Allenhouse Institute of Technology (AKTU Code: 505)
Rooma, Kanpur – 208 008

Digital Notes

[Department of Computer Science & Engineering]

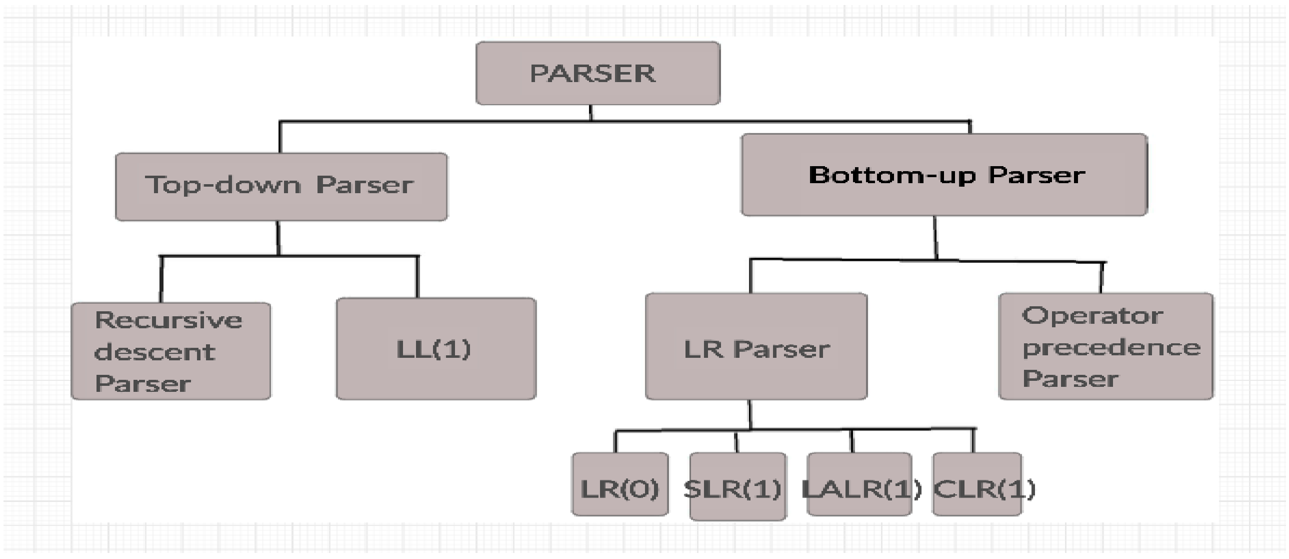
Subject Name	**:**	**Compiler Design**
Subject Code	**:**	**KCS 502**
Course	**:**	**B.TECH**
Branch	**:**	**CS**
Semester	**:**	**5th**
Prepared by	**:**	**Mr. Yogendra Singh**

UNIT-2: Basic Parsing Techniques:

Parsers:

Parser is that phase of the compiler which takes token string as input and with the help of existing grammar, converts it into the corresponding parse tree. Parser is also known as Syntax Analyzer.

Classification of Parsers:



Top down Parser (TDP):

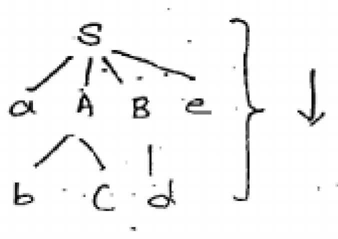
The process of construction of parse tree starting from root and proceed to children is called TDP i.e. starting from start symbol of the grammar and reaching the input string.

Ex: $S \rightarrow aABe$

$A \rightarrow bc$

$B \rightarrow d$

$w = abcde$



Notes:

- TDP internally uses left most derivation (LMD).
- TDP constructed for the grammar if it is free from ambiguity and left recursion.

Drawback of Top-Down Parsing:

Backtracking was the major drawback of top-down parsing.

