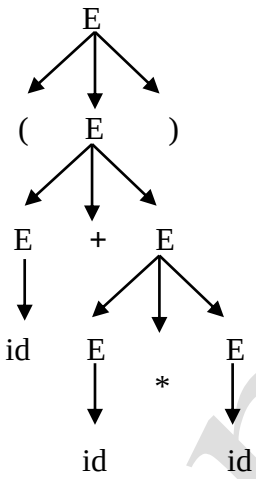
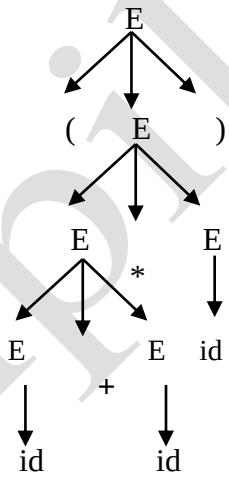
Example (2):-

Parse and derivations, this sentence (id+id*id) by using this grammar:
E→ E+E | E-E | E*E | E/E | (E) | -E | id

Sol 1:-
E→ (E)
E→ (E+**E**)
E→ (E+E*E)
E→ (id+E*E)
E→ (id+id*E)
E→ (id+id*id)

Sol 2:-
E→ (E)
E→ (**E***E)
E→ (E+E*E)
E→ (id+E*E)
E→ (id+id*E)
E→ (id+id*id)





H.W

E→ E+E | E-E | E*E | id
Parse this sentence: id+id*id

2.6.2 Ambiguity: (problems of grammar)

A grammar that produces more than one parse tree for some sentence is said to be ambiguous. An ambiguous grammar is one that produces more than one leftmost or more than one right most derivation for the same sentence. For certain types of parsers, it is desirable that the grammar be made unambiguous, for if it is not, we cannot uniquely determine which parse tree to select for a sentence.

وهي القواعد التي تنتج اكثر من شجرة اعراب واحده لبعض الجمل ويقال عنها غامضة. القواعد الغامضة هو وجود اكثر من اعراب واحد(اشتقاق) لبعض الجمل من اليمين ومن اليسار وتدعى غامضة.

Ex(1):-
S→ AB | aaB
A→ a | Aa
B→ b
W= aab

Sol (1):-

S→ aaB
S→ aab

Sol (2):-
S→ AB
S→ AaB
S→ aaB
S→aab

Ex (2):-
S→ A
A→ aA | Aa |a

Sol:-
S→ A
S→ aA
S→ aAa
S→ aaa
OR
S→ A
S→ Aa
S→ Aaa
S→ aaa

Sol(2):-
S→ A
S→ aA
S→ aaA
S→ aaa

https://www.youtube.com/watch?v=PnOCNxckV48

H.W
Check this sentence Ambiguity or unambiguous id + id + id
E→ E+E | E-E | E*E | id

2.6.3 Left Recursion

A grammar is left recursion if it has a non-terminal A, such that there is a derivation A → αA For some string α. Top-down parsing methods cannot handle left recursion grammars, so a transformation that eliminates left recursion is needed.

Left Recursion: ويقصد به كل حرف يودي الى نفسه.

E→ E+T

T→ M| T*F

كيف يمكن التخلص من مشكلة
Left Recursion

توجد بعض الخطوات المهمة يجب اتباعها عند الحل وهي:

- يتم اخفاء العبارة المشكلة وانزال ماتبقى ويربط مع حرف ماقبل السهم او غيره ووضع علامة البرايم ¯
- العبارة المشكلة يحذف منها حرف المشكلة وينزل ماتبقى ويربط مع حرف ماقبل السهم او غيره ووضع علامة البرايم ¯ وثو نضيف علامة الـ λ كحرف جديد.

Example: Consider the following grammar for arithmetic expressions.

E→ E+T |T
T→ T*F|F
F→ (E) | id

Eliminating the immediate left recursion (productions of the form A→ Aα) to the production for E and then for T, we will get

E→ T$\bar{E}$
$\bar{E}$→ +T $\bar{E}$| λ
T→ F$\bar{T}$
$\bar{T}$→ *F$\bar{T}$ | λ
F→ (E) | id

Example:-

A→ ABx | Aa| a

Sol:

A→ ABx | Aa | a
A→ a$\bar{A}$
$\bar{A}$→ Bx$\bar{A}$ | a$\bar{A}$ | λ

Example:-

A→ Ac | Aad | ba | c

Sol:
A→ Ac | Aad | ba | c
A→ bdĀ | cĀ
Ā→ cĀ | adĀ | λ

https://www.youtube.com/watch?v=FJsVmfPExLs

2.6.4 Left Factoring

Left factoring is a grammar transformation that is useful for producing a grammar suitable for predictive parsing. The basic idea is that when it is not clear which of two alternative productions to use to expand a nonterminal A, we may be able to rewrite the A-productions to defer the decision until we have seen enough of the input to make the right choice.

Left Factoring: هو ايجاد العامل المشترك والتخلص منه لتكون العباره اكثر وضوح.

Example:-
E→ abc | ab | k

Sol:
E→ abc | ab | k

- في الخطوة الاولى نجد الاحرف المشتركة في كل العبارات
- ثم نكتب الحروف المشتركة على شكل كلمة واحدة ومن ثم يربط مع حرف ماقبل السهم او غيره ووضع علامة البرايم ، ونكتب ماتبقى من الجملة الاصلية.
- ثم نكتب الحروف المتبقية من الكلمات المستخرج منها الحروف المشتركة. اما اذا لم يتبقي شيْ من الكلمة المستخرج منه الحروف المشتركة فنكتب رمز الـ λ .

E→ abĒ | k
E→ c | λ

Example:-
S→ abMna | ckL | abMo

Sol:
S→ abMna | ckL | abMo
S → abMS̄ | ckL
S→ na | o

Example:-
A→ xB | xY
Sol:
A→ xB | xY
A→ xĀ
Ā→ B | Y

Example:-
A→ aAB | aA | a

Sol:
A→ aAB | aA | a
A→ aĀ
Ā→ AB | A | λ

https://www.youtube.com/watch?v=whe29gOb8p4

2.6.5 Predictive Parsing Method Top down احد طرق

In many cases, by carefully writing a grammar eliminating left recursion from it, and left factoring the resulting grammar, we can obtain a grammar that can be parsed by a non-backtracking predictive parser. We can build a predictive parser by maintaining a stack. The key problem during predictive parser is that of determining the production to be applied for a nonterminal. The non-recursive parser looks up the production to be applied in a parsing table.

A table-driven predictive parser has an input buffer, a stack, a parsing table, and an output stream. The input buffer contains the string to be parsed, followed by $, (a symbol used as a right endmarker to indicate the end of the input string). The stack contains a sequence of grammar symbols with $ on the bottom,(indicating the bottom of the stack). Initially, the stack contains the start symbol of the grammar on the top of $.

- قبل البدء بهذه الطريقه يجب ان يكون الـ C.F.G خالية من **Left Recursion& Left Factoring**

Example:

E→ E+T | T
T → T*F | F
F → (E) | id

Parse the input id + id * id by using predictive parsing: