CSc 453

Compilers and Systems Software

8 : Top-Down Parsing III

Department of Computer Science
University of Arizona

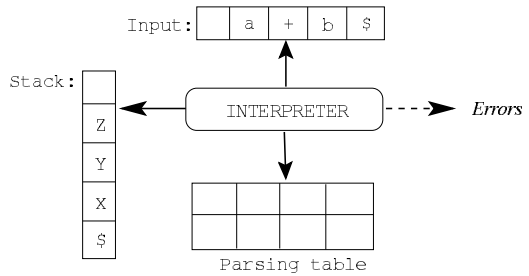
collberg@gmail.com

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.
- Initally, 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. $M[X, a] = error \Rightarrow$ *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 → Y1 Y2 ··· Yk then
         pop()
         push(Yk); ···; Push(Y1)
      else error
until X = $
```

- The Predictive parsing table for the grammar below:

	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	$F \rightarrow$ id			$F \rightarrow (E)$		

$E \rightarrow T\ E'$ $E' \rightarrow +\ T\ E' \mid \epsilon$	$T \rightarrow F\ T'$ $T' \rightarrow *\ F\ T' \mid \epsilon$	$F \rightarrow (\ E\ ) \mid$ id

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

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

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

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
```

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

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

Predictive Parsing – Building the Table

	id	+	*	(	)	$
E	$E \to TE'$			$E \to TE'$		
E'						
T						
T'						
F						

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

Predictive Parsing – Building the Table...

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

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

Predictive Parsing – Building the Table...

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

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

Predictive Parsing – Building the Table...

	id	+	*	(	)	$
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 $M[T, \underline{(}] = M[T, \underline{\texttt{id}}] = T \to FT'$.

Predictive Parsing – Building the Table...

	id	+	*	(	)	$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \epsilon$	$E' \to \epsilon$
T	$T \to FT'$			$T \to FT'$		
T'			$T' \to *FT'$			
F						

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

Predictive Parsing – Building the Table...

	id	+	*	(	)	$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \epsilon$	$E' \to \epsilon$
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 $\texttt{FOLLOW}(T') = \{+,), \$\}$ we set $M[T', +] = M[T',)] = M[T', \$] = T' \to \epsilon$.

Predictive Parsing – Building the Table...

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

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

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;
}}
```

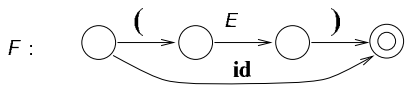
Error Recovery

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

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

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 I...

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

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

Example I...

PRODUCTION $Z \rightarrow d$:
FIRST(d)==$\{d\}$
$\Rightarrow M[Z, d] = Z \rightarrow d$

PRODUCTION $Z \rightarrow X, Y, Z$:
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 $Y \rightarrow c$:
FIRST(c)==$\{c\}$
$\Rightarrow M[Y, c] = Y \rightarrow c$

Example I...

PRODUCTION $Y \rightarrow \epsilon$:
FIRST(ϵ)==$\{\epsilon\}$
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 $X \rightarrow a$:
FIRST(a)==$\{a\}$
$\Rightarrow M[X, a] = X \rightarrow a$

Example I...

PRODUCTION $X \rightarrow Y$:
FIRST(Y)==$\{c, \epsilon\}$
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

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

#	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

Example II...

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

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

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

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

PRODUCTION $C \rightarrow c$:
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