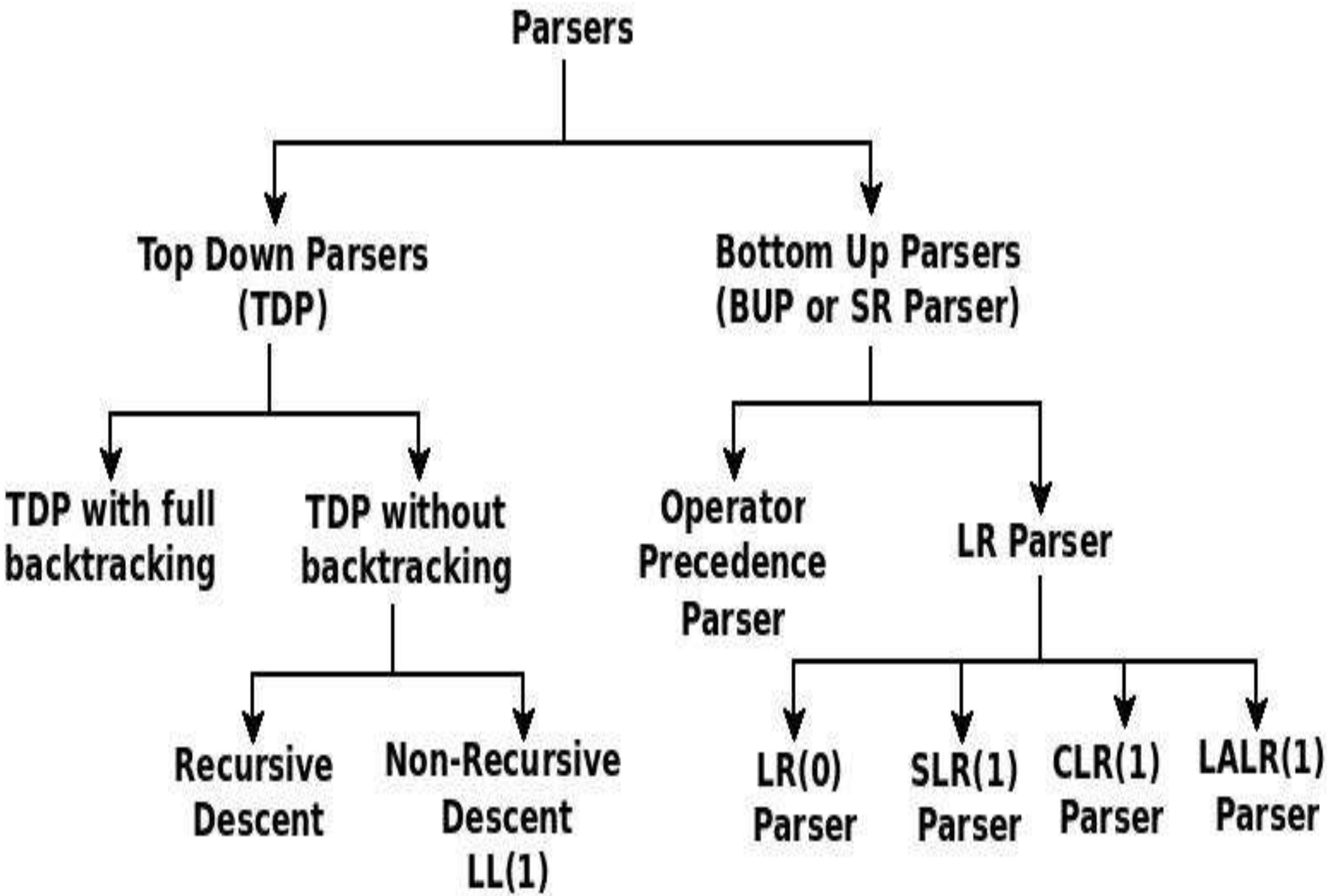
SYNTAX ANALYSIS OR PARSING


Parsers
Top Down Parsers
(TDP)
Bottom Up Parsers
(BUP or SR Parser)
TDP with full
backtracking
TDP without
backtracking
Operator
Precedence
Parser
LR Parser
Recursive
Descent
Non-Recursive
Descent
LL(1)
LR(0)
Parser
SLR(1)
Parser
CLR(1)
Parser
LALR(1)
Parser

TOP DOWN PARSING

- Find a left-most derivation
- Find (build) a parse tree
- Start building from the root and work down…
- As we search for a derivation
  - Must make choices:
    - Which rule to use
    - Where to use it

TOP-DOWN PARSING

- Recursive-Descent Parsing
  - Backtracking is needed (If a choice of a production rule does not work, we backtrack to try other alternatives.)
  - It is a general parsing technique, but not widely used.
  - Not efficient
- Predictive Parsing
  - no backtracking
  - efficient
  - needs a special form of grammars (LL(1) grammars).
  - Recursive Predictive Parsing is a special form of Recursive Descent parsing without backtracking.
  - Non-Recursive (Table Driven) Predictive Parser is also known as LL(1) parser.

