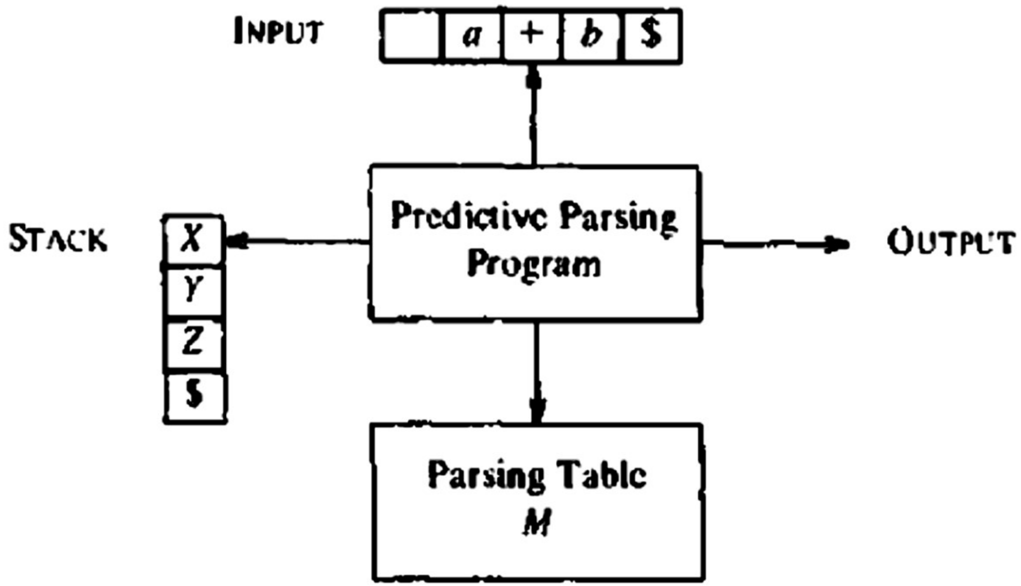
Non-Recursive Predictive Parsing- LL(k) Parsing

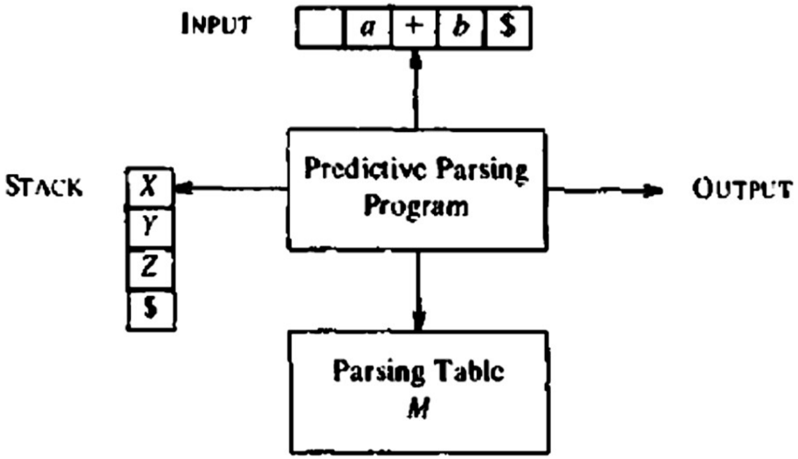
- It is possible to build a nonrecursive predictive parser by maintaining a stack explicitly, rather than implicitly via recursive calls. The key problem during predictive parsing is that of determining the production to be applied for a nonterminal. The nonrecursive parser looks up the production to be applied in a parsing table.



- This parser scan over the input stream using a prefix of tokens to identify the production applied. This parser is also called LL(k) parser, where k is the length of the prefix. “LL” stands for left-to-right scanning of the input stream and left-most derivation respectively.

Non-Recursive Predictive Parsing- LL(k) Parsing

- 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 top of $.
- The parsing table is a two dimensional array M[A, a], where A is a nonterminal, and a is a terminal or the symbol $.



Non-Recursive Predictive Parsing- LL(k) Parsing

Why FIRST and FOLLOW in Compiler Design?

- The need of backtracking is really a complex process to implement a parser.
- If the compiler would have come to know in advance, that what is the "**first** character of the string produced when a production rule is applied", and comparing it to the current character or token in the input string, it can wisely take decision on which production rule to apply.

  S -> cAd

  A -> bc|a

  And the input string is "cad".
- After reading character 'c' in the input string and applying S->cAd, next character in the input string is 'a', then it would directly use the production rule A->a.

Non-Recursive Predictive Parsing- LL(k) Parsing

Why FIRST and FOLLOW in Compiler Design?

- The parser faces one more problem. Let us consider below grammar to understand this problem.

  A -> aBb

  B -> c | ε

  And suppose the input string is “ab” to parse.