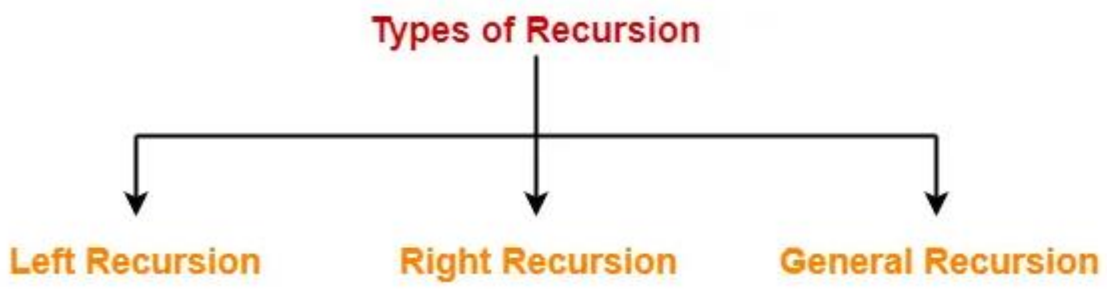
Classification of TOP down Parser:

Top down Parser are of 2 types

- **Recursive descent parser**
  Recursive descent is a top-down parsing technique that constructs the parse tree from the top and the input is read from left to right.
- **Non-recursive descent parser**
  It is also known as LL(1) parser or predictive parser or without backtracking parser or dynamic parser. It uses a parsing table to generate the parse tree instead of backtracking.

Recursion:

Recursion can be classified into following three types-



Left Recursion-

- A production of grammar is said to have left recursion if the leftmost variable of its RHS is same as variable of its LHS.
- A grammar containing a production having left recursion is called as Left Recursive Grammar.

Example:

$$A \rightarrow A\alpha / \beta$$

After elimination of left recursion

$$\mathbf{A \rightarrow \beta A'}$$

$$A' \rightarrow \alpha A' \,/\, \in$$

Right Recursion-

- A production of grammar is said to have right recursion if the rightmost variable of its RHS is same as variable of its LHS.
- A grammar containing a production having right recursion is called as Right Recursive Grammar.

Example-

$$S \rightarrow aS \,/\, \in$$

Notes:

Right recursion does not create any problem for the Top down parsers.

Therefore, there is no need of eliminating right recursion from the grammar.

3. General Recursion-

The recursion which is neither left recursion nor right recursion is called as general recursion

Example-

$$S \rightarrow aSb \,/\, \in$$

Problem-01:

Consider the following grammar and eliminate left recursion-

$$A \rightarrow ABd \,/\, Aa \,/\, a$$

$$B \rightarrow Be \,/\, b$$

Solution-

The grammar after eliminating left recursion is-

$A \rightarrow aA'$

$A' \rightarrow BdA' \,/\, aA' \,/\, \in$

$B \rightarrow bB'$

$B' \rightarrow eB' \,/\, \in$

Problem-02:

Consider the following grammar and eliminate left recursion-

E → E + E / E x E / a

Solution-

The grammar after eliminating left recursion is-

E → aA

A → +EA / xEA / ∈

Problem-03:

Consider the following grammar and eliminate left recursion-

E → E + T / T

T → T x F / F

F → id

Solution-

The grammar after eliminating left recursion is-

E → TE’

E’ → +TE’ / ∈

T → FT’

T’ → xFT’ / ∈

F → id

Problem-04:

Consider the following grammar and eliminate left recursion-

S → (L) / a

L → L , S / S

Solution- The grammar after eliminating left recursion is-

S → (L) / a

L → SL’

L’ →, SL’ / ∈

Problem-05:

Consider the following grammar and eliminate left recursion-

$$S \rightarrow S0S1S \; / \; 01$$

Solution-

The grammar after eliminating left recursion is-

$S \rightarrow 01A$

$A \rightarrow 0S1SA \; / \; \in$

Problem-06:

Consider the following grammar and eliminate left recursion-

$$S \rightarrow A$$
$$A \rightarrow Ad \; / \; Ae \; / \; aB \; / \; ac$$
$$B \rightarrow bBc \; / \; f$$

