CSc 453

Compilers and Systems Software

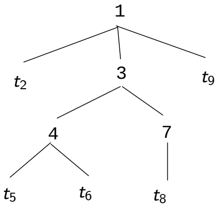
7 : Top-Down Parsing II

Department of Computer Science
University of Arizona

collberg@gmail.com

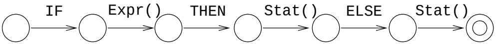
Top-Down Parsing

- The parse tree is constructed
  - From the top
  - From left to right.
- The terminals are seen in order of appearance in the token stream $(t_2, t_5, t_6, t_8, t_9)$:



Building The LL(1) Parser

1. Remove left recursion.
2. Left factor the grammar.
3. Construct transition diagrams for each production:

   1. Compute `FIRST(`A`)` for each grammar symbol A.
   2. Compute `FOLLOW(`A`)` for each nonterminal A.
4. Simplify the transition diagrams.
5. Construct the recursive procedures.

FIRST Sets

- FIRST(α) is the set of terminals that begin strings derived from α.
- FIRST-sets help us solve problem 3.

$$
\begin{array}{lcl}
\text{prog} & \rightarrow & \text{decl} \mid \text{stat} \\
\text{stat} & \rightarrow & \underline{\texttt{if}} \cdots \mid \underline{\texttt{id}}\,\underline{\texttt{()}} \mid \underline{\texttt{while}} \cdots \\
\text{decl} & \rightarrow & \underline{\texttt{int}}\ \underline{\texttt{id}} \mid \underline{\texttt{real}}\ \underline{\texttt{id}}
\end{array}
$$

$$
\begin{array}{lcl}
\texttt{FIRST}(\text{stat}) & = & \{\underline{\texttt{if}}, \underline{\texttt{id}}, \underline{\texttt{while}}\} \\
\texttt{FIRST}(\text{decl}) & = & \{\underline{\texttt{int}}, \underline{\texttt{real}}\}
\end{array}
$$

FIRST Sets...

```
PROCEDURE prog ();
  IF curr_tok ∈ {if, id, while} THEN stat();
  ELSIF curr_tok ∈ {int, real} THEN decl()
  ELSE syntax error ENDIF;
END;
PROCEDURE stat (); ··· END;
PROCEDURE decl (); ··· END;
```

$$\begin{aligned} \mathrm{FIRST}(\text{stat}) &= \{\underline{\texttt{if}}, \underline{\texttt{id}}, \underline{\texttt{while}}\} \\ \mathrm{FIRST}(\text{decl}) &= \{\underline{\texttt{int}}, \underline{\texttt{real}}\} \end{aligned}$$

FIRST Sets...

- $\mathtt{FIRST}(\alpha)$ is the set of terminals that begin strings derived from α, ie. $\mathtt{FIRST}(\alpha) = \{\underline{\mathsf{a}} \mid \underline{\mathsf{a}} \text{ a terminal}, \ \alpha \stackrel{*}{\Rightarrow} \underline{\mathsf{a}}\beta\}$.

Example Grammar		
$E \rightarrow T\ E'$ $E' \rightarrow \underline{+}\ T\ E' \mid \epsilon$	$T \rightarrow F\ T'$ $T' \rightarrow \underline{*}\ F\ T' \mid \epsilon$	$F \rightarrow \underline{(}\ E\ \underline{)} \mid \underline{\mathtt{id}}$

FIRST sets:

$$\begin{aligned} \mathtt{FIRST}(E) &= \{\underline{(}, \underline{\mathtt{id}}\} \\ \mathtt{FIRST}(T) &= \{\underline{(}, \underline{\mathtt{id}}\} \\ \mathtt{FIRST}(F) &= \{\underline{(}, \underline{\mathtt{id}}\} \end{aligned}$$

$$\begin{aligned} \mathtt{FIRST}(E') &= \{\underline{+}, \epsilon\} \\ \mathtt{FIRST}(T') &= \{\underline{*}, \epsilon\} \end{aligned}$$

