

CSc 453

Compilers and Systems Software

8 : Top-Down Parsing III

Department of Computer Science
University of Arizona

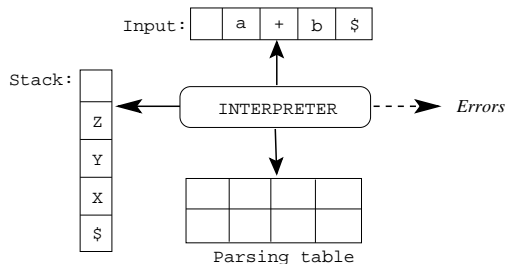
collberg@gmail.com

Copyright © 2009 Christian Collberg

Predictive Parsing

Predictive Parsing

- Just as with lexical analysis, we can either hard-code a top-down parser, or build a generic table-driven interpreter. The latter is called a *Predictive Parser*.
- Instead of using recursion we store the current state of the parse on a stack:



Predictive Parsing. . .

- The predictive parser has
 - ① an input stream (list of tokens followed by the *end-of-file-marker* \$),
 - ② a stack with a sequence of grammar symbols, and an *end-of-stack-marker* \$,
 - ③ a parsing table $M[A, a] \rightarrow P$ mapping a non-terminal A and a terminal a to a grammar production P .
- Initially, the stack holds the start symbol, S .

Predictive Parsing. . .

At each step, the interpreter looks at the top stack element (X) and the current input symbol (a). There are three cases:

- 1 $X = a = \$ \Rightarrow$ *success!*
- 2 $M[X, a] = \text{error} \Rightarrow$ *bail!*
- 3 $X = a \neq \$ \Rightarrow$ *match()*. Move to next token and pop off X .
- 4 $M[X, a] = \{X \rightarrow UVW\} \Rightarrow$
 - 1 Pop X off the stack, then
 - 2 Push(W), Push(V), Push(U).

The next slide shows the algorithm in more detail.

```
a := first token
repeat
  X := top()
  if X is a terminal or $ then
    if X = a then
      pop()
      a := next token
    else error
  else
    if  $M[X, a] = X \rightarrow Y_1 Y_2 \dots Y_k$  then
      pop()
      push( $Y_k$ ); ...; Push( $Y_1$ )
    else error
until X = $
```

- The Predictive parsing table for the grammar below:

	<u>id</u>	<u>+</u>	<u>*</u>	<u>(</u>	<u>)</u>	<u>\$</u>
<u>E</u>	$E \rightarrow TE'$			$E \rightarrow TE'$		
<u>E'</u>		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
<u>T</u>	$T \rightarrow FT'$			$T \rightarrow FT'$		
<u>T'</u>		$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
<u>F</u>	$F \rightarrow \underline{id}$			$F \rightarrow \underline{(E)}$		

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

Stack	Input	$M[A, a] =$
$\$E$	<u>id+id*id</u> \$	
$\$E'T$	<u>id+id*id</u> \$	$E \rightarrow TE'$
$\$E'T'F$	<u>id+id*id</u> \$	$T \rightarrow FT'$
$\$E'T'\underline{id}$	<u>id+id*id</u> \$	$F \rightarrow \underline{id}$
$\$E'T'$	<u>+id*id</u> \$	
$\$E'$	<u>+id*id</u> \$	$T' \rightarrow \epsilon$
$\$E'T\text{+}$	<u>+id*id</u> \$	$E' \rightarrow \text{+}TE'$
$\$E'T$	<u>id*id</u> \$	

Stack	Input	$M[A, a] =$
$\$E'T'F$	<u>id</u> *id\$	$T \rightarrow FT'$
$\$E'T'\underline{id}$	id*id\$	$F \rightarrow \underline{id}$
$\$E'T'$	* <u>id</u> \$	
$\$E'T'F*$	* <u>id</u> \$	$T' \rightarrow *FT'$
$\$E'T'F$	<u>id</u> \$	
$\$E'T'\underline{id}$	<u>id</u> \$	$F \rightarrow \underline{id}$
$\$E'T'$	\$	
$\$E'$	\$	$T' \rightarrow \epsilon$
\$	\$	$E' \rightarrow \epsilon$

