CSc 453 — Compilers and Systems Software

8 : Top-Down Parsing III

Christian Collberg
Department of Computer Science
University of Arizona
collberg@gmail.com

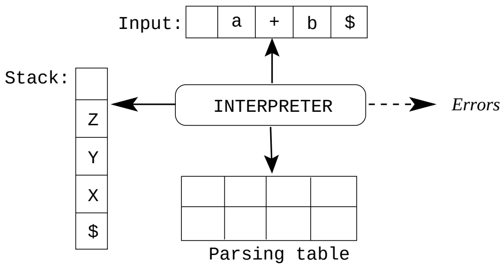
October 12, 2009

1

Predictive Parsing

2 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:



3 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.
- Initally, the stack holds the start symbol, S.

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

The next slide shows the algorithm in more detail.

5

```
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 → Y1 Y2 ··· Yk then
         pop()
         push(Yk); ···; Push(Y1)
      else error
until X = $
```

6

- The Predictive parsing table for the grammar below:

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

$E \rightarrow T\,E'$ $E' \rightarrow \underline{\texttt{+}}\,T\,E' \mid \epsilon$	$T \rightarrow F\,T'$ $T' \rightarrow \underline{\texttt{*}}\,F\,T' \mid \epsilon$	$F \rightarrow$

7

Stack	**Input**	$M[A,a] =$
$\$E$	id+id*id$	
$\$E'T$	id+id*id$	$E \to TE'$
$\$E'T'F$	id+id*id$	$T \to FT'$
$\$E'T'$id	id+id*id$	$F \to$ id
$\$E'T'$	+id*id$	
$\$E'$	+id*id$	$T' \to \epsilon$
$\$E'T$+	+id*id$	$E' \to$ +TE'
$\$E'T$	id*id$	

8

Stack	**Input**	$M[A,a] =$
$\$E'T'F$	id*id$	$T \to FT'$
$\$E'T'$id	id*id$	$F \to$ id
$\$E'T'$	*id$	
$\$E'T'F$*	*id$	$T' \to$ *FT'
$\$E'T'F$	id$	
$\$E'T'$id	id$	$F \to$ id
$\$E'T'$	$	
$\$E'$	$	$T' \to \epsilon$
$	$	$E' \to \epsilon$

9

Stack	**Input**	$M[A,a] =$
$\$E$	id+*$	
$\$E'T$	id+*$	$E \to TE'$
$\$E'T'F$	id+*$	$T \to FT'$
$\$E'T'$id	id+*$	$F \to$ id
$\$E'T'$	+*$	
$\$E'$	+*$	$T' \to \epsilon$
$\$E'T$+	+*$	$E' \to$ +TE'
$\$E'T$	*$	*Error!*

10 Building the Parse Table...

```
for each production A → α do
   for each terminal a in FIRST(α) do
      M[A,a] := {A → α}

   if ϵ is in FIRST(α) then
      for each terminal b in FOLLOW(A) do
         M[A,b] := {A → α}

   if ϵ is in FIRST(α) and $ is in FOLLOW(A) then
      M[A,$] := {A → α}

for all undefined entries M[A,a] do
   M[A,a] := error
```

11

$$
\begin{array}{lcl}
E & \rightarrow & T\ E' \\
E' & \rightarrow & \underline{\texttt{+}}\ T\ E' \mid \epsilon \\
T & \rightarrow & F\ T' \\
T' & \rightarrow & \underline{\texttt{*}}\ F\ T' \mid \epsilon \\
F & \rightarrow & \underline{\texttt{(}}\ E\ \underline{\texttt{)}} \mid \underline{\texttt{id}}
\end{array}
$$

$$
\begin{array}{rcl|rcl}
\texttt{FIRST}(E) & = & \{\underline{\texttt{(}}, \underline{\texttt{id}}\} & \texttt{FOLLOW}(E) & = & \{\underline{\texttt{)}}, \underline{\texttt{\$}}\} \\
\texttt{FIRST}(E') & = & \{\underline{\texttt{+}}, \epsilon\} & \texttt{FOLLOW}(E') & = & \{\underline{\texttt{)}}, \underline{\texttt{\$}}\} \\
\texttt{FIRST}(T) & = & \{\underline{\texttt{(}}, \underline{\texttt{id}}\} & \texttt{FOLLOW}(T) & = & \{\underline{\texttt{+}}, \underline{\texttt{)}}, \underline{\texttt{\$}}\} \\
\texttt{FIRST}(T') & = & \{\underline{\texttt{*}}, \epsilon\} & \texttt{FOLLOW}(T') & = & \{\underline{\texttt{+}}, \underline{\texttt{)}}, \underline{\texttt{\$}}\} \\
\texttt{FIRST}(F) & = & \{\underline{\texttt{(}}, \underline{\texttt{id}}\} & \texttt{FOLLOW}(F) & = & \{\underline{\texttt{+}}, \underline{\texttt{*}}, \underline{\texttt{)}}, \underline{\texttt{\$}}\}
\end{array}
$$

12 Predictive Parsing – Building the Table

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

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

13 Predictive Parsing – Building the Table...

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

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

14 Predictive Parsing – Building the Table...

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

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

15 Predictive Parsing – Building the Table...

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

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

16 Predictive Parsing – Building the Table...

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

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

17 Predictive Parsing – Building the Table...

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

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

18 Predictive Parsing – Building the Table...

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

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

19 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.