- As the first character in the input is a, the parser applies the rule A->aBb. Now the parser checks for the second character of the input string which is b, and the Non-Terminal to derive is B, but the parser can’t get any string derivable from B that contains b as first character.
- But the Grammar does contain a production rule B -> ε, if that is applied then B will vanish. But the parser can apply it only when it knows that the character that follows B in the production rule is same as the current character in the input.

Non-Recursive Predictive Parsing- LL(k) Parsing

Rules to Calculate First(X)

To compute First(X) for all grammar symbols X, apply the following rules:

1. If X is a terminal, then First(X) is {X}.
2. If X->ϵ is a production, then add ϵ to First(X).
3. If X is nonterminal and X->Y_1 Y_2 … Y_k is a production, then place a in First(X) if for some i, a is in First(Y_i), and ϵ is in all of First(Y_1), . . . First(Y_{i-1}); that is, $Y_1 \ldots Y_{i-1} \approx \epsilon$. If ϵ is in First($Y_j$) for all *j* = 1, 2, . . . , k, then add ϵ to First(X).

Non-Recursive Predictive Parsing- LL(k) Parsing

Rules to Calculate First(X)

To compute First(X) for all grammar symbols X, apply the following rules:

1. If X is a terminal, then First(X) is {X}.
2. If X->ϵ is a production, then add ϵ to First(X).
3. If X is nonterminal and $X \to Y_1\ Y_2 \ldots Y_k$ is a production, then place a in First(X) if for some i, a is in First(Y_i), and ϵ is in all of First(Y_1), . . . First(Y_{i-1}); that is, $Y_1 \ldots Y_{i-1} \approx \epsilon$. If ϵ is in First($Y_j$) for all j = 1, 2, . . . , k, then add ϵ to First(X).

Example:

E -> T E '
E ' -> + T E ' | ϵ
T -> F T '
T ' -> * F T ' | ϵ
F -> (E) | id

First(F)={First((E)), First(id))

First(T')={First(* FT '), ϵ)}

First(T)={First(F)}

First(E')={First(+TE '), ϵ)}

First(E)={First(T)}

	First
E	{(,id}
E’	{+,ε}
T	{(,id}
T’	{*,ε}
F	{(,id}

Non-Recursive Predictive Parsing- LL(k) Parsing

Rules to Calculate First(X)

To compute First(X) for all grammar symbols X, apply the following rules:

1. If X is a terminal, then First(X) is {X}.
2. If X->ϵ is a production, then add ϵ to First(X).
3. If X is nonterminal and X->Y_1 Y_2 … Y_k is a production, then place a in First(X) if for some i, a is in First(Y_i), and ϵ is in all of First(Y_1), . . . First(Y_{i-1}); that is, $Y_1 \ldots Y_{i-1} \approx \epsilon$. If ϵ is in First($Y_j$) for all *j* = 1, 2, . . . , k, then add ϵ to First(X).

Consider the grammar

S -> aABe   A -> Abc | b   B -> d

Show the First sets for each nonterminal symbol.

First(B) = {First(d)} = {d}

First(A) = {First(Abc), First(b)} = {First(A), b} ={b}

First(S) = {First(aABe)}={a}

	First
S	**{a}**
A	**{b}**
B	**{d}**

Non-Recursive Predictive Parsing- LL(k) Parsing

Rules to Calculate First(X)

To compute First(X) for all grammar symbols X, apply the following rules:

1. If X is a terminal, then First(X) is {X}.
2. If X->ϵ is a production, then add ϵ to First(X).
3. If X is nonterminal and $X \to Y_1\ Y_2 \ldots Y_k$ is a production, then place a in First(X) if for some i, a is in First(Y_i), and ϵ is in all of First(Y_1), . . . First(Y_{i-1}); that is, $Y_1 \ldots Y_{i-1} \approx \epsilon$. If ϵ is in First($Y_j$) for all *j* = 1, 2, . . . , k, then add ϵ to First(X).

Consider the grammar

S -> A | B,   A -> cA+b | a, B -> cB + a | b

Show the First sets for each nonterminal symbol.

First(B)={c, b}

First(A)={c, a}

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

	First
S	{a,b,c}
A	{a, c}
B	{b, c}

Non-Recursive Predictive Parsing- LL(k) Parsing

Rules to Calculate First(X)

To compute First(X) for all grammar symbols X, apply the following rules:

1. If X is a terminal, then First(X) is {X}.
2. If X->ϵ is a production, then add ϵ to First(X).
3. If X is nonterminal and X->Y_1 Y_2 … Y_k is a production, then place a in First(X) if for some i, a is in First(Y_i), and ϵ is in all of First(Y_1), . . . First(Y_{i-1}); that is, $Y_1 \ldots Y_{i-1} \approx \epsilon$. If ϵ is in First($Y_j$) for all *j* = 1, 2, . . . , k, then add ϵ to First(X).