Stack	Input	$M[A, a] =$
$\$E$	<u>id</u> +\$	
$\$E'T$	<u>id</u> +\$	$E \rightarrow TE'$
$\$E'T'F$	<u>id</u> +\$	$T \rightarrow FT'$
$\$E'T'\underline{id}$	<u>id</u> +\$	$F \rightarrow \underline{id}$
$\$E'T'$	<u>+</u> +\$	
$\$E'$	<u>+</u> +\$	$T' \rightarrow \epsilon$
$\$E'T\underline{+}$	<u>+</u> +\$	$E' \rightarrow \underline{+}TE'$
$\$E'T$	<u>*</u> +\$	<i>Error!</i>

Building the Parse Table...

```
for each production  $A \rightarrow \alpha$  do
  for each terminal  $a$  in  $\text{FIRST}(\alpha)$  do
     $M[A, a] := \{A \rightarrow \alpha\}$ 

  if  $\epsilon$  is in  $\text{FIRST}(\alpha)$  then
    for each terminal  $b$  in  $\text{FOLLOW}(A)$  do
       $M[A, b] := \{A \rightarrow \alpha\}$ 

  if  $\epsilon$  is in  $\text{FIRST}(\alpha)$  and  $\$$  is in  $\text{FOLLOW}(A)$  then
     $M[A, \$] := \{A \rightarrow \alpha\}$ 

for all undefined entries  $M[A, a]$  do
   $M[A, a] := \mathbf{error}$ 
```

$$E \rightarrow T E'$$

$$E' \rightarrow \underline{+} T E' \mid \epsilon$$

$$T \rightarrow F T'$$

$$T' \rightarrow \underline{*} F T' \mid \epsilon$$

$$F \rightarrow \underline{(} E \underline{)} \mid \underline{id}$$

$$\text{FIRST}(E) = \{ \underline{(}, \underline{id} \}$$

$$\text{FIRST}(E') = \{ \underline{+}, \epsilon \}$$

$$\text{FIRST}(T) = \{ \underline{(}, \underline{id} \}$$

$$\text{FIRST}(T') = \{ \underline{*}, \epsilon \}$$

$$\text{FIRST}(F) = \{ \underline{(}, \underline{id} \}$$

$$\text{FOLLOW}(E) = \{ \underline{)}, \underline{\$} \}$$

$$\text{FOLLOW}(E') = \{ \underline{)}, \underline{\$} \}$$

$$\text{FOLLOW}(T) = \{ \underline{+}, \underline{)}, \underline{\$} \}$$

$$\text{FOLLOW}(T') = \{ \underline{+}, \underline{)}, \underline{\$} \}$$

$$\text{FOLLOW}(F) = \{ \underline{+}, \underline{*}, \underline{)}, \underline{\$} \}$$

Predictive Parsing – Building the Table

	<u>id</u>	<u>+</u>	<u>*</u>	<u>(</u>	<u>)</u>	<u>\$</u>
<u>E</u>	$E \rightarrow TE'$			$E \rightarrow TE'$		
<u>E'</u>						
<u>T</u>						
<u>T'</u>						
<u>F</u>						

- We start by looking at production $E \rightarrow TE'$.
- Since $\text{FIRST}(TE') = \text{FIRST}(T) = \{(\underline{\quad}, \underline{\text{id}})\}$ we set $M[E, (\underline{\quad})] = M[E, \underline{\text{id}}] = E \rightarrow TE'$.

Predictive Parsing – Building the Table...

