

(若发现问题, 请及时告知)

A1 给定文法  $G[S]$ :

$S \rightarrow AB$   
 $A \rightarrow aA$   
 $A \rightarrow \varepsilon$   
 $B \rightarrow bB$   
 $B \rightarrow \varepsilon$

针对文法  $G[S]$ , 下表给出各产生式右部文法符号串( $rhs(r)$ )的  $First$  集合, 各产生式左部非终结符( $lhs(r)$ )的  $Follow$  集合, 以及各产生式的预测集合  $PS$ 。试填充其中空白表项的内容 (1)-(6), 验证该文法是否 LL(1)文法 (说明原因), 并补全基于该文法构造的递归下降分析程序 (7)-(10), 其中 `getToken()` 为取下一单词的过程, 变量 `lookahead` 存放当前单词。

$G$ 中的规则 $r$	$First(rhs(r))$	$Follow(lhs(r))$	$PS(r)$
$S \rightarrow AB$	(1)	#	(2)
$A \rightarrow aA$	$a$	(3)	$a$
$A \rightarrow \varepsilon$	$\varepsilon$	此处不填	(4)
$B \rightarrow bB$	(5)	#	(6)
$B \rightarrow \varepsilon$	$\varepsilon$	此处不填	#

```
void ParseS( )                // 主函数
{
    ParseA( );
    ParseB( );
}
void ParseA( )
{
    switch (lookahead)        // lookahead 为下一个输入符号
    {
        case ' a' :
            (7);
            ParseA();
            break;
        case (8):
            break;
        default:
            printf("syntax error \n")
            exit(0);
    }
}
void ParseB( )
```

```

{
    switch (lookahead) {
        case (9):
            MatchToken( 'b' );
            (10);
            break;
        case ' #' :
            break;
        default:
            printf("syntax error \n");
            exit(0);
    }
}

void Match_Token(int expected)
{
    if (lookahead != expected)
    {
        printf("syntax error \n")
        exit(0);
    }
    else
        lookahead = getToken();
}

```

### 参考解答：

$G$ 中的规则 $r$	$First(rhs(r))$	$Follow(lhs(r))$	$PS(r)$
$S \rightarrow AB$	$a, b, \varepsilon$	$\#$	$a, b, \#$
$A \rightarrow aA$	$a$	$b, \#$	$a$
$A \rightarrow \varepsilon$	$\varepsilon$	此处不填	$b, \#$
$B \rightarrow bB$	$b$	$\#$	$b$
$B \rightarrow \varepsilon$	$\varepsilon$	此处不填	$\#$

$$PS(A \rightarrow aA) \cap PS(A \rightarrow \varepsilon) = \{a\} \cap \{b, \#\} = \Phi$$

$$PS(B \rightarrow bB) \cap PS(B \rightarrow \varepsilon) = \{b\} \cap \{\#\} = \Phi$$

所以， $G(S)$  是 LL(1) 文法。

用类似 C 语言写出  $G[E]$  的递归子程序，其中 `yylex()` 为取下一单词过程，变量 `lookahead` 存放当前单词。不需要考虑太多编程语言相关的细节。程序如下：

```

void ParseS( )                // 主函数
{
    ParseA( );
    ParseB( );
}

```

```

}
void ParseA( )
{
    switch (lookahead)          // lookahead 为下一个输入符号
    {
        case ' a' :
            MatchToken( 'a' );
            ParseA();
            break;
        case ' b' , ' #' :
            break;
        default:
            printf("syntax error \n")
            exit(0);
    }
    return A_num;
}
void ParseB( )
{
    switch (lookahead) {
        case ' b' :
            MatchToken( 'b' );
            ParseB( );
            break;
        case ' #' :
            break;
        default:
            printf("syntax error \n");
            exit(0);
    }
}
void Match_Token(int expected)
{
    if (lookahead != expected)
    {
        printf("syntax error \n")
        exit(0);
    }
    else
        lookahead = getToken();
}

```

**A2** 补全下列文法（与上一题相同）的分析表，根据分析表验证该文法是否 LL(1)文法（说明原因），并补全输入符号串 *aabb* 的表驱动 LL(1)分析过程（步骤 3、6、9）。

$S \rightarrow AB$

$A \rightarrow aA$

$A \rightarrow \varepsilon$

$B \rightarrow bB$

$B \rightarrow \varepsilon$

	$a$	$b$	$\#$
$S$	$S \rightarrow AB$	<u>(1)</u>	$S \rightarrow AB$
$A$	<u>(2)</u>	<u>(3)</u>	$A \rightarrow \varepsilon$
$B$	此处不填	<u>(4)</u>	$B \rightarrow \varepsilon$

步骤	下推栈	余留符号串	下一步动作
1	#S	<u>aabb#</u>	应用产生式 $S \rightarrow AB$
2	#BA	<u>aabb#</u>	应用产生式 $A \rightarrow aA$
3			
4	#BA	<u>abb#</u>	应用产生式 $A \rightarrow aA$
5	#BAa	<u>abb#</u>	匹配栈顶和当前输入符号
6			
7	#B	<u>bb#</u>	应用产生式 $B \rightarrow bB$
8	#Bb	<u>bb#</u>	匹配栈顶和当前输入符号
9			
10	#Bb	<u>b#</u>	匹配栈顶和当前输入符号
11	#B	<u>#</u>	应用产生式 $B \rightarrow \varepsilon$
12	#	<u>#</u>	返回，分析成功

