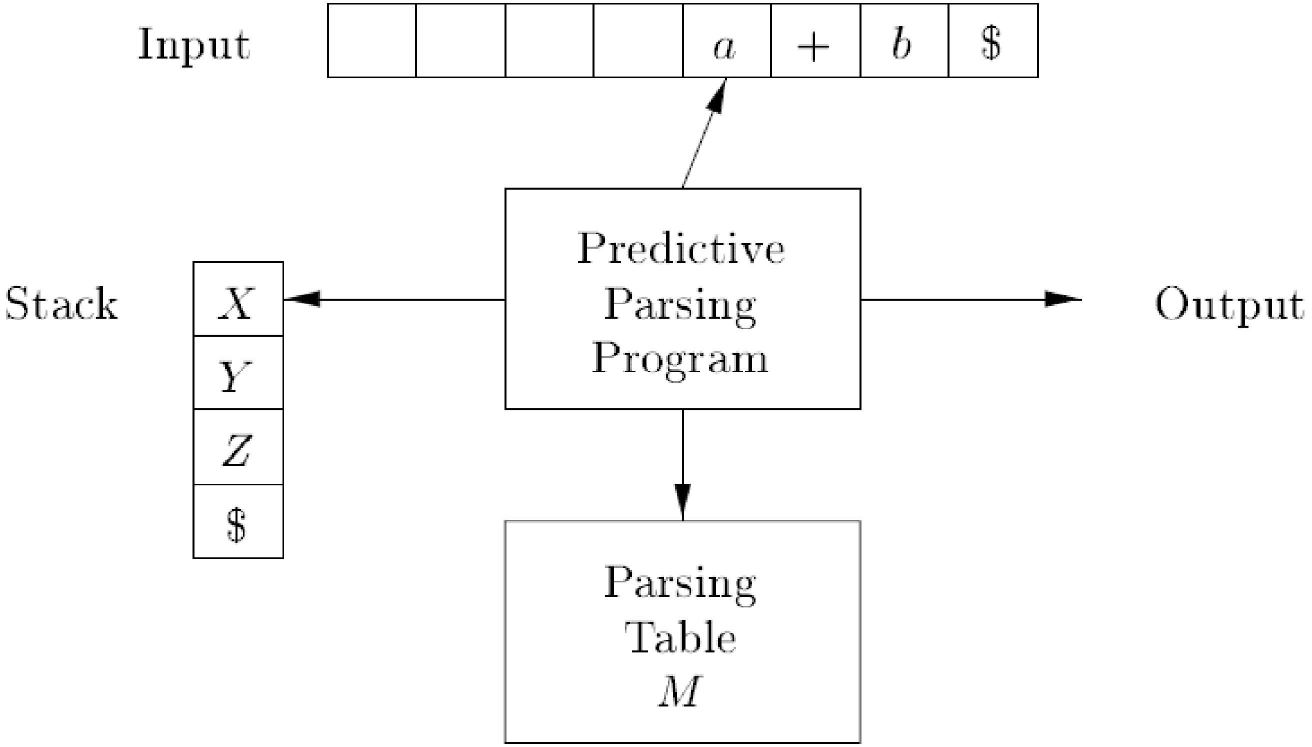
Non-Recursive Predictive Parser

Non-Recursive Predictive Parsing



FIRST ()

- FIRST function is computed for all terminals and non-terminals
- FIRST(α) = the set of terminals that begin all strings derived from α

FIRST ( )

- FIRST(a) = $\{a\}$ if $a \in T$
  FIRST(ε) = $\{\varepsilon\}$
  FIRST(A) = $\cup_{A\to\alpha}$ FIRST(α)
  for $A\to\alpha \in P$

FIRST () – Algorithm

```
• FIRST($X_1X_2\ldots X_k$) =
      if for all $j$ = 1, …, $i$-1 : $\varepsilon \in$ FIRST($X_j$) then
            add non-$\varepsilon$ in FIRST($X_i$) to FIRST($X_1X_2\ldots X_k$)
      if for all $j$ = 1, …, $k$ : $\varepsilon \in$ FIRST($X_j$) then
            add $\varepsilon$ to FIRST($X_1X_2\ldots X_k$)
```

FOLLOW

- FOLLOW(*A*) = the set of terminals that can immediately follow non-terminal *A*

FOLLOW - Algorithm

- FOLLOW(A) =

```
if $A$ is the start symbol $S$ then
        add $ to FOLLOW($A$)
for all $(B \rightarrow \alpha A \beta) \in P$ do
        add FIRST($\beta$)$\backslash\{\varepsilon\}$ to FOLLOW($A$)
for all $(B \rightarrow \alpha A \beta) \in P$ and $\varepsilon \in$ FIRST($\beta$) do
        add FOLLOW($B$) to FOLLOW($A$)
for all $(B \rightarrow \alpha A) \in P$ do
        add FOLLOW($B$) to FOLLOW($A$)
```