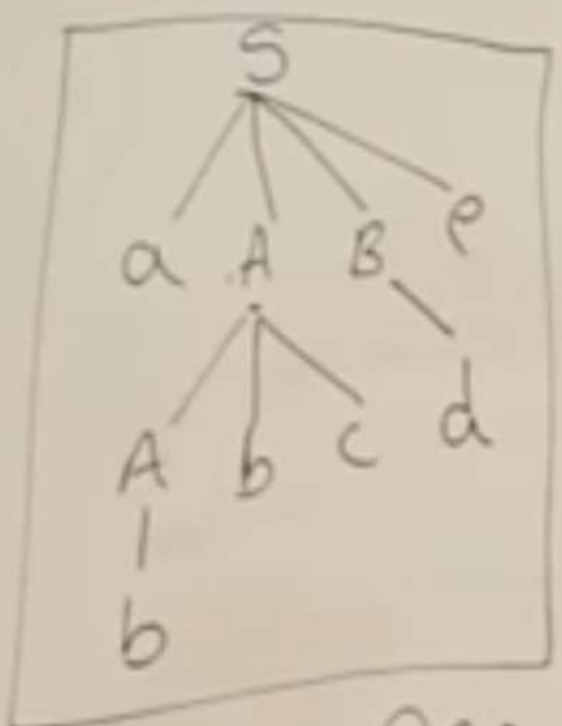
$S \to aABe$

$A \to Abc / b$

$B \to d$

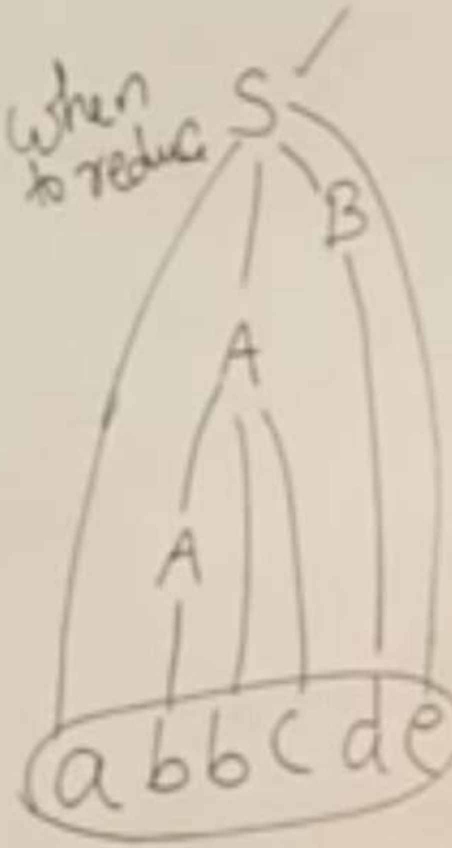
w: abbcde

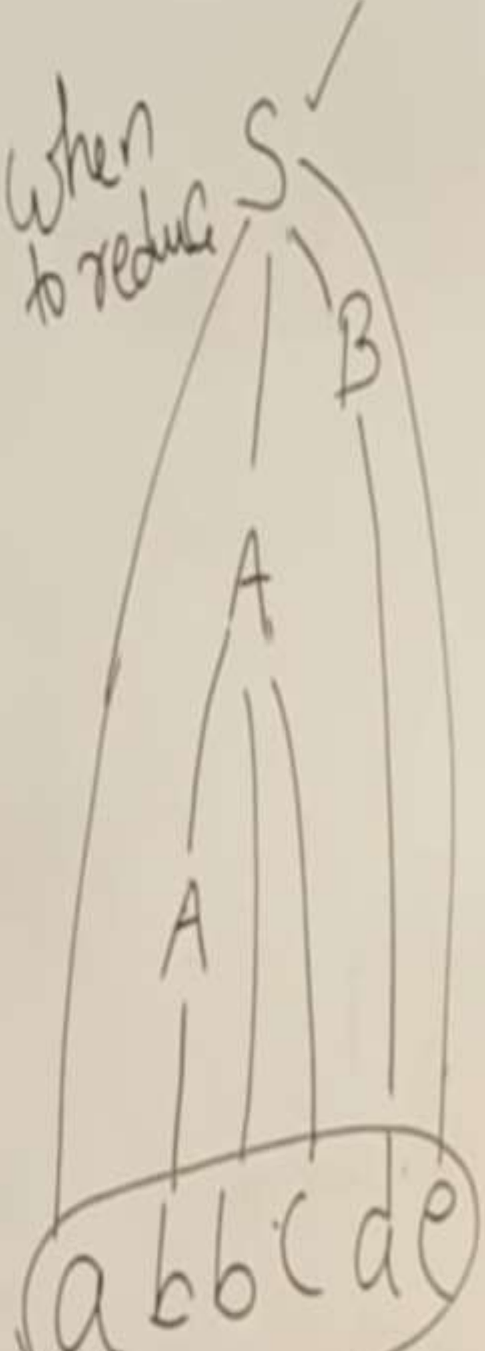
what to use



$$
\begin{aligned}
S &\Rightarrow aABe \\
&\Rightarrow aAbcBe \\
&\Rightarrow abbcBe \\
&\Rightarrow abbcde
\end{aligned}
$$

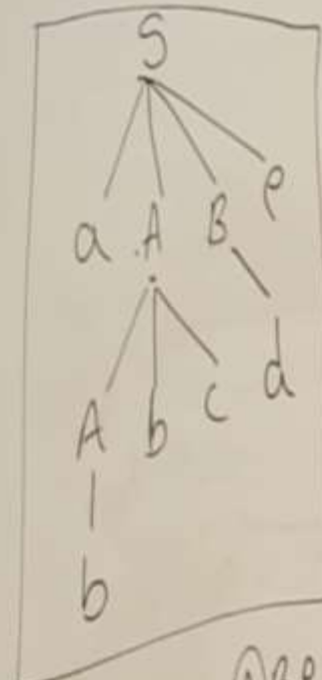
when to reduce



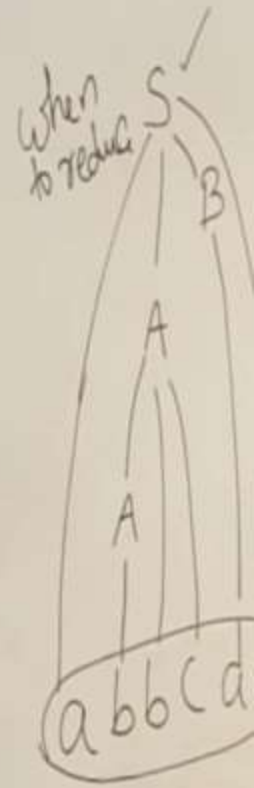


$$S \Rightarrow aABe$$
$$\Rightarrow aAde$$
$$\Rightarrow aAbcde$$
$$\Rightarrow abbcde$$

what to use



$$S \Rightarrow aABe$$
$$\Rightarrow aAbcBe$$
$$\Rightarrow abbcBe$$
$$\Rightarrow abbcde$$



RECURSIVE DESCENT PARSING (BACKTRACKING)

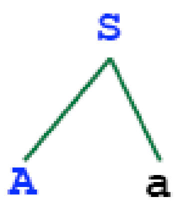
Input: `aabbde`

S

1. S → Aa
2. → Ce
3. A → aaB
4. → aaba
5. B → bbb
6. C → aaD
7. D → bbd

RECURSIVE DESCENT PARSING

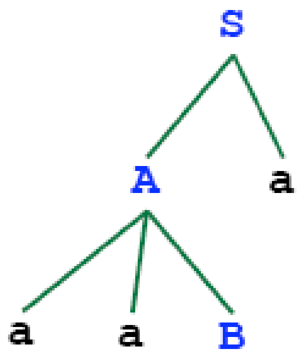
Input: `aabbde`



1.	S	$\rightarrow$	Aa
2.		$\rightarrow$	Ce
3.	A	$\rightarrow$	aaB
4.		$\rightarrow$	aaba
5.	B	$\rightarrow$	bbb
6.	C	$\rightarrow$	aaD
7.	D	$\rightarrow$	bbd