Computing FIRST(A)

```
REPEAT until no more changes:
 1.IF A is a terminal THEN FIRST(A) = {A}.
 2.IF A is a nonterminal, and there
     is a production A → ε THEN ε is in FIRST(A).
 3.IF A is a nonterminal, and there
     is a production A → Y_1 ··· Y_k THEN
     FOR i := 1 to k − 1 DO
       IF ε ∈ FIRST(Y_1) ∧ ··· ∧ ε ∈ FIRST(Y_i)
         and a (≠ ε) ∈ FIRST(Y_{i+1}) THEN a is in FIRST(A);
     IF ε ∈ FIRST(Y_1) ∧ ··· ∧ ε ∈ FIRST(Y_k) THEN
       ε is in FIRST(A);
```

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

1. $\texttt{FIRST}(\underline{+}) = \{\underline{+}\}$, $\texttt{FIRST}(\underline{*}) = \{\underline{*}\}$, etc., because of point 1.
2. $\epsilon \in \texttt{FIRST}(E')$, $\epsilon \in \texttt{FIRST}(T')$, because of point 2.
3. $\underline{(}, \underline{\texttt{id}} \in \texttt{FIRST}(F)$, because of point 3.
4. $\underline{(}, \underline{\texttt{id}} \in \texttt{FIRST}(T)$, because $\underline{(}, \underline{\texttt{id}} \in \texttt{FIRST}(F)$, and $T \rightarrow F\ T'$.
5. $\underline{(}, \underline{\texttt{id}} \in \texttt{FIRST}(E)$, because $\underline{(}, \underline{\texttt{id}} \in \texttt{FIRST}(T)$, and $E \rightarrow T\ E'$.
6. $\underline{+} \in \texttt{FIRST}(E')$, $\underline{*} \in \texttt{FIRST}(T')$, because $E' \rightarrow \underline{+}\ T\ E'$ and $T' \rightarrow \underline{*}\ F\ T'$.

FOLLOW Sets

- We let \$ symbolize *end-of-input*.
- $\texttt{FOLLOW}(A)$ is the set of terminals that can follow right after the nonterminal A in some sentential form.
  $\texttt{FOLLOW}(A) = \{\underline{\mathtt{a}} \mid \underline{\mathtt{a}} \text{ a terminal, } S \stackrel{*}{\Rightarrow} \alpha A \underline{\mathtt{a}} \beta\}$.
- $\$ \in \texttt{FOLLOW}(A)$ if A is the rightmost symbol in a sentential form, i.e. $S \stackrel{*}{\Rightarrow} \alpha A$.

FOLLOW Sets...

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

$$\begin{array}{rcl} \mathtt{FOLLOW}(E) & = & \{\underline{)}, \underline{\$}\} \\ \mathtt{FOLLOW}(T) & = & \{\underline{+}, \underline{)}, \underline{\$}\} \\ \mathtt{FOLLOW}(F) & = & \{\underline{+}, \underline{*}, \underline{)}, \underline{\$}\} \end{array}$$

$$\begin{array}{rcl} \mathtt{FOLLOW}(F) & = & \{\underline{+}, \underline{*}, \underline{)}, \underline{\$}\} \\ \mathtt{FOLLOW}(E') & = & \{\underline{)}, \underline{\$}\}\} \\ \mathtt{FOLLOW}(T') & = & \{\underline{+}, \underline{)}, \underline{\$}\}\} \end{array}$$

FOLLOW Sets...

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

- $\underline{)}$ is in FOLLOW(E), because

$$E \Rightarrow TE' \Rightarrow FT'E' \Rightarrow \underline{(}E\boxed{\underline{)}}T'E'.$$

- $\underline{+}$ is in FOLLOW(T), because

$$E \Rightarrow TE' \Rightarrow T\boxed{\underline{+}}TE'.$$

FOLLOW Sets...

