**第四章进度检查**

1. LL(1)中的第一个L是从左到右扫描输入串，第二个L是最左推导，括弧中的1是每次向前搜索一个输入字符。
2. 自上而下分析的前提是消除左递归和消除回溯。
3. 编译器常用的语法分析方法有两种，一种是自上而下，另一种是自下而上。
4. 一个LL（1）分析程序需要用到预测分析表和符号栈。
5. 判断：一个LL(1)文法，必然是无二义的。（对）
6. 判断：语法分析时，必须先消除文法中的左递归。（错）

LL（1）分析法时才必须先消除左递归。

1. 高级语言编译程序常用的语法分析方法中，递归下降分析法属于（B）分析方法。

A． 自左而右

B． 自上而下

C． 自下而上

D． 自右而左

1. 编译过程中语法分析器的任务是（B）
2. 分析单词是怎样构成的
3. 分析单词串是如何构成语句和说明的
4. 分析语句和说明是如何构成程序的
5. 分析程序的结构
6. (2) (3)
7. (2) (3)(4)
8. (1) (2)(3)
9. (1)(2) (3)(4)
10. 给定文法G[S]，

S→[L] |a

L →L,S |S

其中，G[S]包含的终结符有（D）， FIRST[S] = (B)

1. a
2. a [
3. a [ 和 ]
4. a[ ] 和，
5. 语法分析器可以发现源程序中的（D）
6. 语义错误
7. 语法和语义错误
8. 错误并校正
9. 语法错误
10. 若文法G定义的语言是无限集，则文法必然是（ A ）
11. 递归的
12. 前后文无关的
13. 二义性的
14. 无二义性的
15. 文法G[S]:

S → SS+ | SS\* | a

（a）请提取左公因子。

（b）提取左公因子后的文法是否有左递归，如果有左递归，请消除左递归。

（c）经过上述两步以后的文法是否适用自上而下分析法？为什么？

（d）计算经过上述两步以后的文法的FIRST和FOLLOW集合。

答：

（a） S→SSA | a

A→+ | \*

（b） S→aS’

S’→SAS’ | ε

A→+ | \*

（c）适用，因为不包含左递归，也不包含左公因子。

（d）FIRST(S) = {a} FOLLOW(S) = {+, \*, #}

FIRST(S’) = {a，ε} FOLLOW(S’) = {+, \*, #}

FIRST(A) = {+，\*} FOLLOW(A) = {a, +, \* #}

1. 文法G[S]:S→S(S)S | ε
2. 请消除左递归；
3. 请计算消除左递归后的文法中每个非终结符的FIRST和FOLLOW集合；
4. 请构造预测分析表，并判断该文法是否LL(1)文法，为什么？

答：

（a）S→εS’

S’→(S)SS’ | ε

化简为：S→S’

S’→(S)SS’ | ε

（b）FIRST(S) = {(, ε} FOLLOW(S) = {(, ), #}

FIRST(S’) = {(，ε} FOLLOW(S’) = {(, ), #}

（c）预测分析表为：

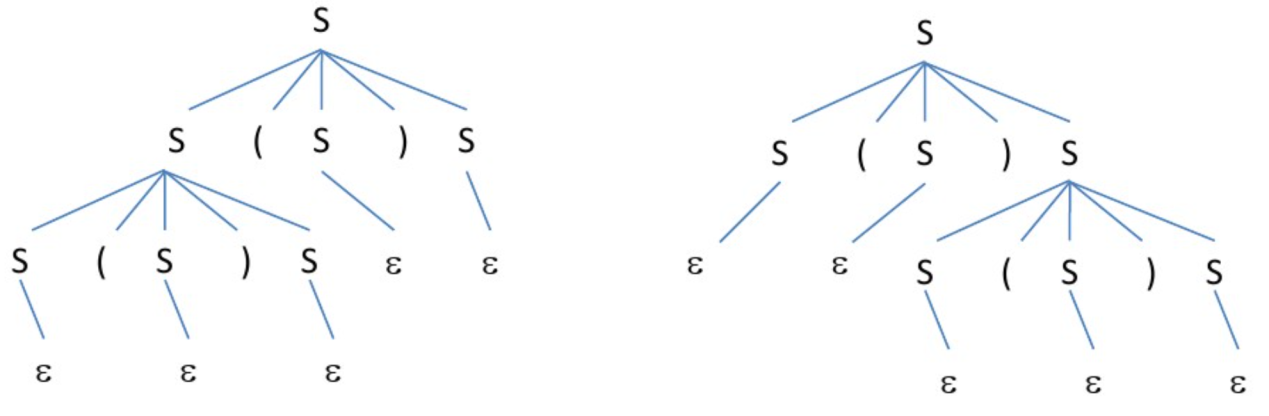
|  |  |  |  |
| --- | --- | --- | --- |
|  | ( | ) | # |
| S | S→S’ | S→S’ | S→S’ |
| S’ | S’→(S)SS’  S’→ε | S’→ε | S’→ε |

对产生式S→S’，FIRST(S’) = {(, ε}，因此把S→S’加入M[S,(]；由于FIRST(S’)有ε，同时看FOLLOW(S) = {(, ), #}，并把S→S’加入M[S,(]、M[S,)]和M[S,#]。

对产生式S’→(S)SS’，FIRST((S)SS’) = {(}，因此把S’→(S)SS’加入M[S’,(]。

对产生式S’→ε，看FOLLOW(S’) = {(, ), #}，把S’→ε加入M[S’,(]、M[S’,)]和M[S’,#]。

该分析表中M[S’,(]有冲突发生，因此不是LL(1)文法。该文法具有二义性，例如()()



1. 已知文法G[S]

