CSc 453

Compilers and Systems Software

7 : Top-Down Parsing II

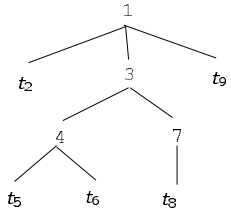
Department of Computer Science
University of Arizona

collberg@gmail.com

Copyright © 2009 Christian Collberg

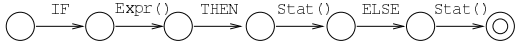
## Top-Down Parsing

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

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

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

## 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;
```

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

## FIRST Sets...

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

| Example Grammar | | |
|---|---|---|
| $E \rightarrow T\ E'$<br>$E' \rightarrow \underline{+}\ T\ E' \mid \epsilon$ | $T \rightarrow F\ T'$<br>$T' \rightarrow \underline{*}\ F\ T' \mid \epsilon$ | $F \rightarrow \underline{(}\ E\ \underline{)} \mid \underline{\text{id}}$ |

FIRST sets:

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

## 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 → Y1 ··· Yk THEN
     FOR i := 1 to k − 1 DO
       IF ϵ ∈ FIRST(Y1) ∧ ··· ∧ ϵ ∈ FIRST(Yi)
         and a (≠ ϵ) ∈ FIRST(Yi+1) THEN a is in FIRST(A);
     IF ϵ ∈ FIRST(Y1) ∧ ··· ∧ ϵ ∈ FIRST(Yk) THEN
       ϵ is in FIRST(A);
```

| | | |
|---|---|---|
| $E \rightarrow T\ E'$<br>$E' \rightarrow \underline{+}\ T\ E' \mid \epsilon$ | $T \rightarrow F\ T'$<br>$T' \rightarrow \underline{*}\ F\ T' \mid \epsilon$ | $F \rightarrow \underline{(}\ E\ \underline{)} \mid \underline{\text{id}}$ |

1. $\text{FIRST}(\underline{+}) = \{\underline{+}\}$, $\text{FIRST}(\underline{*}) = \{\underline{*}\}$, etc., because of point 1.
2. $\epsilon \in \text{FIRST}(E')$, $\epsilon \in \text{FIRST}(T')$, because of point 2.
3. $\underline{(}, \underline{\text{id}} \in \text{FIRST}(F)$, because of point 3.
4. $\underline{(}, \underline{\text{id}} \in \text{FIRST}(T)$, because $\underline{(}, \underline{\text{id}} \in \text{FIRST}(F)$, and $T \rightarrow F\ T'$.
5. $\underline{(}, \underline{\text{id}} \in \text{FIRST}(E)$, because $\underline{(}, \underline{\text{id}} \in \text{FIRST}(T)$, and $E \rightarrow T\ E'$.
6. $\underline{+} \in \text{FIRST}(E')$, $\underline{*} \in \text{FIRST}(T')$, 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.
  $\texttt{FOLLOW}(A) = \{\underline{\mathtt{a}} \mid \underline{\mathtt{a}}\ \mathrm{a\ terminal},\ S \stackrel{*}{\Rightarrow} \alpha A \underline{\mathtt{a}} \beta\}$.
- $\$ \in \texttt{FOLLOW}(A)$ if $A$ is the rightmost symbol in a sentential form, i.e. $S \stackrel{*}{\Rightarrow} \alpha A$.

## FOLLOW Sets...

| | | |
|---|---|---|
| $E \to T\ E'$<br>$E' \to \underline{+}\ T\ E' \mid \epsilon$ | $T \to F\ T'$<br>$T' \to \underline{*}\ F\ T' \mid \epsilon$ | $F \to \underline{(}\ E\ \underline{)} \mid \underline{\mathtt{id}}$ |

| | | | |
|---|---|---|---|
| $\texttt{FOLLOW}(E)$ | $= \{\underline{)}, \$\}$ | $\texttt{FOLLOW}(F)$ | $= \{\underline{+}, \underline{*}, \underline{)}, \$\}$ |
| $\texttt{FOLLOW}(T)$ | $= \{\underline{+}, \underline{)}, \$\}$ | $\texttt{FOLLOW}(E')$ | $= \{\underline{)}, \$\}\}$ |
| $\texttt{FOLLOW}(F)$ | $= \{\underline{+}, \underline{*}, \underline{)}, \$\}$ | $\texttt{FOLLOW}(T')$ | $= \{\underline{+}, \underline{)}, \$\}\}$ |