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

- $\underline{*}$ is in FOLLOW(F), because

$$E \Rightarrow TE' \Rightarrow FT'E' \Rightarrow F\boxed{\underline{*}}FT'E'.$$

- $\underline{)}$ is in FOLLOW(F), because

$$E \Rightarrow TE' \Rightarrow FT'E' \Rightarrow \underline{(}E\underline{)}T'E' \Rightarrow$$

$$\underline{(}TE'\underline{)}T'E' \Rightarrow \underline{(}FT'E'\underline{)}T'E' \Rightarrow$$

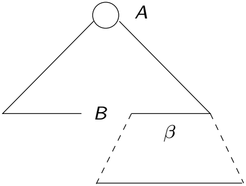
$$\underline{(}FE'\underline{)}T'E' \Rightarrow \underline{(}F\boxed{\underline{)}}T'E'.$$

Computing FOLLOW Sets

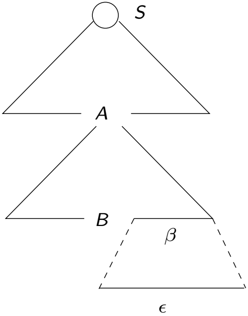
- Let S be the start symbol and $\underline{\$}$ the *end-of-file* marker.

```
REPEAT until no more changes:
 1. Add $ to FOLLOW(S).

 2. IF there is a production A → αBβ THEN
    Add everything in FIRST(β) (except ε) to FOLLOW(B).

 3. IF there is a production A → αB OR
      a production A → αBβ where ε ∈ FIRST(β) THEN
      Add everything in FOLLOW(A) to FOLLOW(B).
```

Computing FOLLOW Sets...



$\texttt{FIRST}(\beta) \subseteq \texttt{FOLLOW}(B)$



$\texttt{FOLLOW}(A) \subseteq \texttt{FOLLOW}(B)$

LL(1) Grammars

- A grammar is LL(1) if we can construct a recursive descent parser that handles it (without using backtracking).
- LL(1) stands for
  - The input is scanned from $\boxed{\text{L}}$eft-to-right.
  - The parse produces a $\boxed{\text{L}}$eftmost derivation.
  - We have $\boxed{1}$-token lookahead.

LL(1) Grammars...

- Formal Definition:

  A grammar is LL(1) iff for any two productions

  $$A \rightarrow \alpha \mid \beta$$

  the following conditions hold

  1. $\mathtt{FIRST}(\alpha) \cap \mathtt{FIRST}(\beta) = \emptyset$
  2. *If* $\beta \stackrel{*}{\Rightarrow} \epsilon$ *then* $\mathtt{FIRST}(\alpha) \cap \mathtt{FOLLOW}(A) = \emptyset$

Recursive Descent Parsers

FOR each non-terminal A DO

    create initial $\bigcirc$ and final $\circledcirc$ states;

FOR each production $A \rightarrow X_1 \cdots X_n$ DO

    create a path A's initial to A's final node

    with edges labeled $X_1 \cdots X_n$:

$$\bigcirc \xrightarrow{X_1} \bigcirc \xrightarrow{X_2} \bigcirc \xrightarrow{\cdots} \bigcirc \xrightarrow{X_n} \circledcirc \;;$$

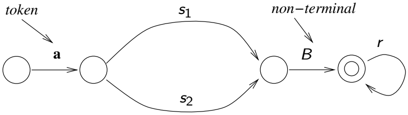
Simplify the diagrams;

FOR each transition diagram P DO

    Create a procedure P that ‘‘traverses’’ the diagram

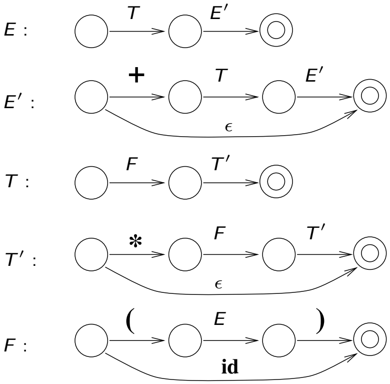
    guided by the input.