Solution-

The grammar after eliminating left recursion is-

$S \rightarrow A$

$A \rightarrow aBA' \; / \; acA'$

$A' \rightarrow dA' \; / \; eA' \; / \; \in$

$B \rightarrow bBc \; / \; f$

Problem-07:

Consider the following grammar and eliminate left recursion-

$$A \rightarrow AA\alpha \; / \; \beta$$

Solution-

The grammar after eliminating left recursion is-

$A \rightarrow \beta A'$

$A' \rightarrow A\alpha A' \; / \; \in$

Problem-08:

Consider the following grammar and eliminate left recursion-

$$A \rightarrow Ba \; / \; Aa \; / \; c$$

$$B \rightarrow Bb \; / \; Ab \; / \; d$$

Solution-

This is a case of indirect left recursion.

Step-01:

First let us eliminate left recursion from A → Ba / Aa / c

Eliminating left recursion from here, we get-

A → BaA’ / cA’

A’ → aA’ / ∈

Now, given grammar becomes-

A → BaA’ / cA’

A’ → aA’ / ∈

B → Bb / Ab / d

Step-02:

Substituting the productions of A in B → Ab, we get the following grammar-

A → BaA’ / cA’

A’ → aA’ / ∈

B → Bb / BaA’b / cA’b / d

Step-03:

Now, eliminating left recursion from the productions of B, we get the following grammar-

A → BaA’ / cA’

A’ → aA’ / ∈

B → cA’bB’ / dB’

B’ → bB’ / aA’bB’ / ∈

This is the final grammar after eliminating left recursion.

Problem-09:

Consider the following grammar and eliminate left recursion-

X → XSb / Sa / b

S → Sb / Xa / a

Solution-

This is a case of indirect left recursion.

Step-01:

First let us eliminate left recursion from X → XSb / Sa / b

Eliminating left recursion from here, we get-

X → SaX’ / bX’

X’ → SbX’ / ∈

Now, given grammar becomes-

X → SaX’ / bX’

X’ → SbX’ / ∈

S → Sb / Xa / a

Step-02:

Substituting the productions of X in S → Xa, we get the following grammar-

X → SaX’ / bX’

X’ → SbX’ / ∈

S → Sb / SaX’a / bX’a / a

Step-03:

Now, eliminating left recursion from the productions of S, we get the following grammar-

X → SaX’ / bX’

X’ → SbX’ / ∈

S → bX’aS’ / aS’

S’ → bS’ / aX’aS’ / ∈

This is the final grammar after eliminating left recursion.

Problem-10:

Consider the following grammar and eliminate left recursion-

S → Aa / b

A → Ac / Sd / ∈

Solution-

This is a case of indirect left recursion.

Step-01:

First let us eliminate left recursion from S → Aa / b

This is already free from left recursion.

Step-02:

Substituting the productions of S in A → Sd, we get the following grammar-

S → Aa / b

A → Ac / Aad / bd / ∈

Step-03:

Now, eliminating left recursion from the productions of A, we get the following grammar-

S → Aa / b

A → bdA’ / A’

A’ → cA’ / adA’ / ∈

This is the final grammar after eliminating left recursion.

Non-Deterministic Grammar:

The grammar with common prefix is called non-deterministic

$$A \rightarrow \alpha\beta_1 / \alpha\beta_2 / \alpha\beta_3$$

Notes:

- The grammar with common prefixes requires backtracking
- Backtracking is costly and time consuming
- To avoid the backtracking, we need to convert the non-deterministic grammar into deterministic. So we need to perform left factoring.

Left Factoring-

The process of conversion of grammar with common prefix into deterministic grammar is called left factoring

Example: Suppose the grammar is in the form:

A ⇒ αβ1 | αβ2 | αβ3 | …… | αβn |

Where A is a non-terminal and α is the common prefix.

A ⇒ αA`

A` ⇒ β1 | β2 | β3 | …… | βn

Example 01:

S → iEtS / iEtSeS / a

E → b

Solution-

The left factored grammar is-

S → iEtSS’ / a

S’ → ∈ /eS

E → b

Example 02:

A → aAB / aBc / aAc

Solution-

Step-01:

A → aA’

A’ → AB / Bc / Ac

Again, this is a grammar with common prefixes.