RECURSIVE DESCENT PARSING

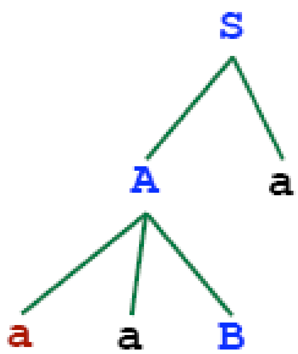
Input: `aabbde`



1. S → Aa
2. → Ce
3. A → aaB
4. → aaba
5. B → bbb
6. C → aaD
7. D → bbd

RECURSIVE DESCENT PARSING

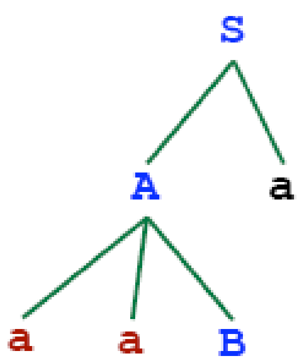
Input: `aabbde`



1. S → Aa
2. → Ce
3. A → aaB
4. → aaba
5. B → bbb
6. C → aaD
7. D → bbd

RECURSIVE DESCENT PARSING

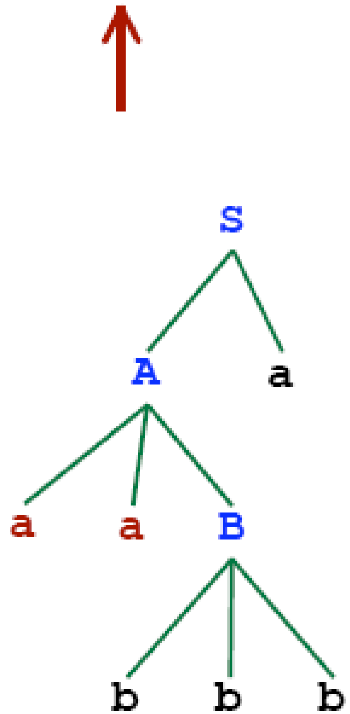
Input: `aabbde`



1. S $\rightarrow$ A`a`
2. $\rightarrow$ C`e`
3. A $\rightarrow$ `aa`B
4. $\rightarrow$ `aaba`
5. B $\rightarrow$ `bbb`
6. C $\rightarrow$ `aa`D
7. D $\rightarrow$ `bbd`

RECURSIVE DESCENT PARSING

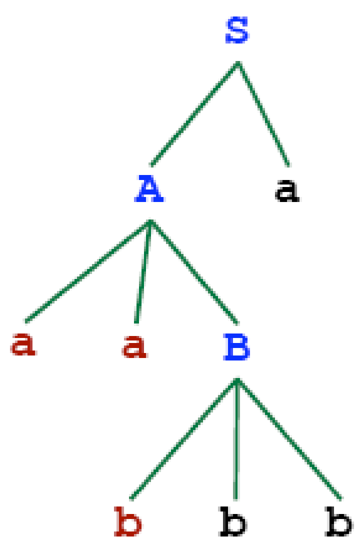
Input: `aabbde`



1. S → Aa
2. → Ce
3. A → aaB
4. → aaba
5. B → bbb
6. C → aaD
7. D → bbd

RECURSIVE DESCENT PARSING

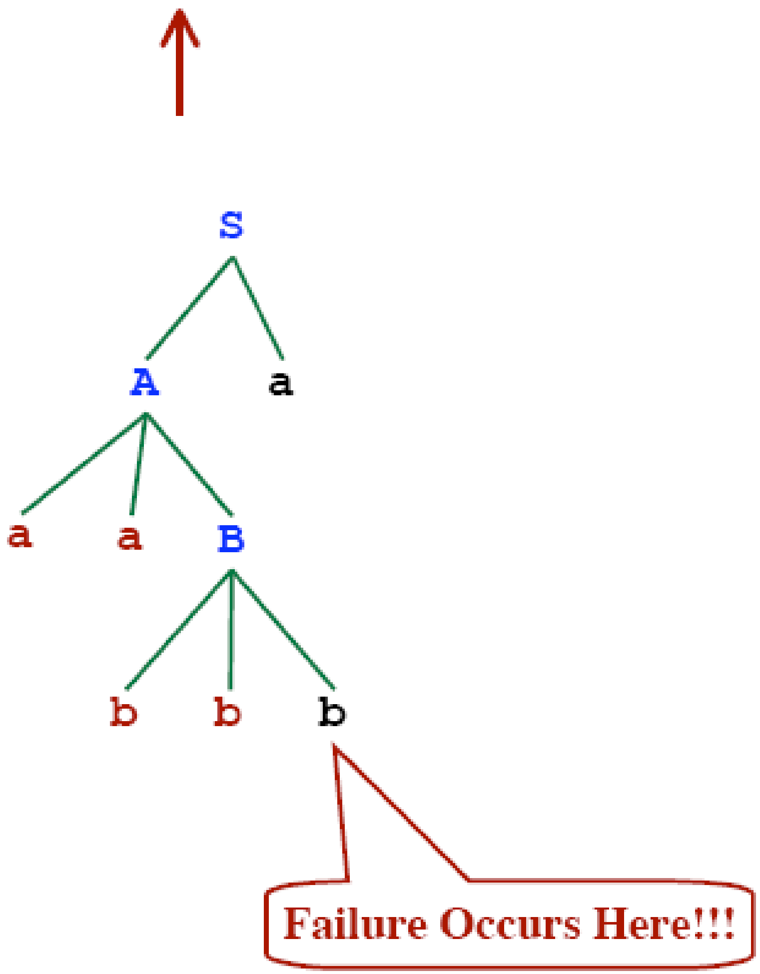
Input: `aabbde`



1. S → Aa
2. → Ce
3. A → aaB
4. → aaba
5. B → bbb
6. C → aaD
7. D → bbd

RECURSIVE DESCENT PARSING

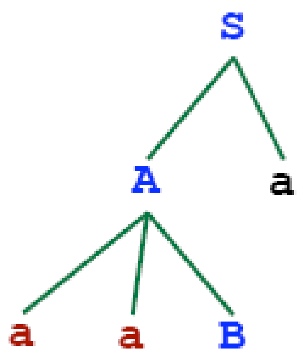
Input: `aabbde`



1. S → Aa
2. → Ce
3. A → aaB
4. → aaba
5. B → bbb
6. C → aaD
7. D → bbd

RECURSIVE DESCENT PARSING

Input: `aabbde`

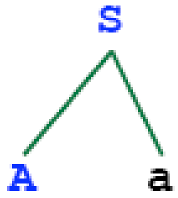


1. S → Aa
2. → Ce
3. A → aaB
4. → aaba
5. B → bbb
6. C → aaD
7. D → bbd

We need an ability to back up in the input!!!

