predictive

Introduction

- The predictive parser has
  1. Input
  2. Stack
  3. Parsing table
  4. Output
- The input contains a sequence of grammar symbols followed by $ (right endmarker)
- The stack contains a sequence of grammar symbols preceded by $ (bottom stack marker)
  - Initially it contains the start symbol of the grammar preceded by the $
- The Parsing table is a two-D array **M[A,a]**
  - **A** is a nonterminal and **a** is the current input symbol

Model of a predictive parser

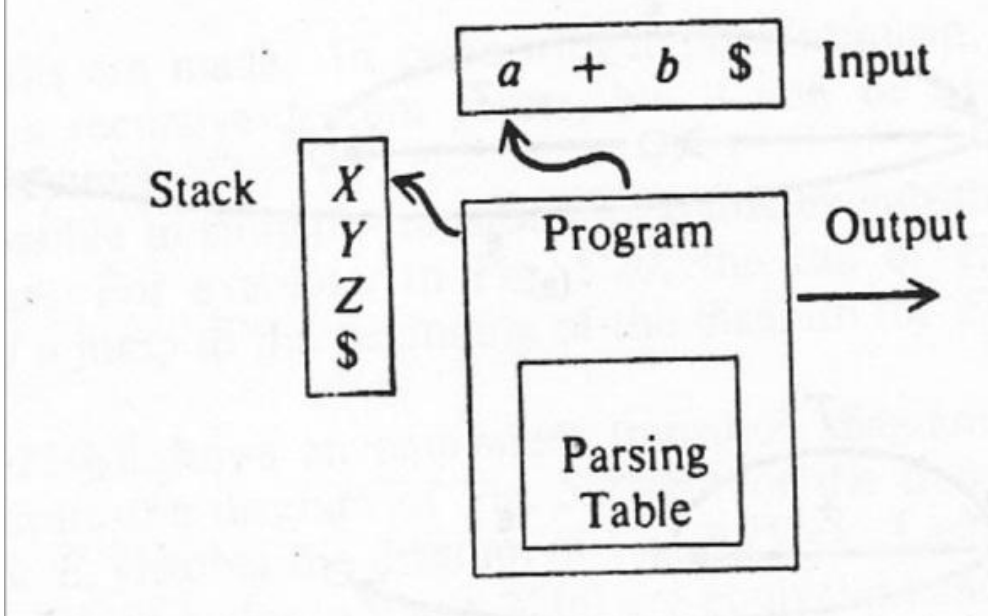

Fig. 5.22. Model of a predictive parser.

Parsing

- The parser is controlled by a program that behaves as follows:
  - The program determined **X**, the top stack symbol and **a** the current input symbol
  - These two symbols determined the action of the parser

Parsing

- There are three possibilities
  1. If $X = a = \$$ then stop and announces successful
  2. If $X = a \neq \$$ pop X from the stack and advance the input pointer
  3. If X is a nonterminal then the program consult entry **M[A,a]**
     - the entry will be a production or an error
     - If M[A,a]= {$X \rightarrow UVW$}
       – the parser replaces *X* on top of the stack by *WVU* (*U* on top)
     - If M[A,a] is an error, the parser calls an error recovery routine

- Initially the input is w$
- Stack is $ S
- W is the input string
- S is the start symbol
- Consider the grammar
- $E \rightarrow TE'$
- $E' \rightarrow +TE' \mid \in$
- $T \rightarrow FT'$
- $T' \rightarrow *FT' \mid \in$
- $F \rightarrow (E) \mid id$

Predictive parsing Table

- A predictive parsing table for the given grammar is
- Blanks are error entries

	id	+	*	(	)	$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
F	$F \rightarrow \textbf{id}$			$F \rightarrow (E)$		

Fig. 5.24. Parsing table for grammar (5.9).

Predictive parsing program

```
repeat
    begin
        let X be the top stack symbol and a the next input symbol;
        if X is a terminal or $ then
            if X = a then
                pop X from the stack and remove a from the input
            else
                ERROR()
        else /* X is a nonterminal */
            if M[X, a] = X → Y_1 Y_2 ··· Y_k then
                begin
                    pop X from the stack;
                    push Y_k, Y_{k-1}, . . . , Y_1 onto the stack, Y_1 on top
                end
            else
                ERROR()
    end
until
    X = $  /* stack has emptied */
```

Fig. 5.23. Predictive parsing program.

example

- Consider w=id + id * id
- The sequence of moves of the predictive parser is as follows

Stack	Input	Output
$\$E$	$\mathbf{id} + \mathbf{id} * \mathbf{id}\$$	
$\$E'T$	$\mathbf{id} + \mathbf{id} * \mathbf{id}\$$	$E \rightarrow TE'$
$\$E'T'F$	$\mathbf{id} + \mathbf{id} * \mathbf{id}\$$	$T \rightarrow FT'$
$\$E'T'\mathbf{id}$	$\mathbf{id} + \mathbf{id} * \mathbf{id}\$$	$F \rightarrow \mathbf{id}$
$\$E'T'$	$+ \mathbf{id} * \mathbf{id}\$$	
$\$E'$	$+ \mathbf{id} * \mathbf{id}\$$	$T' \rightarrow \epsilon$
$\$E'T+$	$+ \mathbf{id} * \mathbf{id}\$$	$E' \rightarrow +TE'$
$\$E'T$	$\mathbf{id} * \mathbf{id}\$$	
$\$E'T'F$	$\mathbf{id} * \mathbf{id}\$$	$T \rightarrow FT'$
$\$E'T'\mathbf{id}$	$\mathbf{id} * \mathbf{id}\$$	$F \rightarrow \mathbf{id}$
$\$E'T'$	$* \mathbf{id}\$$	
$\$E'T'F*$	$* \mathbf{id}\$$	$T' \rightarrow *FT'$
$\$E'T'F$	$\mathbf{id}\$$	
$\$E'T'\mathbf{id}$	$\mathbf{id}\$$	$F \rightarrow \mathbf{id}$
$\$E'T''$	$\$$	
$\$E'$	$\$$	$T' \rightarrow \epsilon$
$\$$	$\$$	$E' \rightarrow \epsilon$