S → aBc | bAB

A → aAb | b

B→b |ε

1. 构造文法G[S]的LL(1)预测分析表；
2. 判断符号串baabbb是否为该文法的句子（写出含有符号栈、输入串和规则的分析过程）。

答：（a）

FIRST(S) = {a, b} FOLLOW(S) = {#}

FIRST(A) = {a, b} FOLLOW(A) = {b, #}

FIRST(B) = {b, ε} FOLLOW(B) = {c, #}

FIRST(aBc) = {a}, 把S→aBc放入M[S,a]；

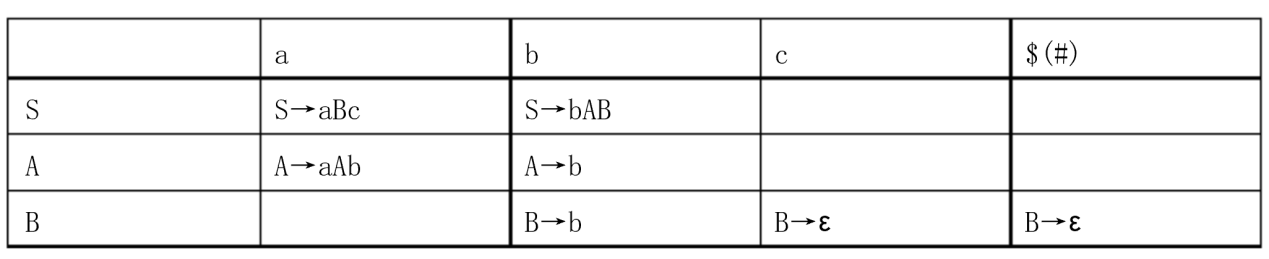
FIRST(bAB) = {b}, 把S→bAB放入M[S,b]；

FIRST(aAb) = {a}, 把A→aAb放入M[A,a]；

FIRST(b) = {b}, 把A→b放入M[A,b]；

FIRST(b) = {b}, 把B→b放入M[B,b]；

B→ε,FOLLOW(B) ={c, #}中的每个终结符（如果FOLLOW(B)中有#，则也包括#），添加B →ε到M[B, c] 和M[B, #]



（b）

|  |  |  |  |
| --- | --- | --- | --- |
| Matched | Stack | Input | Action |
|  | S# | baabbb# |  |
|  | bAB# | baabbb# | Output S→bAB |
| b | AB# | aabbb# | Match b |
| b | aAbB# | aabbb# | Output A→aAb |
| ba | AbB# | abbb# | Match a |
| ba | aAbbB# | abbb# | Output A→aAb |
| baa | AbbB# | bbb# | Match a |
| baa | bbbB# | bbb# | Output A→b |
| baab | bbB# | bb# | Match b |
| baabb | bB# | b# | Match b |
| baabbb | B# | # | Match b |
| baabbb | # | # | Output B→ε |

1. 有文法G[E]，

E→Aa | Bb

A→cA | eB

B→bd

(a)请判断该文法是否LL(1)文法，为什么；

(b)请构造该文法的递归子程序。

答：

（a）

(1)文法不含左递归，满足条件；

(2)文法中每个非终结符的各个产生式的候选首符集两两不相交，满足条件；

E→Aa | Bb和A→cA | eB的产生式有部有多个候选集

FIRST(Aa) ∩FIRST(Bb) ={c,e} ∩{b} = Ø

FIRST(cA) ∩FIRST(eB) ={c} ∩{e} = Ø

(3)文法中的每个非终结符，候选首符集都不包含ε，满足条件。

该文法满足以上三个条件，所以该文法是LL(1)文法。

（b）

**ADVANCE：**读入IP 指向的输入符号到SYM中，把输入串指示器IP指向下一个输入符号

**SYM ：**IP当前所指的输入符号

E→Aa | Bb

A→cA | eB

B→bd

**ERROR：**出错处理程序

构造递归子程序为：

PROCEDURE E

BEGIN

IF **SYM** in FIRST(Aa) THEN (判断是否应该分析A)

BEGIN

A; (调用PROCEDURE A，把SYM中的符号交给A)

**ADVANCE**； (读入符号，即当前IP指向的符号，同时IP向前移动一位，指向下一个符号)

IF **SYM**==‘a’ THEN (如果读入的符号为a，即是否符合产生式右部Aa)

**ADVANCE**;

ELSE **ERROR**; （如果读入的符号不是a，则调用出错处理程序）

END

ELSE IF **SYM** in FIRST(Bb) THEN (判断是否应该分析B)

BEGIN

B; (调用PROCEDURE B，，把SYM中的符号交给B)

**ADVANCE**；

IF **SYM**==‘b’ THEN (如果读入的符号为b，即是否符合产生式右部Bb)

**ADVANCE**;

ELSE **ERROR**;

END

END

PROCEDURE A

BEGIN

IF SYM=‘c’ THEN (判断输入是否为c)

BEGIN

ADVANCE；

A;

END

ELSE IF SYM=‘e’ THEN (判断读入的符号是否为e)

BEGIN

ADVANCE；

B;

END

ELSE ERROR;

END

PROCEDURE B

BEGIN

IF SYM=‘b’ THEN (判断读入的符号是否为b)

BEGIN

ADVANCE；

IF SYM=‘d’ THEN (判断读入的符号是否为e)

BEGIN

ADVANCE；

END

ELSE ERROR;

END

ELSE ERROR;

END

主程序：

ADVANCE；

E; (调用PROCEDURE E，把SYM中的符号交给E)

IF SYM=‘#’ THEN (判断读入的符号是否为最后的#)

BEGIN

WRITE(“RIGHT”);

END

ELSE ERROR;