RECURSIVE DESCENT PARSING

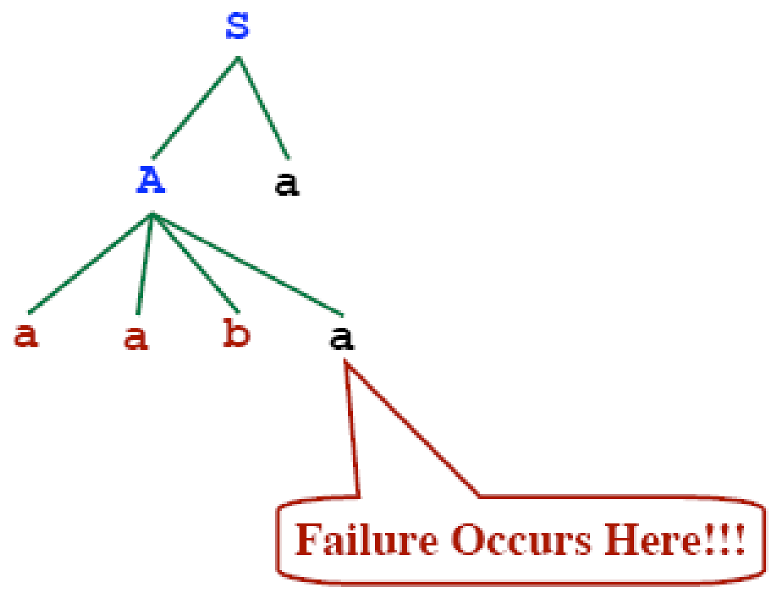
Input: `aabbde`



1. S → Aa
2. → Ce
3. A → aaB
4. → aaba
5. B → bbb
6. C → aaD
7. D → bbd

RECURSIVE DESCENT PARSING

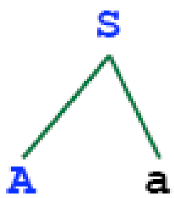
Input: `aabbde`



1.	S	→	Aa
2.		→	Ce
3.	A	→	aaB
4.		→	aaba
5.	B	→	bbb
6.	C	→	aaD
7.	D	→	bbd

RECURSIVE DESCENT PARSING

Input: `aabbde`



1. S → Aa
2. → Ce
3. A → aaB
4. → aaba
5. B → bbb
6. C → aaD
7. D → bbd

RECURSIVE DESCENT PARSING

Input: `aabbde`

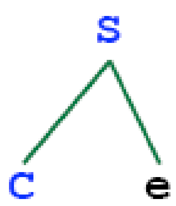
↑

S

1. S → Aa
2. → Ce
3. A → aaB
4. → aaba
5. B → bbb
6. C → aaD
7. D → bbd

RECURSIVE DESCENT PARSING

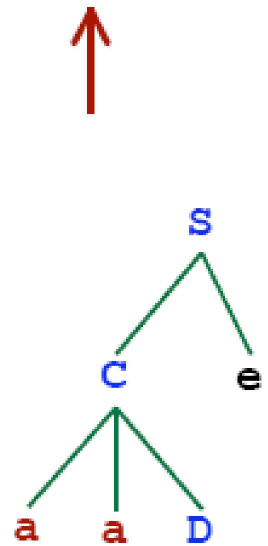
Input: `aabbde`



1.	S	→	A`a`
2.		→	C`e`
3.	A	→	`aa`B
4.		→	`aaba`
5.	B	→	`bbb`
6.	C	→	`aa`D
7.	D	→	`bbd`

RECURSIVE DESCENT PARSING

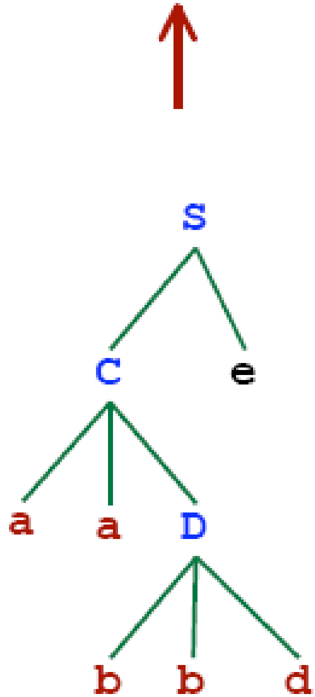
Input: **aabbde**



1. S $\rightarrow$ A**a**
2. $\rightarrow$ C**e**
3. A $\rightarrow$ **aa**B
4. $\rightarrow$ **aaba**
5. B $\rightarrow$ **bbb**
6. C $\rightarrow$ **aa**D
7. D $\rightarrow$ **bbd**

RECURSIVE DESCENT PARSING

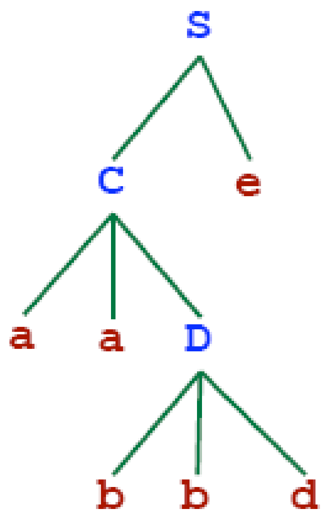
Input: **aabbde**



1. S → A**a**
2. → C**e**
3. A → **aa**B
4. → **aaba**
5. B → **bbb**
6. C → **aa**D
7. D → **bbd**

RECURSIVE DESCENT PARSING

Input: `aabbde`



1.	S	$\rightarrow$	Aa
2.		$\rightarrow$	Ce
3.	A	$\rightarrow$	aaB
4.		$\rightarrow$	aaba
5.	B	$\rightarrow$	bbb
6.	C	$\rightarrow$	aaD
7.	D	$\rightarrow$	bbd

Successfully parsed!!

