \epsilon)$

state; state; Input alphabet; top of Stack; pop the element

Q: Construct a PDA equivalent to following CFG productions ?

$S \rightarrow aAA$

$A \rightarrow aS \mid bS \mid a$

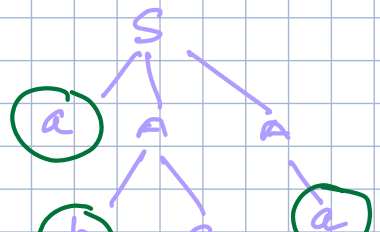
1. CFG → GNF : Already in GNF
2. $\{q\}$
3. S
4. $\delta(q, \epsilon, S) \rightarrow (q, aAA)$

   $\delta(q, \epsilon, A) \rightarrow (q, aS) \mid (q, bS) \mid (q, a)$
5. $\delta(q, a, a) \rightarrow (q, \epsilon)$

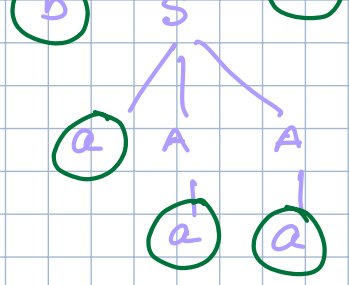
   $\delta(q, b, b) \rightarrow (q, \epsilon)$

$S \rightarrow aAA$

$A \rightarrow aS \mid bS \mid a$



If u are able to accept abaaaa string with the PDA transitions ?



$\delta(q, abaaaa, S)$ — (state, input string, symbol on stack)

$\delta(q, \underline{a}baaaa, \underline{a}AA)$

$\delta(q, baaaa, \underline{A}A)$

$\delta(q, \underline{b}aaaa, \underline{b}SA)$

$\delta(q, aaaa, \underline{S}A)$

$\delta(q, \underline{a}aaa, \underline{a}AAA)$

$\delta(q, aaa, AAA)$

$\delta(q, \underline{a}aa, \underline{a}AA)$

$\delta(q, aa, AA)$

$\delta(q, \underline{a}a, \underline{a}A)$

$\delta(q, a, A)$

$\delta(q, \underline{a}, \underline{a})$

$\delta(q, \varepsilon)$ Accepted

abaaaa