Step-02:

A → aA’

A’ → AD / Bc

D → B / c

This is a left factored grammar.

<u>Problem-03:</u>

Do left factoring in the following grammar-

$$S \rightarrow bSSaaS \,/\, bSSaSb \,/\, bSb \,/$$

<u>Solution-</u>

<u>Step-01:</u>

$$S \rightarrow bSS' \,/\, a$$

$$S' \rightarrow SaaS \,/\, SaSb \,/\, b$$

Again, this is a grammar with common prefixes.

<u>Step-02:</u>

$$S \rightarrow bSS' \,/\, a$$

$$S' \rightarrow SaA \,/\, b$$

$$A \rightarrow aS \,/\, Sb$$

This is a left factored grammar.

<u>Problem-04:</u>

Do left factoring in the following grammar-

$$S \rightarrow aSSbS \,/\, aSaSb \,/\, abb \,/\, b$$

<u>Solution-</u>

<u>Step-01:</u>

$$S \rightarrow aS' \,/\, b$$

$$S' \rightarrow SSbS \,/\, SaSb \,/\, bb$$

Again, this is a grammar with common prefixes.

<u>Step-02:</u>

$$S \rightarrow aS' \,/\, b$$

$$S' \rightarrow SA \,/\, bb$$

$$A \rightarrow SbS \,/\, aSb$$

This is a left factored grammar

<u>Problem-05:</u>

Do left factoring in the following grammar-

$$S \rightarrow a \,/\, ab \,/\, abc \,/\, abcd$$

Solution-

Step-01:

$$S \rightarrow aS'$$

$$S' \rightarrow b \,/\, bc \,/\, bcd \,/\, \epsilon$$

Again, this is a grammar with common prefixes.

Step-02:

$$S \rightarrow aS'$$

$$S' \rightarrow bA \,/\, \epsilon$$

$$A \rightarrow c \,/\, cd \,/\, \epsilon$$

Again, this is a grammar with common prefixes.

Step-03:

$$S \rightarrow aS'$$

$$S' \rightarrow bA \,/\, \epsilon$$

$$A \rightarrow cB \,/\, \epsilon$$

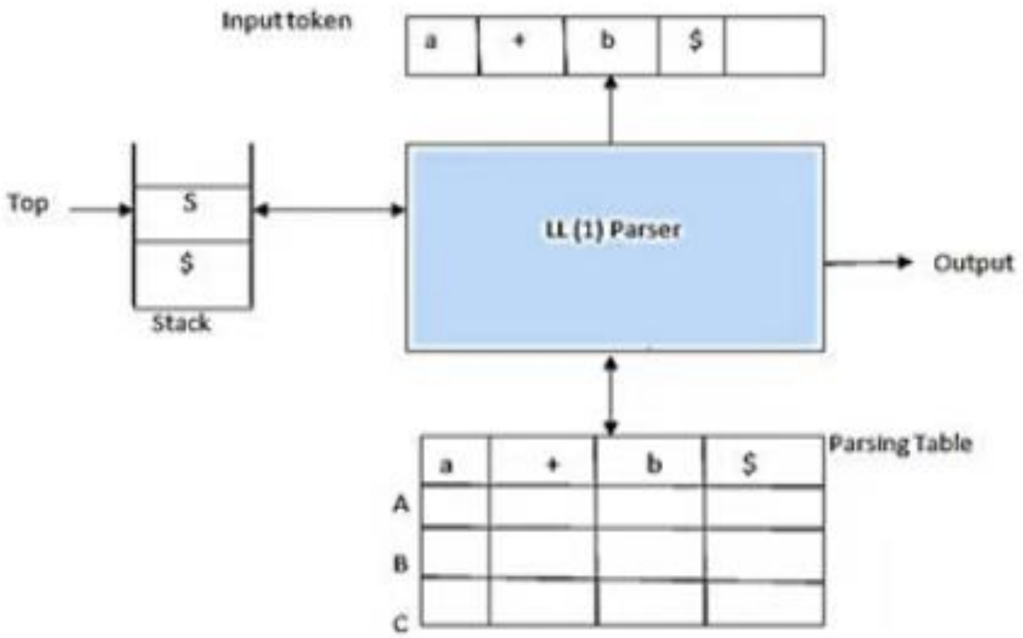
$$B \rightarrow d \,/\, \epsilon$$

This is a left factored grammar

LL (1) Parse /Table driven Parse:-

- This top down parsing algorithm is of Non-recursive type.
- Table is build in this type of parsing
- It is also called LL (1) parser because
    - The first L indicates that the input is read from left to right.
    - The second L says that it produces a left-to-right derivation.
    - And the 1 says that it uses one look ahead token.

Block Diagram of LL (1) parser:



Fig(a):- Model for LL(1) Parser

Data structure used by LL (1) Parser:-

1. Input Buffer
2. Stack
3. Parsing table

Input: This contains a string that will be parsed with the end-marker $.

Stack: A predictive parser sustains a stack. It is a collection of grammar symbols with the dollar sign ($) at the bottom.

Parsing table: M [A, S] is a two-dimensional array, where A is a non-terminal, and S is a terminal. With the entries in this table, it becomes effortless for the top-down parser to choose the production to be applied.

LL(1) Parsing algorithm:

Let x is a grammar symbol (top of stack symbol) and a is the look-ahead symbol then

1. if (x=a=$) then the parsing is successful
2. if ((x=a) ≠ $), then pop out from the top of stack and increment the i/p pointer

3. if x is variable then see the entry M[x, a] and M[x, a] contain the production, x->uvw, then replace x by uvw in reverse order and continue the process.

How to construct predictive LL(1) parsing table:

The construction of predictive LL(1) parser is based on two important functions

1. FIRST
2. FOLLOW

FIRST function:

FIRST (A) is a set of terminal symbols that begin in string derived from A.

Example:

A-> abc | def | ghi

Then, we have

First(A)={a,d,g}

Rules;-

1. For a production rule $X \rightarrow \in$

   First(X) = { ∈ }

2. For any terminal symbol a

   First (a) = {a}

3. For a production rule $X \rightarrow Y_1Y_2Y_3$

Calculating First(X)

- If $\in \notin$ First(Y_1), then First(X) = First(Y_1)
- If $\in \in$ First(Y_1), then First(X) = { First(Y_1) – ∈ } ∪ First(Y_2Y_3)

Calculating First(Y_2Y_3)

- If $\in \notin$ First(Y_2), then First(Y_2Y_3) = First(Y_2)
- If $\in \in$ First(Y_2), then First(Y_2Y_3) = { First(Y_2) – ∈ } ∪ First(Y_3)

Example:

$X \rightarrow Y_1Y_2Y_3$

$Y_1 \rightarrow$ a | ||

$Y_2 \rightarrow$ b | ∈

$Y_3 \rightarrow c \mid \in$

Then, we have

First(X) = First (Y_1)

= {First (Y_1) – ∈ } ∪ First(Y_2Y_3)

= { a || First (Y_2) }

= {a || {First (Y_2) – ∈} ∪ First (Y_3)}

= {a, b, c, ∈}

Problem-01:

Calculate the first and follow functions for the given grammar-

S → aBDh

B → cC

C → bC / ∈

D → EF

E → g / ∈

F → f / ∈

Solution:

First Functions-

- First(S) = { a }
- First(B) = { c }
- First(C) = { b , ∈ }
- First(D) = { First(E) – ∈ } ∪ First(F) = { g , f , ∈ }
- First(E) = { g , ∈ }
- First(F) = { f , ∈ }

Problem-03:

Calculate the first and follow functions for the given grammar-

S → (L) / a

L → SL’

L’ → ,SL’ / ∈

Solution-

First Functions-

- First(S) = { (, a }
- First(L) = First(S) = { (, a }
- First(L’) = { , , ∈ }

Example:

Find first and follow function of the given grammar?