## FOLLOW Sets...

| | | |
|---|---|---|
| $E \to T\ E'$<br>$E' \to \underline{+}\ T\ E' \mid \epsilon$ | $T \to F\ T'$<br>$T' \to \underline{*}\ F\ T' \mid \epsilon$ | $F \to \underline{(}\ E\ \underline{)} \mid \underline{\mathtt{id}}$ |

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

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

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

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

## FOLLOW Sets...

| | | |
|---|---|---|
| $E \to T\ E'$<br>$E' \to \underline{+}\ T\ E' \mid \epsilon$ | $T \to F\ T'$<br>$T' \to \underline{*}\ F\ T' \mid \epsilon$ | $F \to \underline{(}\ E\ \underline{)} \mid \underline{\mathtt{id}}$ |

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

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

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

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

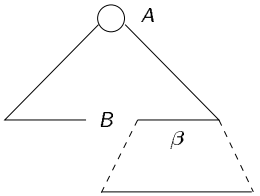
## Computing FOLLOW Sets

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

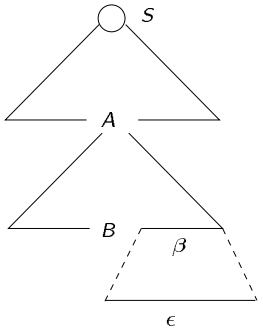
REPEAT until no more changes:

1. Add $\underline{\$}$ to FOLLOW($S$).
2. IF there is a production $A \rightarrow \alpha B \beta$ THEN
   Add everything in FIRST($\beta$) (except $\epsilon$) to FOLLOW($B$).
3. IF there is a production $A \rightarrow \alpha B$ OR
   a production $A \rightarrow \alpha B \beta$ where $\epsilon \in$ FIRST($\beta$) THEN
   Add everything in FOLLOW($A$) to FOLLOW($B$).

## Computing FOLLOW Sets...




FIRST($\beta$) $\subseteq$ FOLLOW($B$)




FOLLOW($A$) $\subseteq$ 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 L eft-to-right.
  - The parse produces a L eftmost derivation.
  - We have 1 -token lookahead.

## LL(1) Grammars...

- Formal Definition:

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

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

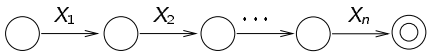
  *the following conditions hold*

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

## Recursive Descent Parsers

FOR each non-terminal $A$ DO
  create initial ◯ and final ◎ 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 $X_1 \cdots X_n$:

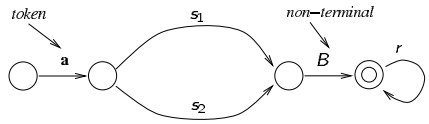



;

Simplify the diagrams;

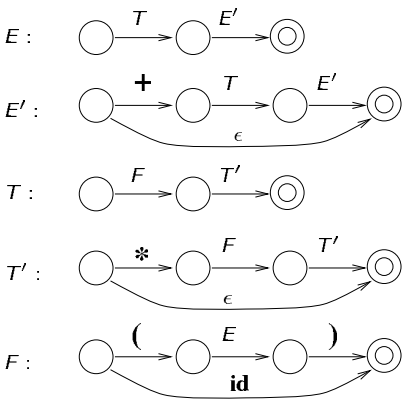
FOR each transition diagram $P$ DO
  Create a procedure $P$ that ''traverses'' the diagram
  guided by the input.