Consider the grammar

$$S \rightarrow iEtSS' \mid a$$
$$S' \rightarrow eS \mid \epsilon$$
$$E \rightarrow b$$

Show the First sets for each nonterminal symbol.

	First
S	{i, a}
S’	{e, ε}
E	{b}

Non-Recursive Predictive Parsing- LL(k) Parsing

Rules to Calculate Follow(A)

To compute Follow(X) for all grammar symbols X, apply the following rules:

1. Place \$ in Follow(S), where S is the start symbol and \$ is the input right endmarker.
2. If there is a production, $A \to \alpha B \beta$, then everything in First(β) except for ϵ is placed in Follow(B).
3. If there is a production, $A \to \alpha B$, or a production $A \to \alpha B \beta$ where First(β) contains ϵ (i.e. $\beta \approx \epsilon$), then everything in Follow(A) is in Follow(B).

Non-Recursive Predictive Parsing- LL(k) Parsing

Rules to Calculate Follow(A)

To compute Follow(X) for all grammar symbols X, apply the following rules:

1. Place \$ in Follow(S), where S is the start symbol and \$ is the input right endmarker.
2. If there is a production, A->αBβ, then everything in First(β) except for ϵ is placed in Follow(B).
3. If there is a production, A-> αB, or a production A->αBβ where First(β) contains ϵ (i.e. β ≈ ϵ), then everything in Follow(A) is in Follow(B).

Example:

E -> T E '
E ' -> + T E ' | ϵ
T -> F T '
T ' -> * F T ' | ϵ
F -> (E) | id

Follow(E)={\$, First( ))}

Follow(E')={Follow(E)}

Follow(T)={First(E '), Follow(E)}

Follow(T')={Follow(T)}

Follow(F)={First(T '), Follow(T))

	First	Follow
E	{(,id}	{), \$}
E’	{+,ε}	{), \$}
T	{(,id}	{+,), \$}
T’	{*,ε}	{+,), \$}
F	{(,id}	{+, *,), \$}

Non-Recursive Predictive Parsing- LL(k) Parsing

Rules to Calculate Follow(A)

To compute Follow(X) for all grammar symbols X, apply the following rules:

1. Place \$ in Follow(S), where S is the start symbol and \$ is the input right endmarker.
2. If there is a production, A->αBβ, then everything in First(β) except for ϵ is placed in Follow(B).
3. If there is a production, A-> αB, or a production A->αBβ where First(β) contains ϵ (i.e. β ≈ ϵ), then everything in Follow(A) is in Follow(B).

Consider the grammar S -> aABe A -> Abc | b B -> d

Show the First and Follow sets for each nonterminal symbol.

Follow(S) = {\$}

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

Follow(B) = {First(e)} ={ e }

	First	Follow
S	{a}	{\$}
A	{b}	{b, d}
B	{d}	{e}

Non-Recursive Predictive Parsing- LL(k) Parsing

Rules to Calculate Follow(A)

To compute Follow(X) for all grammar symbols X, apply the following rules:

1. Place $ in Follow(S), where S is the start symbol and $ is the input right endmarker.
2. If there is a production, A->αBβ, then everything in First(β) except for ϵ is placed in Follow(B).
3. If there is a production, A-> αB, or a production A->αBβ where First(β) contains ϵ (i.e. β ≈ ϵ), then everything in Follow(A) is in Follow(B).

Consider the grammar S -> A | B, A -> cA+b | a, B -> cB + a | b

Show the First and Follow sets for each nonterminal symbol.

First(B)={c, b}

First(A)={c, a}

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

Follow(S)={$}

Follow(A)={Follow(S), +}

Follow(B)={Follow(S), +}

	First	Follow
S	{a,b,c}	{$}
A	{a, c}	{+, $}
B	{b, c}	{+, $}

Non-Recursive Predictive Parsing- LL(k) Parsing

LL(1) Grammar

A grammar is an LL(1) if all productions conform to the following LL(1) conditions:

1. For each production $A \to \alpha_1 | \alpha_2 | \ldots | \alpha_n$, $First(\alpha_i) \cap First(\alpha_j) = \Phi$, $\forall i \neq j$
2. If nonterminal X can derive ϵ, then $First(X) \cap Follow(X) = \Phi$

Check the grammar S -> A | B, A -> cA+b | a, B -> cB + a | b is LL(1) or not.

First(B)={c, b}

First(A)={c, a}

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

Follow(S)={$}

Follow(A)={Follow(S), +}

Follow(B)={Follow(S), +}

	First	Follow
S	{a, b, c}	{$}
A	{a, c}	{+, $}
B	{b, c}	{+, $}

First(A) ∩ First(B) = {c}
So, this is not LL(1).