



编译原理

第四章 语法分析——自上而下分析

第四章 语法分析—自上而下分析

- 语法分析器的功能
- 自上而下分析面临的问题
- LL(1) 分析法
- 递归下降分析程序构造
- 预测分析程序

第四章 语法分析—自上而下分析

- 语法分析器的功能
- 自上而下分析面临的问题
- LL(1) 分析法
 - 消除文法的左递归
 - 克服回溯
- 递归下降分析程序构造
- 预测分析程序

4.4 递归下降分析程序构造

- 构造不带回溯的自上而下分析程序
 - 要消除文法的左递归性
 - 克服回溯

构造不带回溯的自上而下分析器

■ 主要思路

- 分析程序由一组递归过程组成，对每一语法变量（非终结符）构造一个相应的子程序，识别对应的语法单位
- 通过子程序间的相互调用实现对输入串的认可
- 这样的分析程序称为递归下降分析器（因为文法的定义通常是递归的）

■ 几个全局过程和变量

- ADVANCE，把输入串指示器 IP 指向下一个输入符号，即读入一个单字符
- SYM，IP 当前所指的输入符号
- ERROR，出错处理子程序

■ 例：文法 $G(E)$:

$E \rightarrow TE'$

$E' \rightarrow +TE' \mid \varepsilon$

$T \rightarrow FT'$

$T' \rightarrow *FT' \mid \varepsilon$

$F \rightarrow (E) \mid i$

- 每个非终结符有对应的子程序的定义，首先在分析过程中，当需要从某个非终结符出发进行展开（推导）时，就调用这个非终结符对应的子程序。

递归下降子程序

$A \rightarrow TE' \mid BC \mid \varepsilon$

对应的递归下降子程序为

PROCEDURE A ;

BEGIN

IF SYM \in FIRST(TE') THEN

BEGIN T ; E' END

ELSE IF SYM \in FIRST(BC) THEN

BEGIN B; C END

ELSE IF SYM \notin FOLLOW(A) THEN

ERROR

END ;

■ 例：文法 $G(E)$:

$E \rightarrow TE'$

$E' \rightarrow +TE' \mid \varepsilon$

$T \rightarrow FT'$

$T' \rightarrow *FT' \mid \varepsilon$

$F \rightarrow (E) \mid i$

■ 对应的递归下

```
PROCEDURE F ;  
IF SYM= 'i' THEN ADVANCE  
ELSE  
  IF SYM= '(' THEN  
    BEGIN  
      ADVANCE;  
      E ;  
      IF SYM= ')' THEN ADVANCE  
      ELSE ERROR  
    END  
  ELSE ERROR;
```


■ 例：文法 $G(E)$:

$E \rightarrow TE'$

$E' \rightarrow +TE' \mid \varepsilon$

$T \rightarrow FT'$

$T' \rightarrow *FT' \mid \varepsilon$

$F \rightarrow (E) \mid i$

■ 对应的递归下降子程序为

```
PROCEDURE E' ;
IF SYM='+' THEN
BEGIN
    ADVANCE ;
    T ; E'
END
ELSE IF SYM<>'#' AND SYM<>')'
    THEN ERROR
```

■ E' 实现了 ε 吗 ?

A 实现了

B. 没实现

$FOLLOW(E') = \{), \# \}$

```
PROCEDURE E' ;
IF SYM= '+' THEN
BEGIN
    ADVANCE ;
    T ; E'
END
```

■ 例：文法 $G(E)$:

$E \rightarrow TE'$

$E' \rightarrow +TE' \mid \varepsilon$

$T \rightarrow FT'$

$T' \rightarrow *FT' \mid \varepsilon$

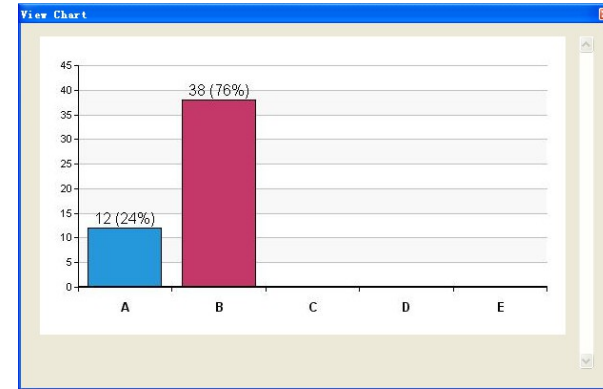
$F \rightarrow (E) \mid i$

■ E' 不考虑 Follow 集合有问题吗？

A 有问题

B. 没有问题

■ 对应的递归下降子程序为



```

PROCEDURE E' ;
IF SYM='+' THEN
BEGIN
    ADVANCE ;
    T ; E'
END
ELSE IF SYM<>'#' AND SYM<>')'
    THEN ERROR
    
```

```

PROCEDURE E' ;
IF SYM= '+' THEN
BEGIN
    ADVANCE ;
    T ; E'
END
    
```

- 例：文法 $G(E)$:

$E \rightarrow TE'$

$E' \rightarrow +TE' \mid \varepsilon$

$T \rightarrow FT'$

$T' \rightarrow *FT' \mid \varepsilon$

$F \rightarrow (E) \mid i$

- 对应的递归下降

```

PROCEDURE T ;
BEGIN
    F ; T'
END

```

```

PROCEDURE T' ;

```

```

IF SYM='*' THEN
BEGIN

```

```

    ADVANCE ;
    F ; T'

```

```

END

```

```


ELSE IF SYM<>'#' AND
      SYM<>')' AND SYM<>'+
THEN ERROR

```

```

PROCEDURE T' ;
IF SYM= '*' THEN
BEGIN
    ADVANCE ;
    F ; T'
END ;

```



```
主程序：  
PROGRAM PARSER;  
BEGIN  
    ADVANCE;  
    E;  
    IF SYM <> ' #' THEN  
        ERROR  
    END;  
END;
```

文法的另一种表示法和转换图

- 在元符号 “ \rightarrow ” 和 “ $|$ ” 的基础上，扩充几个元语言符号：



John W. Backus

For profound, influential, and lasting contributions to the design of practical high-level programming systems, notably through his work on FORTRAN, and for seminal publication of formal procedures for the specification of programming languages.

- 例如，通常的“实数”可定义为：

$\text{decimal} \rightarrow [\text{sign}] \text{integer} . \{ \text{digit} \}$
 $[\text{exponent}]$

$\text{exponent} \rightarrow E [\text{sign}] \text{integer}$

$\text{integer} \rightarrow \text{digit} \{ \text{digit} \}$

$\text{sign} \rightarrow + \mid -$

- 用扩充的巴科斯范式来描述语法，直观易懂，便于表示左递归消去和因子提取。

■ 例 4.5 文法

$$E \rightarrow T \mid E+T$$
$$T \rightarrow F \mid T * F$$
$$F \rightarrow i \mid (E)$$