FIRST FUNCTION

<u>What is FIRST of a Non-Terminal of a Grammar:</u>

A Non-terminal can generate a sequence of terminals(non-empty string) or empty string. The collection of initial terminal of all these strings is called a FIRST of a Non-terminal of a Grammar.

<u>How to find set FIRST(X):</u>

For all productions whose LHS is X,

1. If RHS starts with terminal, then add that terminal to the set FIRST(X).
2. If RHS is ϵ, then add ϵ to the set FIRST(X).
3. If RHS starts with Non-Terminal(say Y), then add FIRST(Y) to the set FIRST(X). If FIRST(Y) includes ϵ, then, also add FIRST(RHS except Y) to the set FIRST(X).

COMPUTING THE FIRST FUNCTION

For all symbols X in the grammar...

```
if X is a terminal then
   FIRST(X) = { X }

if X → ε is a rule then
   add ε to FIRST(X)

if X → Y1 Y2 Y3 ... YK is a rule then
   if a ∈ FIRST(Y1) then
      add a to FIRST(X)
   if ε ∈ FIRST(Y1) and a ∈ FIRST(Y2) then
      add a to FIRST(X)
   if ε ∈ FIRST(Y1) and ε ∈ FIRST(Y2) and a ∈ FIRST(Y3) then
      add a to FIRST(X)
   ...
   if ε ∈ FIRST(Yi) for all Yi then
      add ε to FIRST(X)
```

Repeat until nothing more can be added to any sets.

THE FIRST($X_1X_2X_3...X_N$)

```
Result = {}
Add everything in FIRST(X₁), except ε, to result
if ε ∈ FIRST(X₁) then
  Add everything in FIRST(X₂), except ε, to result
  if ε ∈ FIRST(X₂) then
    Add everything in FIRST(X₃), except ε, to result
    if ε ∈ FIRST(X₃) then
      Add everything in FIRST(X₄), except ε, to result
      ...
        if ε ∈ FIRST(X_{N-1}) then
          Add everything in FIRST(X_N), except ε, to result
          if ε ∈ FIRST(X_N) then
            // Then X₁ ⇒* ε, X₂ ⇒* ε, X₃ ⇒* ε, ... X_N ⇒* ε
            Add ε to result
          endIf
        endIf
      ...
    endIf
  endIf
endIf
```

First Sets

- First(X) is specified like this:
  - base case:
    - if T is a terminal symbol then First (T) = {T}
  - inductive case:
    - if X is a non-terminal and (X:= ABC...) then
      - First (X) = First (ABC...)
        where First(ABC...) = F1 U F2 U F3 U ... and
        - F1 = First (A)
        - F2 = First (B), if A is Nullable; emptyset otherwise
        - F3 = First (C), if A is Nullable & B is Nullable; emp...
        - ...

Computing First Sets

- Compute First(X):
  - initialize:
    - if T is a terminal symbol then First (T) = {T}
    - if T is non-terminal then First(T) = { }
  - while First(X) changes (for any X) do
    - for all X and all rules (X:= ABC...) do
      - First (X) := First(X) U First (ABC...)

        where First(ABC...) := F1 U F2 U F3 U ... and
        - F1 := First (A)
        - F2 := First (B), if A is Nullable; emptyset otherwise
        - F3 := First (C), if A is Nullable & B is Nullable; emp...
        - ...

Computing Follow Sets

- Follow(X) is computed iteratively
  - base case:
    - initially, we assume nothing in particular follows X
      - (when computing, Follow (X) is initially { })
  - inductive case:
    - if (Y := s1 X s2) for any strings s1, s2 then
      - Follow (X) = First (s2)
    - if (Y := s1 X s2) for any strings s1, s2 then
      - Follow (X) = Follow(Y), if s2 is Nullable

What is FOLLOW of a Non-Terminal of a Grammar:

In a derivation process, the collection of initial terminal(i.e. FIRST) of a string which follows Non-terminal, is called FOLLOW of that Non-terminal. For finding FOLLOW set of a Non-Terminal, check in RHS of all productions which consist of that Non-Terminal.

How to find set FOLLOW(X):

1. If X is "Start" symbol, then add $ to the set FOLLOW(X). (Reason: Each string generated from grammar is assumed that it ends in $. For e.g. abc$ or a+b$. As that string can be generated from the Start symbol, Start symbol is followed by $).
2. If in any RHS, X is followed by terminal (say t), then add t to the set FOLLOW(X).
3. If in any RHS, X is followed by Non-terminal (say Y), then add FIRST(Y) except ϵ to the set FOLLOW(X). If FIRST(Y) contains ϵ, then you have to also add FIRST of remaining part of RHS after Y to the set FOLLOW(X). If remaining part of RHS after Y is empty, then add FOLLOW(LHS) to the set FOLLOW(X).
4. If X is the last symbol in any RHS (For e.g. Z->wX), then add FOLLOW(LHS) i.e. FOLLOW(Z) to the set FOLLOW(X). (Reason: RHS is derived from LHS. So whatever follows RHS, also follows LHS. As X is last symbol in RHS, FOLLOW(X) includes FOLLOW(LHS)).

FOLLOW SETS

- FOLLOW(A) is the set of terminals (including end marker of input - \$) that may follow non-terminal A in some sentential form.
- FOLLOW(A) = $\{c \mid S \Rightarrow^+ \ldots Ac \ldots\} \cup \{\$\}$ if $S \Rightarrow^+ \ldots A$
- For example, consider $L \Rightarrow^+ (())(L)L$

  Both ‘)’ and end of file can follow L
- NOTE: ε is ***never*** in FOLLOW sets

COMPUTING FOLLOW(A)