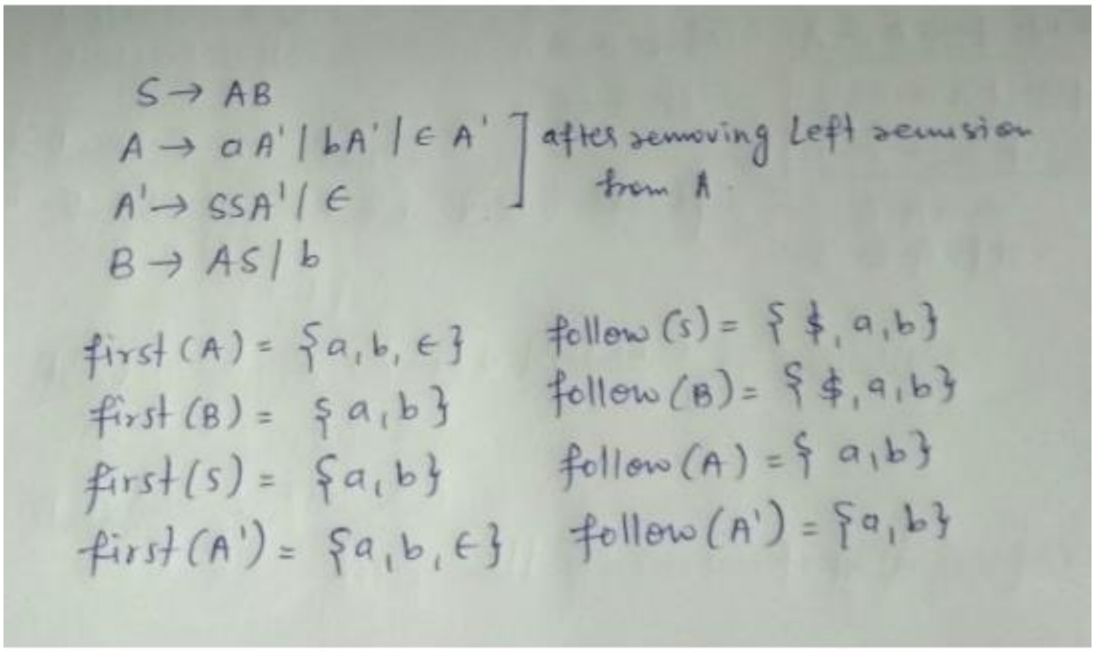
S->AB

A->BS/a/€

B->AS/b

Solution:

check this



S → AB

A → aA' | bA' | ∈ A'] after removing Left recursion from A

A' → SSA' | ∈

B → AS | b

first(A) = {a, b, ∈}   follow(S) = { $, a, b}

first(B) = {a, b}   follow(B) = { $, a, b}

first(S) = {a, b}   follow(A) = { a, b}

first(A') = {a, b, ∈}   follow(A') = {a, b}

Follow Function-

Follow (A) is a set of all terminal symbols that appear immediately to the right of A.

Rule-01: if S is start symbol, then Follow (S) = {$}

Rule-02: For any production, A → αB, then

Follow (B) = Follow (A)

Rule-03: For any production, A → αBβ,

- If ∈ ∉ First(β), then Follow(B) = First(β)
- If ∈ ∈ First(β), then Follow(B) = { First(β) – ∈ } ∪ Follow(A)

Example:

Calculate the follow functions for the given grammar-

S → A

A → aB / Ad

B → b

C → g

Solution-

We have-

- The given grammar is left recursive.
- So, we first remove left recursion from the given grammar.

After eliminating left recursion, we get the following grammar-

S → A

A → aBA’

A’ → dA’ / ∈

B → b

C → g

Follow function:

- Follow(S) = { $ }
- Follow(A) = Follow(S) = { $ }
- Follow(A’) = Follow(A) = { $ }
- Follow(B) = { First(A’) – ∈ } ∪ Follow(A) = { d , $ }

Problem-04:

Calculate the follow functions for the given grammar-

S → AaAb / BbBa

A → ∈

B → ∈

Solution:

Follow Functions-

- Follow(S) = { $ }
- Follow(A) = First(a) ∪ First(b) = { a , b }
- Follow(B) = First(b) ∪ First(a) = { a , b }

Problem-05:

Calculate the first and follow functions for the given grammar-

E → E + T / T

T → T x F / F

F → (E) / id

Solution-

- The given grammar is left recursive.
- So, we first remove left recursion from the given grammar.