	<u>id</u>	<u>+</u>	<u>*</u>	<u>(</u>	<u>)</u>	<u>\$</u>
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow \underline{+}TE'$				
T						
T'						
F						

- We next consider production $E' \rightarrow \underline{+}TE'$.
- Since $\text{FIRST}(\underline{+}TE') = \{+\}$ we set $M[E', \underline{+}] = E' \rightarrow \underline{+}TE'$.

Predictive Parsing – Building the Table...

	<u>id</u>	<u>+</u>	<u>*</u>	<u>(</u>	<u>)</u>	<u>\$</u>
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
T						
T'						
F						

- We next consider production $E' \rightarrow \epsilon$.
- Since $\text{FOLLOW}(E') = \{ \underline{)}, \$ \}$ we set
 $M[E', \underline{)}] = M[E', \$] = E' \rightarrow \epsilon$.

Predictive Parsing – Building the Table...

	<u>id</u>	<u>+</u>	<u>*</u>	<u>(</u>	<u>)</u>	<u>\$</u>
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'						
F						

- We next consider production $T \rightarrow FT'$.
- Since $\text{FIRST}(FT') = \{(\underline{), \underline{id}}\}$ we set
 $M[T, (\underline{)]} = M[T, \underline{id}] = T \rightarrow FT'$.

Predictive Parsing – Building the Table...

	<u>id</u>	<u>+</u>	<u>*</u>	<u>(</u>	<u>)</u>	<u>\$</u>
<u>E</u>	$E \rightarrow TE'$			$E \rightarrow TE'$		
<u>E'</u>		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
<u>T</u>	$T \rightarrow FT'$			$T \rightarrow FT'$		
<u>T'</u>			$T' \rightarrow *FT'$			
<u>F</u>						

- We next consider production $T' \rightarrow *FT'$.
- Since $\text{FIRST}(*FT') = \{\underline{*}\}$ we set $M[T', \underline{*}] = T' \rightarrow *FT'$.

Predictive Parsing – Building the Table...

	<u>id</u>	<u>+</u>	<u>*</u>	<u>(</u>	<u>)</u>	<u>\$</u>
<u>E</u>	$E \rightarrow TE'$			$E \rightarrow TE'$		
<u>E'</u>		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
<u>T</u>	$T \rightarrow FT'$			$T \rightarrow FT'$		
<u>T'</u>		$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
<u>F</u>						

- We next consider production $T' \rightarrow \epsilon$.
- Since $\text{FOLLOW}(T') = \{+, _, \$\}$ we set
 $M[T', +] = M[T', _] = M[T', \$] = T' \rightarrow \epsilon$.

Predictive Parsing – Building the Table...

	<u>id</u>	<u>+</u>	<u>*</u>	<u>(</u>	<u>)</u>	<u>\$</u>
<u>E</u>	$E \rightarrow TE'$			$E \rightarrow TE'$		
<u>E'</u>		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
<u>T</u>	$T \rightarrow FT'$			$T \rightarrow FT'$		
<u>T'</u>		$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
<u>F</u>	$F \rightarrow \underline{id}$			$F \rightarrow \underline{(E)}$		

- We next consider production $F \rightarrow \underline{(E)}$.
- Since $\text{FIRST}(\underline{(E)}) = \{(\underline{)}$ we set $M[F, (\underline{)}] = F \rightarrow \underline{(E)}$.
- We finally consider production $F \rightarrow \underline{id}$.
- Since $\text{FIRST}(\underline{id}) = \{\underline{id}\}$ we set $M[F, \underline{id}] = F \rightarrow \underline{id}$.

