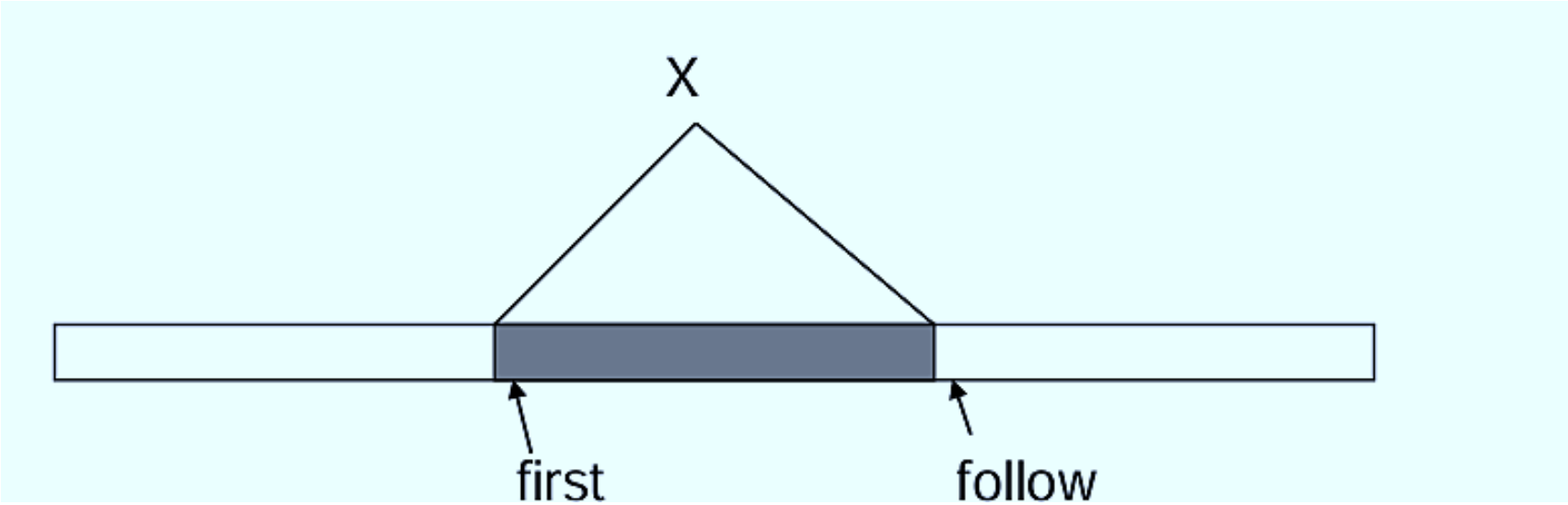
Constructing Parse Table

- Table can be constructed if for every non terminal, every lookahead symbol can be handled by at most one production
- First(α) for a string of *terminals and non terminals* α is

  Set of symbols that might begin the fully expanded (made of only tokens) version of α
- Follow(X) for a non terminal X is

  set of symbols that might follow the derivation of X in the input stream



First

- The **FIRST** set of a grammar symbol (terminal or non-terminal) is the set of terminals that can appear **at the beginning** of strings derived from that symbol.

Rules to Compute FIRST:

1. If **X** is a terminal → **FIRST(X) = { X }**

2. If **X → ε** (X can produce epsilon) → **ε ∈ FIRST(X)**

3. If **X** is a non-terminal and has production: X → Y1 Y2 Y3 … Yn
Then:
   - Add FIRST(Y1) to FIRST(X) (excluding ε).
   - If Y1 ⇒* ε, then also add FIRST(Y2), and so on.
   - If all Y1, Y2, … Yn can derive ε, then add ε to FIRST(X).

FIRST helps in predictive parsing to decide which production to use.

Examples

S → ABC | ghi | jkl — First (s) = {a, b, c, g, j }

A → a | b | c — First (A) = {a, b, c}

B → b — First (B) = {b}

D → d — First (D) = {d}

S → ABC — First (A) = {a, b, c, d, e, f, €}

A → a | b | € — First (A) = {a, b, €}

B → c | d | € — First (B) = {c, d, €}

D → e | f | € — First (D) = {e, f, €}

E → TE’ — First (E) = {id, (}

E’ → *TE’|€ — First (E’) = {*, €}

T → FT’ — First (T) = {id, (}

T’ → € | +FT’ — First (T’) = {€, +}

F → id | (E — First (F) = {id, (}

Follow

- The **FOLLOW** set of a non-terminal **A** is the set of terminals that can appear **immediately to the right of A** in some derivation

Rules to Compute FOLLOW:

1. For the start symbol **S:** **$ ∈ FOLLOW(S)** (end of input marker).

2. If there is a production $A \rightarrow \alpha B \beta$:
   - Add FIRST(β) (excluding ε) to FOLLOW(B).

3. If there is a production $A \rightarrow \alpha B$ or $A \rightarrow \alpha B \beta$ where $\beta \Rightarrow^* \varepsilon$:
   - Add FOLLOW(A) to FOLLOW(B).