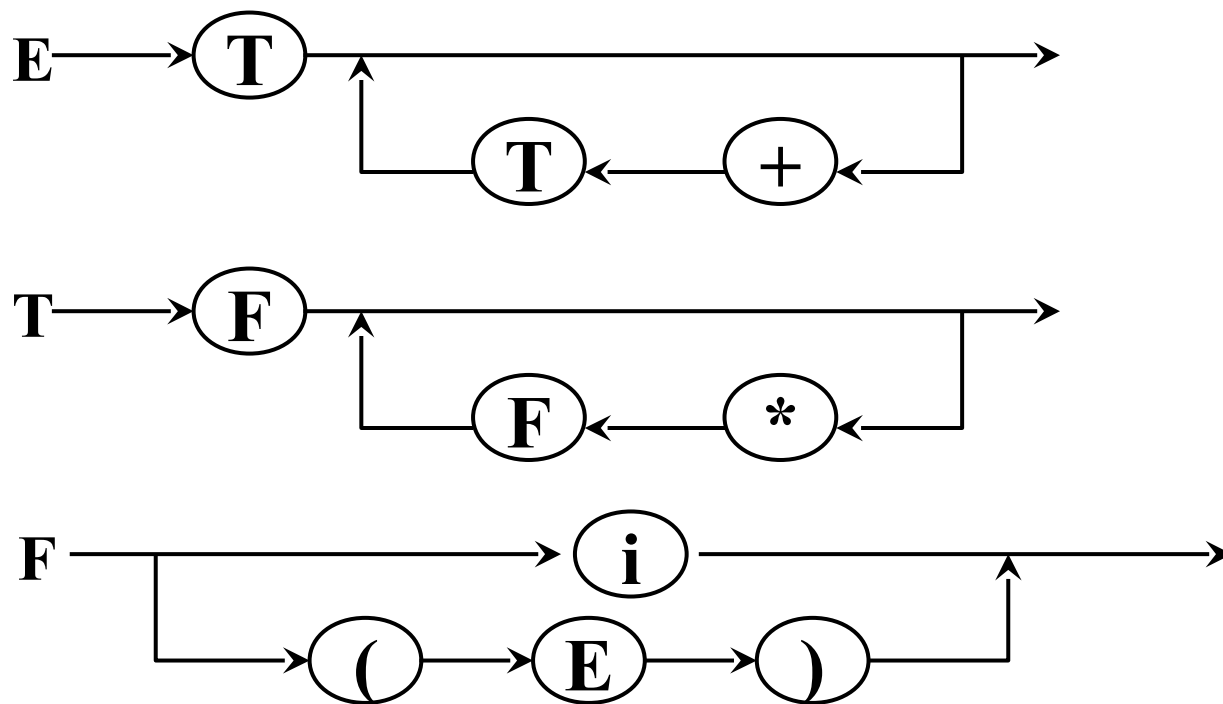
可表示成

$$E \rightarrow T\{+T\}$$
$$T \rightarrow F\{ * F\}$$
$$F \rightarrow i \mid (E)$$

(4.6)

- $E \rightarrow T\{+T\}$
 $T \rightarrow F\{*F\}$
 $F \rightarrow i \mid (E)$ (4.6)

- 可以用语法图来表示语言的文法



- $E \rightarrow T\{+T\}$
 $T \rightarrow F\{*F\}$
 $F \rightarrow i \mid (E)$ (4.6)
- 可构造一组递归下降分析程序

```
PROCEDURE E ;  
BEGIN  
  T ;  
  WHILE SYM= '+' DO  
  BEGIN  
    ADVANCE ;  
    T  
  END  
END ;
```

■ $E \rightarrow T\{+T\}$

$T \rightarrow F\{*F\}$

$F \rightarrow i \mid (E)$ (4.6)

■ 可构造一组递归下降分析程序

```
PROCEDURE T ;  
BEGIN  
    F ;  
    WHILE SYM= '*' DO  
    BEGIN  
        ADVANCE ;  
        F  
    END  
END ;
```

- $E \rightarrow T\{+T\}$
 $T \rightarrow F\{*F\}$
 $F \rightarrow i \mid (E)$

(4.6)