Parsing Expressions

- Here's a simple implementation of predictive parsing in Java.
- Grammar symbols (both terminals and non-terminals) are represented by single characters: e represents E', f represents F', i represents id, "" represents epsilon, null represents error.
- The function `table(X,a)` looks up the relevant production.

```

class Expr {
    static String[] input = {"i","+","i","*","i","$"};
    static int current = 0;

    static boolean isTerminal(String S) {
        return S.equals("*") | S.equals("+") |
            S.equals("i") |
            S.equals("(") | S.equals(")");
    }

    static String[][] TABLE = {
        //   id   +       *       (       )       $
        {"Te", null, null, "Te", null, null}, // E
        {null, "+Te", null, null, "", ""}, // E'
        {"Ft", null, null, "Ft", null, null}, // T
        {null, "", "*Ft", null, "", ""}, // T'
        {"i", null, null, "(E)", null, null} // F
    };
}

```

```
static String table(String X, String a){
    int aIdx, XIdx=-1;
    switch (a.charAt(0)) {
        case 'i' : aIdx=0; break;
        case '+' : aIdx=1; break;
        case '*' : aIdx=2; break;
        case '(' : aIdx=3; break;
        case ')' : aIdx=4; break;
        case '$' : aIdx=5; break; }
    switch (X.charAt(0)) {
        case 'E' : XIdx=0; break;
        case 'e' : XIdx=1; break;
        case 'T' : XIdx=2; break;
        case 't' : XIdx=3; break;
        case 'F' : XIdx=4; break; }
    return TABLE[XIdx][aIdx];
}
```

```
static boolean interpret(){
    push("$"); push("E");
    String a, X;
    while(true) {
        a = input[current];
        X = top();
        if (isTerminal(X) | X.equals("$")) {
            if (a.equals(X)) {pop(); current++;}
            else return false;
        } else {
            pop();
            String prod = table(X,a);
            if (prod==null) return false;
            for(int i=prod.length()-1;i>=0;i--)
                push(prod.charAt(i));
        }
        if (X.equals("$")) return true;
    }
}
```

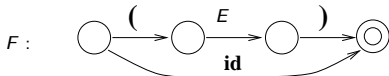
Error Recovery

Error Recovery

- 1 Quit on first error. Considered unacceptable. But, with fast parse-edit-compile-cycles, why not?
- 2 Repair the error by inserting/deleting tokens, to turn the erroneous program into a legal one.
- 3 Repair the error by guessing what the programmer meant.

_____ Simple error handling: _____

- 1 Skip tokens until a *synchronizing* token is found.
- 2 The tokens in FOLLOW(*A*) are in the SYNCH set.
- 3 If the language has an hierarchical structure
prog → proc → **stat** → **expr**, then add FIRST sets of higher constructs to the SYNCH sets of lower ones.



```

PROCEDURE F();
  CONST SYNCH = FOLLOW(F) ∪ FIRST(stat);
  IF cur_tok = ( THEN
    cur_tok := next_token(); E();
    IF cur_tok = ) THEN
      cur_tok := next_token()
    ELSE WHILE cur_tok ∉ SYNCH DO
      cur_tok := next_token() ENDDO
    ENDIF
  ELSIF cur_tok = id THEN cur_tok := next_token();
  ELSE WHILE cur_tok ∉ SYNCH DO
    cur_tok := next_token() ENDDO
  ENDIF;

```

Example I

$Z \rightarrow d$	$Y \rightarrow c$
$Z \rightarrow XYZ$	$X \rightarrow Y$
$Y \rightarrow \epsilon$	$X \rightarrow a$

round	FIRST(Z)	FIRST(Y)	FIRST(X)
1	d	c	a
2	d	c, ϵ	a, c
3	d, a	c, ϵ	a, c, ϵ
4	d, a, c	c, ϵ	a, c, ϵ
5	d, a, c	c, ϵ	a, c, ϵ

Example 1...

$Z \rightarrow d$	$Y \rightarrow c$
$Z \rightarrow XYZ$	$X \rightarrow Y$
$Y \rightarrow \epsilon$	$X \rightarrow a$

round	FOLLOW(Z)	FOLLOW(Y)	FOLLOW(X)
0	\$		
1	\$	d, a, c	c
2	\$	d, a, c	$c, d, a, \$$

Example 1...