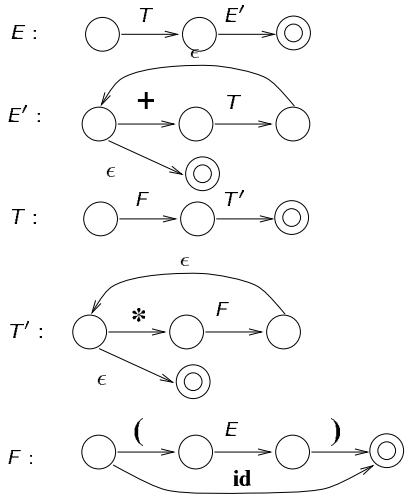
## Recursive Descent Parsers...

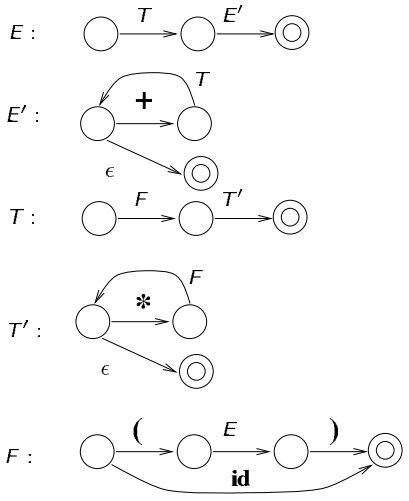


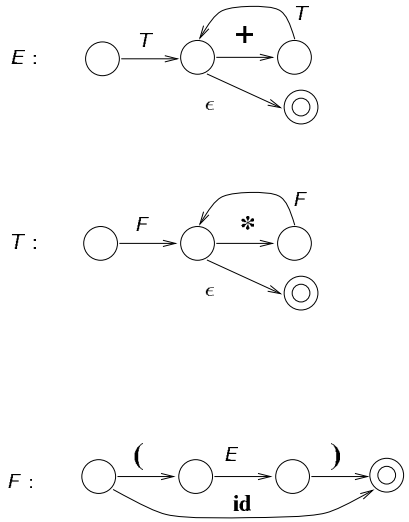

```
PROCEDURE P();
   IF curr_tok = a THEN curr_tok := next_token()
   ELSE syntax_error; ENDIF
   IF curr_tok ∈ FIRST(s1) THEN code for parsing s1
   ELSIF curr_tok ∈ FIRST(s2) THEN code for parsing s2
   ELSE syntax_error; ENDIF
   B();
   WHILE curr_tok ∈ FIRST(r) DO code for parsing r ENDDO
END P;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

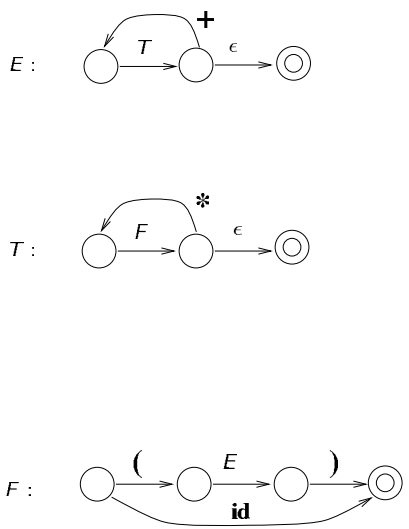
| | | |
|---|---|---|
| $E \rightarrow T\ E'$ | $T \rightarrow F\ T'$ | $F \rightarrow (\ E\ ) \mid \mathrm{id}$ |
| $E' \rightarrow +\ T\ E' \mid \epsilon$ | $T' \rightarrow *\ F\ T' \mid \epsilon$ | |

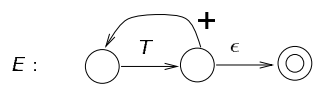


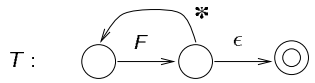


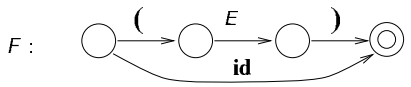




```
PROCEDURE E();
   LOOP
      T();
      IF cur_tok = + THEN
         cur_tok := next_token()
      ELSE EXIT ENDIF
   ENDLOOP;
```




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




```
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