20

```
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
    };
```

21

```
    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];
    }
```

22

```
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;
}}
```

23

Error Recovery

24 Error Recovery

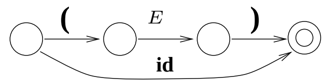
1. Quit on first error. Considered unacceptable. But, with fast parse-edit-compile-cycles, why not?
2. Repair the error by inserting/deleting tokens, to turn the erroneous program into a legal one.
3. Repair the error by guessing what the programmer meant.

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 $\rightarrow$ proc $\rightarrow$ stat $\rightarrow$ expr, then add FIRST sets of higher constructs to the SYNCH sets of lower ones.

25

F :




```
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;
```

26 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

27 Example I...

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

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

28 Example I...

$$\begin{array}{l}
\texttt{PRODUCTION } Z \rightarrow d\texttt{:} \\
\quad \texttt{FIRST(}d\texttt{)==}\{d\} \\
\qquad \Rightarrow M[Z,d] = Z \rightarrow d \\
\\
\texttt{PRODUCTION } Z \rightarrow X,Y,Z\texttt{:} \\
\quad \texttt{FIRST(}XYZ\texttt{)==}\{a,c,d\} \\
\qquad \Rightarrow M[Z,a] = Z \rightarrow XYZ \\
\qquad \Rightarrow M[Z,c] = Z \rightarrow XYZ \\
\qquad \Rightarrow M[Z,d] = Z \rightarrow XYZ \\
\\
\texttt{PRODUCTION } Y \rightarrow c\texttt{:} \\
\quad \texttt{FIRST(}c\texttt{)==}\{c\} \\
\qquad \Rightarrow M[Y,c] = Y \rightarrow c
\end{array}$$

29 Example I...

$$\begin{array}{l}
\texttt{PRODUCTION } Y \rightarrow \epsilon\texttt{:} \\
\quad \texttt{FIRST(}\epsilon\texttt{)==}\{\epsilon\} \\
\quad \texttt{FOLLOW(}Y\texttt{)=}\{d,a,c\} \\
\qquad \Rightarrow M[Y,d] = Y \rightarrow \epsilon \\
\qquad \Rightarrow M[Y,a] = Y \rightarrow \epsilon \\
\qquad \Rightarrow M[Y,c] = Y \rightarrow \epsilon \\
\\
\texttt{PRODUCTION } X \rightarrow a\texttt{:} \\
\quad \texttt{FIRST(}a\texttt{)==}\{a\} \\
\qquad \Rightarrow M[X,a] = X \rightarrow a
\end{array}$$

30 Example I...

$$\begin{array}{l}
\texttt{PRODUCTION } X \rightarrow Y\texttt{:} \\
\quad \texttt{FIRST(}Y\texttt{)==}\{c,\epsilon\} \\
\quad \texttt{FOLLOW(}Y\texttt{)=}\{d,a,c\} \\
\qquad \Rightarrow M[X,c] = X \rightarrow Y \\
\qquad \Rightarrow M[X,d] = X \rightarrow Y \\
\qquad \Rightarrow M[X,a] = X \rightarrow Y \\
\qquad \Rightarrow M[X,\$] = X \rightarrow Y
\end{array}$$

- 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.

31 Example II

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

#	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	$\$, b$

32 Example II...

`PRODUCTION` $S \to ABC$:
  `FIRST(`ABC`)==`$\{a, c\}$
    $\Rightarrow M[S, a] = S \to ABC$
    $\Rightarrow M[S, b] = S \to ABC$

`PRODUCTION` $A \to aA$:
  `FIRST(`aA`)==`$\{a\}$
    $\Rightarrow M[A, a] = A \to aA$

`PRODUCTION` $A \to C$:
  `FIRST(`C`)==`$\{c\}$
    $\Rightarrow M[A, c] = A \to C$

`PRODUCTION` $B \to b$:
  `FIRST(`b`)==`$\{b\}$
    $\Rightarrow M[B, b] = B \to b$

`PRODUCTION` $C \to c$:
  `FIRST(`c`)==`$\{c\}$
    $\Rightarrow M[C, c] = C \to c$

33 Example II...

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

34 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