PRODUCTION $Z \rightarrow d$:

$\text{FIRST}(d) = \{d\}$

$\Rightarrow M[Z, d] = Z \rightarrow d$

PRODUCTION $Z \rightarrow X, Y, Z$:

$\text{FIRST}(XYZ) = \{a, c, d\}$

$\Rightarrow M[Z, a] = Z \rightarrow XYZ$

$\Rightarrow M[Z, c] = Z \rightarrow XYZ$

$\Rightarrow M[Z, d] = Z \rightarrow XYZ$

PRODUCTION $Y \rightarrow c$:

$\text{FIRST}(c) = \{c\}$

$\Rightarrow M[Y, c] = Y \rightarrow c$

Example 1...

PRODUCTION $Y \rightarrow \epsilon$:

$\text{FIRST}(\epsilon) = \{\epsilon\}$

$\text{FOLLOW}(Y) = \{d, a, c\}$

$\Rightarrow M[Y, d] = Y \rightarrow \epsilon$

$\Rightarrow M[Y, a] = Y \rightarrow \epsilon$

$\Rightarrow M[Y, c] = Y \rightarrow \epsilon$

PRODUCTION $X \rightarrow a$:

$\text{FIRST}(a) = \{a\}$

$\Rightarrow M[X, a] = X \rightarrow a$

Example 1. . .

PRODUCTION $X \rightarrow Y$:

$\text{FIRST}(Y) = \{c, \epsilon\}$

$\text{FOLLOW}(Y) = \{d, a, c\}$

$\Rightarrow M[X, c] = X \rightarrow Y$

$\Rightarrow M[X, d] = X \rightarrow Y$

$\Rightarrow M[X, a] = X \rightarrow Y$

$\Rightarrow M[X, \$] = X \rightarrow Y$

- Note that this grammar is not LL(1): it has multiple entries in some table cells.
- The grammar is, in fact, ambiguous. The string "d" can be derived in more than one way.

Example II

$S \rightarrow ABC$	$B \rightarrow b$
$A \rightarrow aA C$	$C \rightarrow c$

#	FIRST(S)	FIRST(A)	FIRST(B)	FIRST(C)
0		a	b	c
1	a	a, c	b	c
2	a, c	a, c	b	c
#	FOLLOW(S)	FOLLOW(A)	FOLLOW(B)	FOLLOW(C)
0	$\$$	b	c	
1	$\$$	b	c	$\$, b$

Example II. . .

PRODUCTION $S \rightarrow ABC$:

$\text{FIRST}(ABC) = \{a, c\}$

$\Rightarrow M[S, a] = S \rightarrow ABC$

$\Rightarrow M[S, b] = S \rightarrow ABC$

PRODUCTION $A \rightarrow aA$:

$\text{FIRST}(aA) = \{a\}$

$\Rightarrow M[A, a] = A \rightarrow aA$

PRODUCTION $A \rightarrow C$:

$\text{FIRST}(C) = \{c\}$

$\Rightarrow M[A, c] = A \rightarrow C$

PRODUCTION $B \rightarrow b$:

$\text{FIRST}(b) = \{b\}$

$\Rightarrow M[B, b] = B \rightarrow b$

PRODUCTION $C \rightarrow c$:

$\text{FIRST}(c) = \{c\}$

$\Rightarrow M[C, c] = C \rightarrow c$

Example II. . .

	a	b	c	$\$$
S	$S \rightarrow ABC$	$S \rightarrow ABC$		
A	$A \rightarrow aA$		$A \rightarrow C$	
B		$B \rightarrow b$		
C			$C \rightarrow c$	

Readings and References

- Read Louden, pp. 143–196.
- Or, the Dragon Book:
 - Top-Down Parsing 181–190
 - Error Recovery 192–195
 - Recursive Descent Parsing 40–55, 75–76