Fig. 5.25. Moves by predictive parser.

FIRST and FOLLOW

- In order to fill the entries of a predictive parsing table we require two functions which are associated with a grammar G

1. FIRST
   - If α is any string of grammar symbols. Let FIRST(α) be the set of terminals that begin strings derived from α
   - If $\alpha \Rightarrow^* \in$, then $\in$ is also in FIRST(α)
2. FOLLOW
   - For FOLLOW(A), for nonterminal A, is the set of terminals a that can appear immediately to the right of A in some sentential form
   - S $\Rightarrow^* \alpha A a \beta$ for some α and β
   - If A can be the rightmost symbol in some sentential form then add $ to FOLLOW(A)

Compute FIRST(X)

1. If X is terminal, then FIRST(X) is {X}
2. If X is nonterminal and $X \rightarrow a\ \alpha$ is a production, then add a to FIRST(X)
   - If $X \rightarrow \in$, then add $\in$ to FIRST(X)
3. If $X \rightarrow Y_1 Y_2 \ldots Y_K$ is a production, then
   - For all i such that all of $Y_1 \ldots Y_{i-1}$ are nonterminals and FIRST(Y_j) contains $\in$ for j=1,2,…,i-1, add every non -$\in$ symbol in FIRST(Y_j) to FIRST(X)
   - If $\in$ is in FIRST(Y_j) for all j=1,2,…,k then add $\in$ to FIRST(X)

Compute FOLLOW(X)

1. \$ is in FOLLOW(S), where S is the start symbol
2. If there is a production $A \rightarrow \alpha B \beta$, $\beta \neq \in$, then everything in FIRST(β) but $\in$ is in FOLLOW(B)
3. If there is a production $A \rightarrow \alpha B$ or a production $A \rightarrow \alpha B\ \beta$ where FIRST(β) contains $\in$, then everything in FOLLOW(A) is in FOLLLOW(B)

Application

- Consider the grammar G given below
- $E \rightarrow TE'$
- $E' \rightarrow +TE' \mid \in$
- $T \rightarrow FT'$
- $T' \rightarrow *FT' \mid \in$
- $F \rightarrow (E) \mid id$
- FIRST(E)= FIRST(T)=FIRST(F)={(, id}
- FIRST(E`)={+, $\in$}
- FIRST(T`}={*, $\in$}
- FOLLOW(E)=FOLLOW(E`)={),$}
- FOLLOW(T)=FOLLOW(T`)={+,),$}
- FOLLOW(F)={+,*,),$}

Construction of Parsing table

- The idea behind the algorithm is
  - $A \rightarrow \alpha$ is a production with a in FIRST(α)
    - Then whenever A is on the top stack and a the current input symbol the parser expand A by α
  - $\alpha = \in$ or $\alpha \Rightarrow^* \in$
    - Expand A by $\in$ if the current input is in FOLLOW(A)
    - or if \$ is on the input and \$ is in FOLLOW(A)

Algorithm for constructing a predictive parsing table

- Input: Grammar G
- Output: Parsing table M
- Method

1. For each production $A \rightarrow \alpha$ of the grammar do step 2 and 3
2. For terminal a in FIRST(α), add $A \rightarrow \alpha$ to M[A,a]
3. If $\in$ is in FIRST(α), add $A \rightarrow \alpha$ to M[A,b] for each terminal b in FOLLOW(A)
    1. If $\in$ is in FIRST(α) and $ is in FOLLOW(A), add $A \rightarrow \alpha$ to M[A,$]
4. Make each undefined entry of M error

	id	+	*	(	)	$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
F	$F \rightarrow \mathbf{id}$			$F \rightarrow (E)$		

Fig. 5.24. Parsing table for grammar (5.9).

LL(1) Grammars

- A grammar G is LL(1) iff whenever $A \rightarrow \alpha \mid \beta$ are two distinct productions of G the following conditions hold:
  1. For no terminal a do α and β derive strings beginning with a
  2. At most one of α and β can derive an empty string
  3. If $\beta \Rightarrow^* \in$ then α does not derive any strings beginning with a terminal in FOLLOW(A)

LL(1) Grammars

- A grammar whose parsing table has no multiply defined entries is said to be LL(1)

Not an LL(1)

	a	b	e	i	t	$\$$
S	$S \to a$			$S \to iCtSS'$		
S'			$S' \to \epsilon$ $S' \to eS$			$S' \to \epsilon$
C		$C \to b$				

Fig. 5.26. Parsing table.