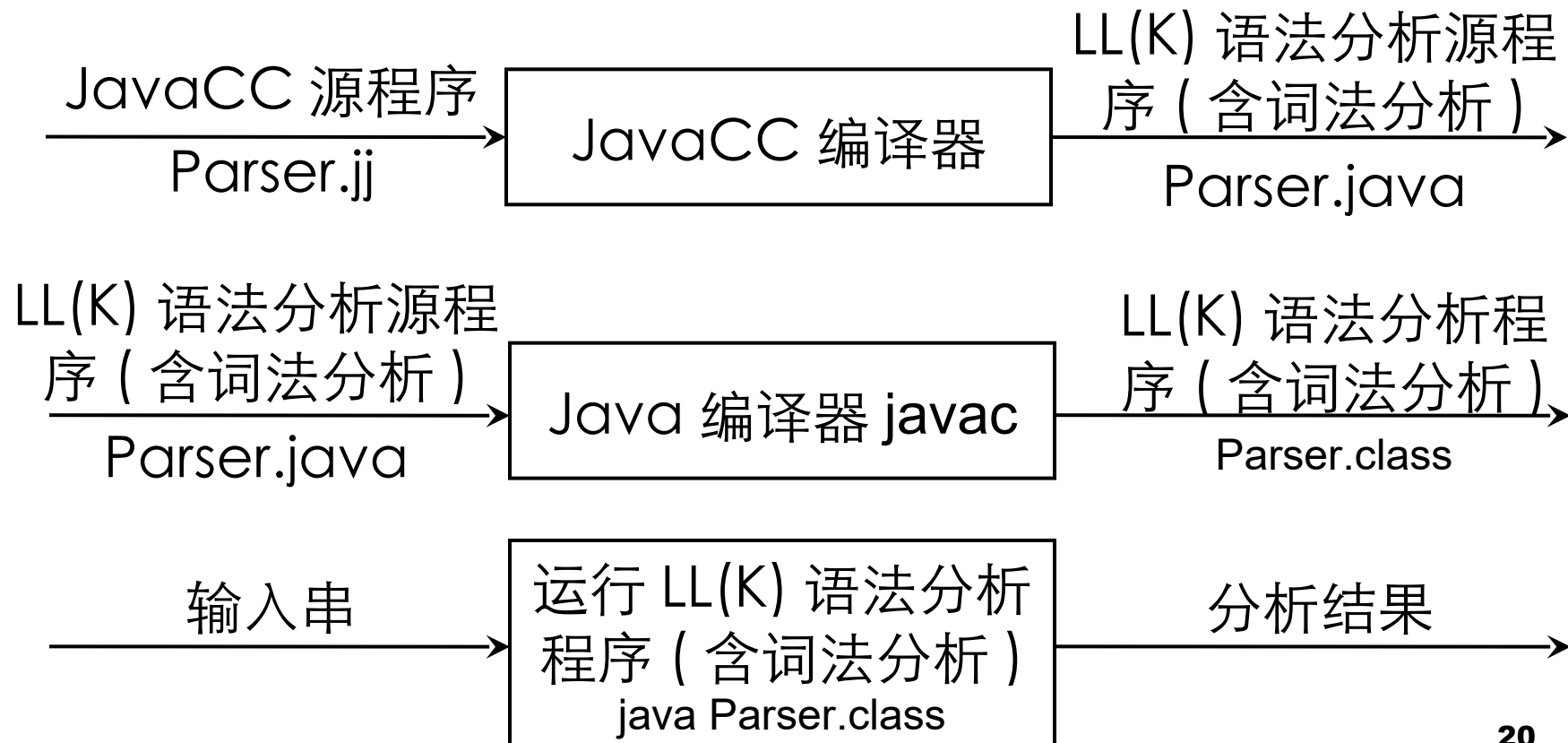
- 可构造一组递归下降分析程序

```
PROCEDURE F ;  
  IF SYM= 'i' THEN ADVANCE  
  ELSE  
    IF SYM= '(' THEN  
      BEGIN  
        ADVANCE;  
        E ;  
        IF SYM= ')' THEN ADVANCE  
        ELSE ERROR  
      END  
    ELSE ERROR;
```

JavaCC

- Java Compiler Compiler (JavaCC) - The Java Parser Generator

□ <http://javacc.java.net/>



JavaCC

- `<parser_name>.java`
- `<parser_name>Constants.java`
- `<parser_name>TokenManager.java`
- `ParseException.java`
- `SimpleCharStream.java`
- `Token.java`
- `TokenMgrError.java`



JavaCC

- JJTree
 - 构建分析树的附加工具

实验作业

■ JavaCC 实验

- 阅读：Howard Katz , JavaCC, parse trees, and the XQuery grammar ([Part 1](#) / [Part2](#))
 - 运行其中的例子 SimpleLang
- 用 JavaCC 构建一个 C 语言的语法分析程序
 - 阅读 C.JJ(在 repository of JavaCC grammars 中)
 - 用 JavaCC 编译 C.JJ , 产生一个 C 的语法分析程序
 - 运行该语法分析程序, 输入一个 C 程序, 观察输出结果

■ PL 语言编译器实验

- 阅读实验指南的 “第三节 PL 语法分析”
- 分析语法分析程序的结构和调用关系

小结

- LL(1) 分析法——递归下降分析程序
 - 消除左递归，消除回溯，转换成 LL(1) 文法
 - 分析程序由一组递归过程组成，对每一语法变量（非终结符）构造一个相应的子程序，识别对应的语法单位
 - 通过子程序间的相互调用实现对输入串的认可
- JavaCC
 - 递归下降分析程序自动生成