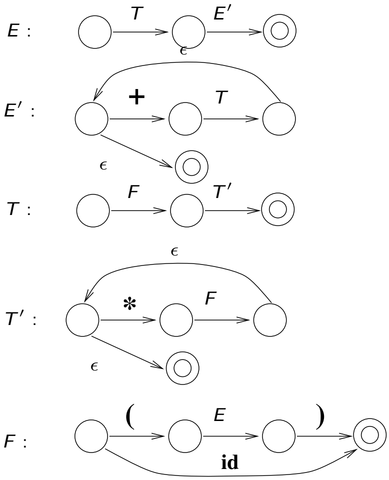
Recursive Descent Parsers. . .



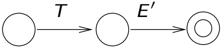

```
PROCEDURE P();
   IF curr_tok = a THEN curr_tok := next_token()
   ELSE syntax_error; ENDIF
   IF curr_tok ∈ FIRST(s1) THEN code for parsing s1
   ELSIF curr_tok ∈ FIRST(s2) THEN code for parsing s2
   ELSE syntax_error; ENDIF
   B();
   WHILE curr_tok ∈ FIRST(r) DO code for parsing r ENDDO
END P;
```

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

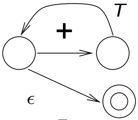



E :
T
E′
ε
E′ :
+
T
ε
T :
F
T′
ε
T′ :
*
F
ε
F :
(
E
)
id

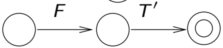
E :



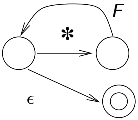
E' :



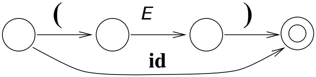
T :



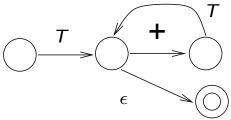
T' :



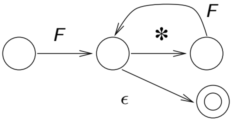
F :



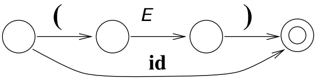
E :



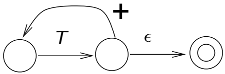
T :



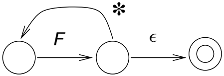
F :



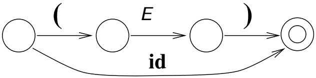
E :



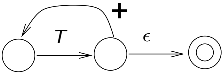
T :



F :

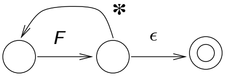


E :



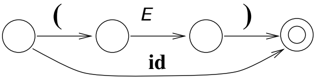

```
PROCEDURE E();
   LOOP
      T();
      IF cur_tok = + THEN
         cur_tok := next_token()
      ELSE EXIT ENDIF
   ENDLOOP;
```

T :




```
PROCEDURE T();
   LOOP
      F();
      IF cur_tok = * THEN
         cur_tok := next_token()
      ELSE EXIT ENDIF
   ENDLOOP;
```

F :




```
PROCEDURE F();
   IF cur_tok = ( THEN
      cur_tok := next_token();
      E();
      IF cur_tok = ) THEN
         cur_tok := next_token()
      ELSE syntax error ENDIF
   ELSIF cur_tok = id THEN
      cur_tok := next_token();
   ELSE syntax error ENDIF
```

Parsing Expressions

- Here's the parser coded in Java.

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

    static boolean lookahead(String S) {
       return input[current].equals(S);
    }

    static void match(String S) throws Exception {
        if   (input[current].equals(S)) current++;
        else throw new Exception();
    }
```

```
static void E() throws Exception {
    T(); while(lookahead("+")) {match("+"); T();}
}

static void T() throws Exception {
    F(); while(lookahead("*")) {match("*"); F();}
}

static void F() throws Exception {
    if (lookahead("(")) {match("("); E(); match(")");
    } else match("i");
}

public static void main(String[] args) {
   try {E(); System.out.println("true");
   } catch (Exception e)
       {System.out.println("false");}
}
```

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