1. If A is start symbol, put \$ in FOLLOW(A)
2. Productions of the form B -> α A β,

   Add FIRST(β) – {ε} to FOLLOW(A)
3. Productions of the form B -> α A or

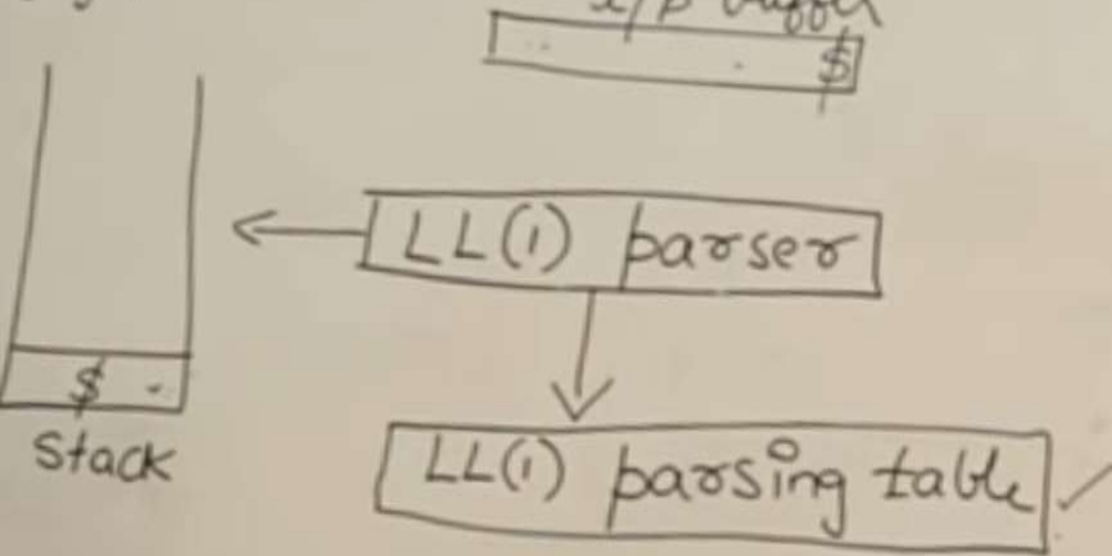
   B -> α A β where $\beta \Rightarrow^* \varepsilon$

   Add FOLLOW(B) to FOLLOW(A)

LL(1)

left to righ

LMD



First( )

Follow( )

S → aABCD

A → b

B → c

C → d

D → e

First( )

Follow( )

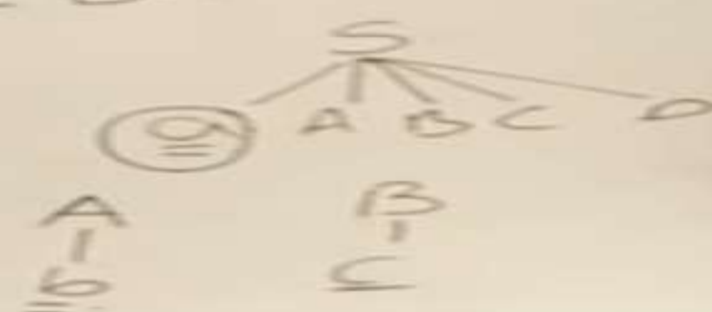
S → aABCD

A → b

B → c

C → d

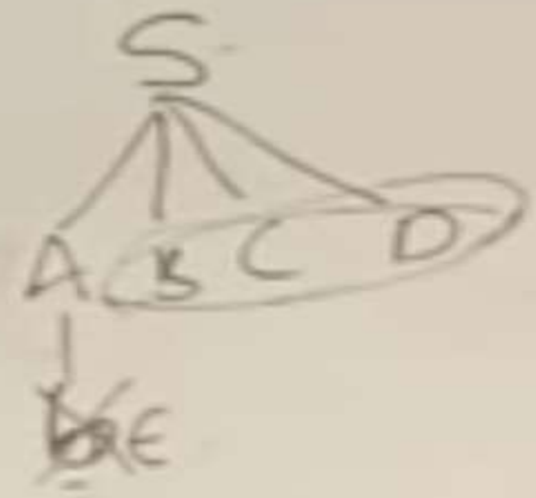
D → e



First( )

Follow( )

$\boxed{S\$}$



$S \to ABCD$

$A \to b/\epsilon$

$B \to c$

$C \to d$

$D \to e$

$A \to BC$

Aabc$

BCabc$

First( )

Follow( )

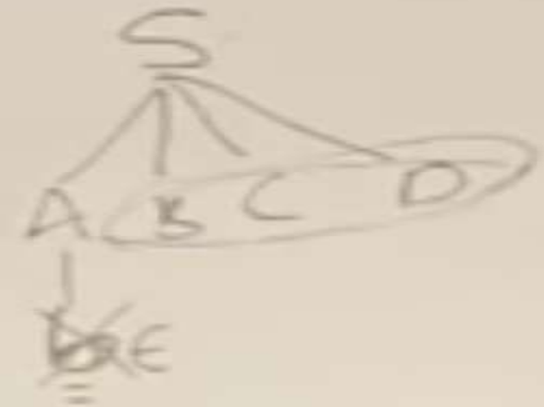
$S \rightarrow \cancel{a}\,ABCD$

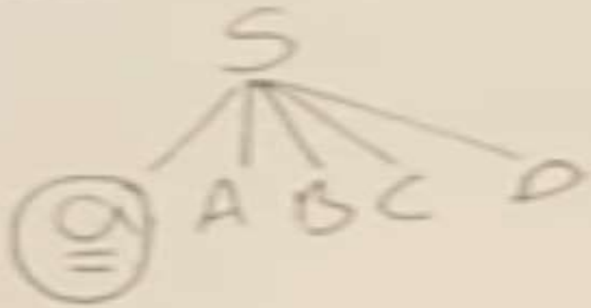
$A \rightarrow b/\epsilon$

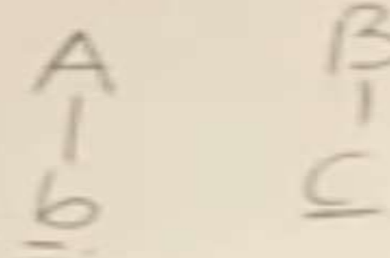
$B \rightarrow c$

$C \rightarrow d$

$D \rightarrow e$







S$

ABCD$

ABdD$

$\boxed{\text{abcd\$}}$

$S \to ABCDE$

$A \to a / \epsilon$

$B \to b / \epsilon$

$C \to c$

$D \to d / \epsilon$

$E \to e / \epsilon$

$S \to Bb / Cd$

$B \to aB / \epsilon$

$C \to cC / \epsilon$

$E \to TE'$

$E' \to +TE' / \epsilon$

$T \to FT'$

$T' \to *FT' / \epsilon$

$F \to id / (E)$

$S \to ACB / CbB / Ba$

$A \to da / BC$

$B \to g / \epsilon$

$C \to h / \epsilon$

	First	Follow
$S \to ABCDE$	$\{a, b, c\}$	$\{\$\}$
$A \to a/\epsilon$	$\{a, \epsilon\}$	$\{b, c\}$
$B \to b/\epsilon$	$\{b, \epsilon\}$	$\{c\}$
$C \to c$	$\{c\}$	$\{d, e, \$\}$
$D \to d/\epsilon$	$\{d, \epsilon\}$	$\{e, \$\}$
$E \to e/\epsilon$	$\{e, \epsilon\}$	$\{\$\}$