Example

- E → TE’
- E’ → +TE’ | ε
- T → FT’
- T’ → *FT’ | ε
- F→ (E) | id

FIRST

- FIRST (E) = FIRST (T) = FIRST(F)

  = {(, id}
- FIRST (E’) = { +, ε}
- FIRST (T’) = { *, ε}

FOLLOW

- FOLLOW(E) = FIRST(‘)’)

  ={$,) }
- FOLLOW(T) = FIRST(E’) U FOLLOW(E)

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

  FIRST(T’) U FOLLOW(T)

- FOLLOW (E’) = FOLLOW (E)

  = {$,) }

- FOLLOW (T’) = FOLLOW (T)

  = {$, +,) }

Another Example

- Ambiguous grammar
  $S \rightarrow$ **i** C **t** $S\ S'$ | **a**
  $S' \rightarrow$ **e** S | ε
  $C \rightarrow$ **b**

- First (S) = {i,a}
- First (S’) = {e, ε }
- First (C) = {b}
- Follow (S) ={$, e}
- Follow(S’) = {$, e}

Predictive Parsing Table

- Row for each non-terminal
- Column for each terminal symbol

  Table[NT, symbol] = Production that matches the [NT, symbol]

  if First(NT) has ε, then add production

  NT → ε in all [NT, a] for all ‘a’ in FOLLOW(NT)

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)		

	a	**b**	**e**	**i**	**t**	**$**
S	$S \rightarrow \mathbf{a}$			$S \rightarrow \mathbf{i}\ C\ \mathbf{t}\ S\ S'$		
S'			$S' \rightarrow \varepsilon$ $S' \rightarrow \mathbf{e}\ S$			$S' \rightarrow \varepsilon$
C		$C \rightarrow \mathbf{b}$				

Parsing action

- push(**$**)
  push(*S*)
  a := *lookahead*
- **repeat**

```
        X := pop()
        if X is a terminal or X = $ then
                match(X) // move to next token, a := lookahead
        else if M[X,a] = X → $Y_1Y_2…Y_k$ then
                push($Y_k$, $Y_{k-1}$, …, $Y_2$, $Y_1$) // such that $Y_1$ is on top
                produce output and/or invoke actions
        else    error()
        endif
until X = $
```

Parsing action Example

Stack	Input String	Action
$E	id + id* id $	[E, id]
$E’T	id + id *id $	[T, id]
$E’T’F	id + id *id $	[F, id]
$E’T’id	id + id *id $	id, id -> pop stack and move input
$E’T’	+ id *id$	[T’, +] -> replace with ε
$E’	+ id *id$	[E’, +]
$E’T+	+ id *id$	+, + → pop stack and move
$E’T	id * id $	[T, id]
$E’T’F	id *id$	[F, id]

Stack	Input String	Action
$E’T’id	id *id$	id, id –> pop
$E’T’	*id $	[T’, *]
$E’T’F*	*id $	*,* -> pop, and move
$E’T’F	id$	[F, id]
$E’T’id	id $	id, id → pop
$E’T’	$	T’, $ -> replace with ε
$E’	$	E’, $ -> replace with ε
$	$	Accept

Error Recovery in LL (1) parser

- *Panic mode*
  - Discard input until a token in a set of designated synchronizing tokens is found
- *Phrase-level recovery*
  - Perform local correction on the input to repair the error
- *Error productions*
  - Augment grammar with productions for erroneous constructs
- *Global correction*
  - Choose a minimal sequence of changes to obtain a global least-cost correction

Error Recovery

- Panic Mode
  - Add synchronizing actions to undefined entries based on FOLLOW
- Phrase Mode
  - Change input stream by inserting missing +, *, (, or)
    For example: **id id** is changed into **id * id or id + id**

Error Recovery

- Error Production
    - Add productions that will take care of incorrect input combinations

Error Recovery

	id	+	*	(	)	$
E	E → TE’			E → TE’	**synch**	**synch**
E’		E’→ +TE’			E’ → ε	E’ → ε
T	T → FT’	**synch**		T → FT’	**synch**	**synch**
T’		T’ → ε	T’ → *FT’		T’ → ε	T’ → ε
F	F → id	**synch**	**synch**	F → (E)	**synch**	**synch**

LL (1)

- A grammar *G* is LL(1) if for each collections of productions

  $A \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$

  for nonterminal *A* the following holds:

  1. FIRST(α_i) $\cap$ FIRST(α_j) = $\varnothing$ for all $i \neq j$
  2. if $\alpha_i \Rightarrow^* \varepsilon$ then
     2.a. $\alpha_j \Rightarrow^* \varepsilon$ for all $i \neq j$
     2.b. FIRST(α_j) $\cap$ FOLLOW(A) = $\varnothing$
     for all $i \neq j$

If then Grammar

- The if then grammar has multiple entries in the parsing table.
- So, confusion on which production to apply
- Ambiguous grammar hence not LL (1)