After eliminating left recursion, we get the following grammar-

E → TE’
E’ → + TE’ / ∈
T → FT’
T’ → x FT’ / ∈
F → (E) / id

Follow Functions-

- Follow(E) = { $,) }
- Follow(E’) = Follow(E) = { $,) }

- Follow(T) = { First(E’) – ∈ } ∪ Follow(E) ∪ Follow(E’) = { + , $,) }
- Follow(T’) = Follow(T) = { + , $,) }
- Follow(F) = { First(T’) – ∈ } ∪ Follow(T) ∪ Follow(T’) = { x , + , $,) }

Problem-06:

Calculate the first and follow functions for the given grammar-

S → ACB / CbB / Ba

A → da / BC

B → g / ∈

C → h / ∈

Solution:

Follow Functions-

- Follow(S) = { $ }
- Follow(A) = { First(C) – ∈ } ∪ { First(B) – ∈ } ∪ Follow(S) = { h , g , $ }
- Follow(B) = Follow(S) ∪ First(a) ∪ { First(C) – ∈ } ∪ Follow(A) = { $, a , h , g }
- Follow(C) = { First(B) – ∈ } ∪ Follow(S) ∪ First(b) ∪ Follow(A) = { g , $, b , h }

Example of LL(1) parser

Example: Construct the parsing table for the following grammar

S → aABb

A → c / ∈

B → d / ∈

Step 1: No left recursion in the grammar

Step 2: Calculation of First set

First(S) = {a}
First (A) = {c, ∈}
First (B) = {d, ∈}

Step 3: Calculation of Follow set

Follow(S) = {$}

Follow(A) = First(B)={d,b}

Step 4: To construct LL(1) parsing table

	a	b	c	d	$
S	S→aABb				
A		A→€	A→c	A→€	
B		B→€		B→d	

Above LL(1) parsing table contain single entry in each cell. So that grammar is LL(1).

Example 2: **Construct the parsing table for the following grammar**

$$S \rightarrow AaAb \mid BbBa$$
$$A \rightarrow \in$$
$$B \rightarrow \in$$

Solution :

Step 1: No left recursion in the grammar

Step 2: Calculation of First set

First(S) = {a,b}
First (A) = {∈}
First (B) = {∈}

Step 3: Calculation of Follow set

Follow(S) = {$}

Follow(A) = {a,b}

Follow(B) = {a,b}

Step 4: To construct LL(1) parsing table

	a	b	$
S	S→AaAb	S→BbBa	
A	A→€	A→€	
B	B→€	B→€	

Above LL(1) parsing table contain single entry in each cell. So that grammar is LL(1).

Example 3:

E->TE'
E'->+TE'/ϵ
T->FT'
T'->*FT'/ϵ
F->id/(E)

Solution
Step 1: No left recursion in the given grammar

Step 2: find FIRST function

FIRST(E)={id,(}
FIRST(E')={+,ϵ}
FIRST(T)={id,(}
FIRST(T')={*,ϵ}
FIRST(F)={id,(}
Step3: find FOLLOW function

FOLLOW(E)={$,)}
FOLLOW(E')={$,)}
FOLLOW(T)={+,),$}
FOLLOW(T')={+,),$}
FOLLOW(F)={*,+,),$}

Step4: LL(1) Parsing Table:

	id	+	*	**(**	**)**	**$**
E	E->TE'			E->TE'		
E'		E'->+TE'			E'->ϵ	E'->ϵ
T	T->FT'			T->FT'		
T'		T'->ϵ	T'->*FT'		T'->ϵ	T'->ϵ
F	F->id			F->(E)		

Note: In each cell, there should be at max one entry. If there are more than one entry, then the grammar is not LL(1).