$S \to Bb/Cd$	$\{a, b, c, d\}$	$\{\$\}$
$B \to aB/\epsilon$	$\{a, \epsilon\}$	$\{b\}$
$C \to cC/\epsilon$	$\{c, \epsilon\}$	$\{d\}$

$E \to TE'$

$E' \to +TE'/\epsilon$

$T \to FT'$

$T' \to *FT'/\epsilon$

$F \to id/(E)$

$S \to ACB/CbB/Ba$

$A \to da/BC$

$B \to g/\epsilon$

$C \to h/\epsilon$

	First	Follow
$E \rightarrow TE'$	$\{id, (\}$	$\{\$,)\}$
$E' \rightarrow +TE' / \epsilon$	$\{+, \epsilon\}$	$\{\$,)\}$
$T \rightarrow FT'$	$\{id, (\}$	$\{+, \$,)\}$
$T' \rightarrow *FT' / \epsilon$	$\{*, \epsilon\}$	$\{+, \$,)\}$
$F \rightarrow id / (E)$	$\{id, (\}$	$\{*, +, \$,)\}$

	First	Follow
$S \rightarrow ACB / CbB / Ba$	$\{d, g, h, \epsilon, b, a\}$	$\{\$\}$
$A \rightarrow da / BC$	$\{d, g, h, \epsilon\}$	$\{h, g\ \$\}$
$B \rightarrow g / \epsilon$	$\{g, \epsilon\}$	$\{\$, a, h, g\}$
$C \rightarrow h / \epsilon$	$\{h, \epsilon\}$	$\{g, \$, b, h,$

	First	Follow
$S \rightarrow ACB / CbB / Ba$	$\{d, g, h, \epsilon, b, a\}$	$\{\$\}$
$A \rightarrow da / BC$	$\{d, g, h, \epsilon\}$	$\{h, g\ \$\}$
$B \rightarrow g / \epsilon$	$\{g, \epsilon\}$	$\{\$, a, h, g\}$
$C \rightarrow h / \epsilon$	$\{h, \epsilon\}$	$\{g, \$, b, h,$

$S \rightarrow aABb$	$\{a\}$	$\{\$\}$
$A \rightarrow c/\epsilon$	$\{c, \epsilon\}$	$\{d, b\}$
$B \rightarrow d/\epsilon$	$\{d, \epsilon\}$	$\{b\}$

$S \rightarrow aBDh$	$\{a\}$	$\{\$\}$
$B \rightarrow cC$	$\{c\}$	$\{g, f, h\}$
$C \rightarrow bC/\epsilon$	$\{b, \epsilon\}$	$\{g, f, h\}$
$D \rightarrow EF$	$\{g, f, \epsilon\}$	$\{h\}$
$E \rightarrow g/\epsilon$	$\{g, \epsilon\}$	$\{f, h\}$
$F \rightarrow f/\epsilon$	$\{f, \epsilon\}$	$\{h\}$

FIRST - EXAMPLE

- $P \longrightarrow i \mid c \mid n\ T\ S$
- $Q \longrightarrow P \mid a\ S \mid b\ S\ c\ S\ T$
- $R \longrightarrow b \mid \varepsilon$
- $S \longrightarrow c \mid R\ n \mid \varepsilon$
- $T \longrightarrow R\ S\ q$

- FIRST(P) =
- FIRST(Q) =
- FIRST(R) =
- FIRST(S) =
- FIRST(T) =

FIRST - EXAMPLE

- $S \longrightarrow a\ S\ e \mid S\ T\ S$
- $T \longrightarrow R\ S\ e \mid Q$
- $R \longrightarrow r\ S\ r \mid \varepsilon$
- $Q \longrightarrow S\ T \mid \varepsilon$

- FIRST(S) =
- FIRST(R) =
- FIRST(T) =
- FIRST(Q) =

EXAMPLE

- $E \longrightarrow TE'$
- $E' \longrightarrow +TE' \mid \varepsilon$
- $T \longrightarrow FT'$
- $T' \longrightarrow *FT' \mid \varepsilon$
- $F \longrightarrow (E) \mid id$

- FIRST(E) = {(,id}
- FIRST(E′) = {+, ε}
- FIRST(T) = {(,id}
- FIRST(T′) = {*, ε}
- FIRST(F) = {(,id}}

- FOLLOW(E) = {$}
- FOLLOW(E′) =
- FOLLOW(T) =
- FOLLOW(T′) =
- FOLLOW(F) =

Using rule #1

1. If A is start symbol, put $ in FOLLOW(A)

Assume the first non-terminal is the start symbol

EXAMPLE

- E → TE′
- E′ → + T E′ | ε
- T → F T′
- T′ → * F T′ | ε
- F → (E) | id

- FIRST(E) = {(, id}
- FIRST(E′) = {+, ε}
- FIRST(T) = {(, id}
- FIRST(T′) = {*, ε}
- FIRST(F) = {(, id}

- FOLLOW(E) = {$,)}
- FOLLOW(E′) =
- FOLLOW(T) = {+}
- FOLLOW(T′) =
- FOLLOW(F) = {*}

Using rule #2

2. Productions of the form B ⟶ α A β, Add FIRST(β) – {ε} to FOLLOW(A)

EXAMPLE

- $E \rightarrow T E'$
- $E' \rightarrow + T E' \mid \varepsilon$
- $T \rightarrow F T'$
- $T' \rightarrow * F T' \mid \varepsilon$
- $F \rightarrow (E) \mid id$

- $FIRST(E) = \{(, id\}$
- $FIRST(E') = \{+, \varepsilon\}$
- $FIRST(T) = \{(, id\}$
- $FIRST(T') = \{*, \varepsilon\}$
- $FIRST(F) = \{(, id\}\}$