FOLLOW helps in cases when ε is in FIRST, to know where parsing can continue.

Examples

S → ACD

C → a | b

Follow (A) = First (C) = {a, b)

Follow (D) = Follow (S) = {$ }

S → aSbS | bSaS| $

Follow (S) = {$, b, a}

S → AaAb | BbBa

A → €

B → €

Follow (A) = {a, b}

Follow (B) = {b, a}

S → ABC

A → DEF

B → €

C → €

D → €

E → €

F → €

Follow (A) = First (B)

But B, C indicates €

So follow (A) = $

More Examples

E → TE’

E’ → +TE’|€

T → FT’

T’ → *FT’ | €

F → id | (E)

First (E) = {id, (}

First (E’) = {+, € }

First (T) = {id, (}

First (T’) = {*, € }

First (F) = { id, (}

Follow (E) = {$,)}

Follow (E’) = follow (E) = {$,)}

Follow (T) = First (E’) U Follow (E) = {+, $,)}

Follow (T’) = Follow (T) = {+, $,)}

Follow (F) = First (T’) = {*, €}
= {*} U Follow (T) U Follow (T’)
= { *, +, $,)}

More Examples Contd.,

S → ABCDE	First (S) = {a, b, c } bcz no €	Follow (S) = {$ }
A → a | €	First (A) = {a, € }	Follow (A) = {b, c }
B → b | €	First (B) = {b, € }	Follow (B) = {c }
C → c	First (C) = {c }	Follow (C) = {d, e, $ }
D → d | €	First (D) = {d, € }	Follow (D) = {e, $ }
E → e | €	First (E) = {e, € }	Follow (E) = {$ }

Assignment: Identify First

1 *Program* ::= *Header DeclSec Block* **.**
2 *Header* ::= **program identifier ;**
3 *DeclSec* ::= *VarDecls ProcDecls*
4 *VarDecls* ::= *VarDecl VarDecls*
5 *VarDecls* ::= ε
6 *VarDecl* ::= *DataType IdList;*
7 *DataType* ::= **integer**
8 *DataType* ::= **real**
9 *IdList* ::= **identifier** *MoreIdList*
10 *MoreIdList* ::= **,** **identifier** *MoreIdList*
11 *MoreIdList* := ε
12 *ProcDecls* ::= *ProcDecl ProcDecls*
13 *ProcDecls* ::= ε
14 *ProcDecl* ::= *ProcHeader DeclSec Block* **;**
15 *ProcHeader* ::= **procedure identifier** *ParamList* **;**
16 *ParamList* ::= **(** *ParamDecls* **)**
17 *ParamList* ::= ε
18 *ParamDecls* ::= *ParamDecl MoreParamDecls*
19 *MoreParamDecls* ::= **;** *ParamDecl MoreParamDecls*
20 *MoreParamDecls* ::= ε
21 *ParamDecl* ::= *DataType* **identifier**
22 *Block* ::= **begin** *Statements* **end**
23 *Statements* ::= *Statement MoreStatements*
24 *MoreStatements* ::= **;** *Statement MoreStatements*
25 *MoreStatements* ::= ε
26 *Statement* ::= **read identifier**
27 *Statement* ::= **set identifier =** *Expression*
28 *Statement* ::= **write identifier**
29 *Statement* ::= **if** *Condition* **then** *Statements ElseClause* **endif**
30 *Statement* ::= **while** *Condition* **do** *Statements* **endwhile**
31 *Statement* ::= **until** *Condition* **do** *Statements* **enduntil**
32 *Statement* ::= **call identifier** *ArgList*
33 *Statement* ::= ε
34 *ElseClause* ::= **else** *Statements*
35 *ElseClause* ::= ε
36 *ArgList* ::= **(** *Args* **)**

Contd.,

37 *ArgList* ::= ε
38 *Args* ::= **identifier** *MoreArgs*
39 *MoreArgs* ::= , **identifier** *MoreArgs*
40 *MoreArgs* ::= ε
41 *Condition* ::= *Expression RelOp Expression*
42 *RelOp* ::= **=**
43 *RelOp* ::= **!**
44 *RelOp* ::= **>**
45 *RelOp* ::= **<**

46 *Expression* ::= *Term MoreExpression*
47 *MoreExpression* ::= *AddOp Term MoreExpression*
48 *MoreExpression* ::= ε
49 *Term* ::= *Factor MoreTerm*
50 *MoreTerm* ::= *MultOp Factor MoreTerm*
51 *MoreTerm* ::= ε
52 *Factor* ::= **identifier**
53 *Factor*::= **constant**
54 *AddOp* ::= **+**

55 *AddOp* ::= **-**
56 *MultOp* ::= ***
57 *MultOp* ::= **/**