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

Using rule #3

3. Productions of the form $B \longrightarrow \alpha A$

or $B \longrightarrow \alpha A \beta$ where $\beta \Rightarrow^* \varepsilon$

Add FOLLOW(B) to FOLLOW(A)

EXAMPLE

- $S \longrightarrow (A) \mid \varepsilon$
- $A \longrightarrow T E$
- $E \longrightarrow \& T E \mid \varepsilon$
- $T \longrightarrow (A) \mid a \mid b \mid c$

- FOLLOW(S) =
- FOLLOW(A) =
- FOLLOW(E) =
- FOLLOW(T) =

- FIRST(T) =
- FIRST(E) =
- FIRST(A) =
- FIRST(S) =

EXAMPLE

- $S \longrightarrow (A) \mid \varepsilon$
- $A \longrightarrow T\ E$
- $E \longrightarrow \&\ T\ E \mid \varepsilon$
- $T \longrightarrow (A) \mid a \mid b \mid c$

- FIRST(T) = {(,a,b,c}
- FIRST(E) = {&, ε }
- FIRST(A) = {(,a,b,c}
- FIRST(S) = {(,ε}

- FOLLOW(S) = {$}
- FOLLOW(A) = {)}
- FOLLOW(E) = FOLLOW(A) = {)}
- FOLLOW(T) = FIRST(E) ∪ FOLLOW(E) = {&,)}

EXAMPLE

- $S \rightarrow a\ S\ e \mid B$
- $B \rightarrow b\ B\ C\ f \mid C$
- $C \rightarrow c\ C\ g \mid d \mid \varepsilon$

- FIRST(C) =
- FIRST(B) =
- FIRST(S) =

- FOLLOW(C) =
- FOLLOW(B) =
- FOLLOW(S) = {\$}

Assume the first non-terminal is the start symbol

1. If A is start symbol, put \$ in FOLLOW(A)
2. Productions of the form B α A β,

   Add FIRST(β) – {ε} to FOLLOW(A)
3. Productions of the form B α A or

   B α A β where $\beta \Rightarrow^* \varepsilon$

   Add FOLLOW(B) to FOLLOW(A)

EXAMPLE

- $S \longrightarrow a\ S\ e \mid \underline{B}$
- $B \longrightarrow b\ B\ C\ f \mid \underline{C}$
- $C \longrightarrow c\ C\ g \mid d \mid \varepsilon$

- FIRST(C) = {c,d,ε}
- FIRST(B) = {b,c,d,ε}
- FIRST(S) = {a,b,c,d,ε}

- FOLLOW(C) =

  $\{f,g\} \cup \text{FOLLOW}(B)$

  $= \{c,d,e,f,g,\$\}$
- FOLLOW(B) =

  $\{c,d\} \cup \text{FOLLOW}(S)$

  $= \{c,d,e,\$\}$
- FOLLOW(S) = {$, e }

$S \rightarrow A$
$A \rightarrow aB \,/\, Ad$
$B \rightarrow b$
$C \rightarrow g$

- 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-
-
  - $S \rightarrow A$
  - $A \rightarrow aBA'$
  - $A' \rightarrow dA' \,/\, \in$
  - $B \rightarrow b$
  - $C \rightarrow g$
-
- Now, the first and follow functions are as follows-

- **First Functions-**
-
- First(S) = First(A) = { a }
- First(A) = { a }
- First(A') = { d , ∈ }
- First(B) = { b }
- First(C) = { g }
-
- **Follow Functions-**
-
- Follow(S) = { \$ }
- Follow(A) = Follow(S) = { \$ }
- Follow(A') = Follow(A) = { \$ }
- Follow(B) =

$S \rightarrow (L) / a$
$L \rightarrow SL'$
$L' \rightarrow ,SL' / \in$

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

- Follow(S) =
- { \$ } ∪ { First(L') – ∈ } ∪ Follow(L) ∪ Follow(L')
- = { \$, , ,) }
- Follow(L) = {) }
- Follow(L') = Follow(L) = {) }

S → AaAb / BbBa
A → ∈
B → ∈

- **First Functions-**
-
- First(S) =
- { First(A) – ∈ } ∪ First(a) ∪ { First(B) – ∈ } ∪ First(b)
- = { a , b }
- First(A) = { ∈ }
- First(B) = { ∈ }
-

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

$E \rightarrow E + T \,/\, T$

$T \rightarrow T \times F \,/\, F$

$F \rightarrow (E) \,/\, id$

- 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-
-
  - $E \rightarrow TE'$
  - $E' \rightarrow + TE' \,/\, \in$
  - $T \rightarrow FT'$
  - $T' \rightarrow x\, FT' \,/\, \in$
  - $F \rightarrow (E) \,/\, id$

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

S → ACB / CbB / Ba
A → da / BC
B → g / ∈
C → h / ∈

First Functions-

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

- 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)

S->xyz/aBC
B->c/cd
C->eg/df

FIRST(S)={x,a}
FIRST(B)={c}
FIRST(C)={e,d}

FOLLOW(S)={$}
FOLLOW(B)={e,d}
FOLLOW(C)={$}

S->ABCDE
A->a/ϵ
B->b/ϵ
C->c
D->d/ϵ
E->e/ϵ

FIRST(S)={a,b,c}
FIRST(A)={a,ϵ}
FIRST(B)={b,ϵ}
FIRST(C)={c}
FIRST(D)={d,ϵ}
FIRST(E)={e,ϵ}

FOLLOW(S)={$}
FOLLOW(A)={b,c}
FOLLOW(B)={c}
FOLLOW(C)={d,e,$}
FOLLOW(D)={e,$}
FOLLOW(E)={$}

S->AB/C
A->D/a/ϵ
B->b
C->ϵ
D->d

FIRST(S)={d,a,b,ϵ}
FIRST(A)={d,a,ϵ}
FIRST(B)={b}
FIRST(C)={ϵ}
FIRST(D)={d}

FOLLOW(S)={$}
FOLLOW(A)={b}
FOLLOW(B)={$}
FOLLOW(C)={$}
FOLLOW(D)={b}

X->YZ
Y->m/n/ϵ
Z->m

FIRST(X)={m,n}
FIRST(Y)={m,n,ϵ}
FIRST(Z)={m}

FOLLOW(X)={$}
FOLLOW(Y)={m}
FOLLOW(Z)={$}

QUESTIONS ?