### 参考解答：

	$a$	$b$	$\#$
$S$	$S \rightarrow AB$	$S \rightarrow AB$	$S \rightarrow AB$
$A$	$A \rightarrow aA$	$A \rightarrow \varepsilon$	$A \rightarrow \varepsilon$
$B$		$B \rightarrow bB$	$B \rightarrow \varepsilon$

分析表中每个单元最多有一个产生式，所以是 LL(1)。

步骤	下推栈	余留符号串	下一步动作
1	#S	<u>aabb#</u>	应用产生式 $S \rightarrow AB$
2	#BA	<u>aabb#</u>	应用产生式 $A \rightarrow aA$
3	#BAa	<u>aabb#</u>	匹配栈顶和当前输入符号
4	#BA	<u>abb#</u>	应用产生式 $A \rightarrow aA$
5	#BAa	<u>abb#</u>	匹配栈顶和当前输入符号
6	#BA	<u>bb#</u>	应用产生式 $A \rightarrow \varepsilon$

7	#B	<u>bb</u> #	应用产生式 $B \rightarrow bB$
8	#Bb	<u>bb</u> #	匹配栈顶和当前输入符号
9	#B	<u>b</u> #	应用产生式 $B \rightarrow bB$
10	#Bb	<u>b</u> #	匹配栈顶和当前输入符号
11	#B	#	应用产生式 $B \rightarrow \epsilon$
12	#	#	返回, 分析成功

A3 给定命题表达式文法  $G[S]$ :

$$S \rightarrow P$$

$$P \rightarrow \wedge P P \mid \vee P P \mid \neg P \mid \underline{id}$$

其中,  $\wedge$ 、 $\vee$ 、 $\neg$  分别代表命题逻辑与、或、非等运算符单词,  $\underline{id}$  代表标识符单词。

容易得出:  $G[S]$  是  $LL(1)$  文法。基于  $G[S]$  的预测分析表和一个分析栈, 课程中介绍了一种表驱动的  $LL(1)$  分析过程。假设有输入符号串:  $\vee \vee a \wedge b c \vee \neg a \wedge c b$ 。试问, 在分析过程中, 分析栈中最多会出现几个  $S$ ? 几个  $P$ ? 若因误操作使输入串多了一个符号, 变为  $\vee \vee a \wedge b c c \vee \neg a \wedge c b$ , 当分析过程中发生错误时, 关于报错信息, 你认为最不可能的选择是 (4选1):  
 (1) 缺运算数; (2) 多运算数; (3) 缺运算符; (4) 多运算符。如果想要从该出错位置恢复分析, 可以进行什么操作?

参考解答:

在分析过程中, 分析栈中最多会出现1个 $S$ , 3个 $P$ 。

(1) 缺运算数。提示: 读入第二个  $c$  时出错, 可以直接报“多运算数”(多第二个 $c$ ), 或者“缺运算符”(若不遇到第二个 $c$ , 而是遇到一个运算符, 则此时不会出错); 另外, 若此时输入已结束, 则不会出错, 但下一个输入符号是运算符 ( $\vee$ ), 所以也可以报“多运算符”。

如果想要从该出错位置恢复分析, 可以删掉当前输入符号 ( $c$ ), 并将出错时使用的产生式左边的非终结符 ( $P$ ) 退回到分析栈顶, 然后就可以恢复分析了。可能还有其他恢复手段, 只要合理都是可以的。

.....  
 以下是 Lecture03 文档中的题目  
 .....

A4 设有文法  $G[S]$ :

$$S \rightarrow a S b \mid a a b$$

若针对该文法设计一个自顶向下预测分析过程，则需要向前察看多少个输入符号？

参考解答：

需要向前察看 3 个单词。若向前察看 3 个单词是  $aab$  时，可选第 2 个分支； $aaa$  时，可选第 1 个分支。

A5 试验证下列文法  $G[S]$  是 LL(1) 文法:

$$S \rightarrow P \mid \varepsilon$$

$$P \rightarrow (P)P \mid a$$

其中  $(, )$ ，以及  $a$  为终结符

参考解答：

先对于每条语法规则  $r$ ，计算  $PS(r)$ 。

$G$ 中的规则 $r$	$First(rhs(r))$	$Follow(lhs(r))$	$PS(r)$
$S \rightarrow P$	$a, ($	$\#$	$a, ($
$S \rightarrow \varepsilon$	$\varepsilon$		$\#$
$P \rightarrow (P)P$	$($	$), \#$	$($
$P \rightarrow a$	$a$		$a$

因为  $PS(S \rightarrow P) \cap PS(S \rightarrow \varepsilon) = \emptyset$  及  $PS(P \rightarrow (P)P) \cap PS(P \rightarrow a) = \emptyset$ ，所以文法  $G[S]$  是 LL(1) 文法。

A6 给出如下文法的预测分析表，并根据分析表指出相应文法是否 LL(1) 文法。

$$S \rightarrow aSa \mid bSb \mid \varepsilon$$

参考解答：

$$PS(S \rightarrow aSa) = \{ a \}$$

$$PS(S \rightarrow bSb) = \{ b \}$$

$$PS(S \rightarrow \varepsilon) = \{ \#, a, b \}$$

	a	b	#
S	$S \rightarrow aSa$ $S \rightarrow \varepsilon$	$S \rightarrow bSb$ $S \rightarrow \varepsilon$	$S \rightarrow \varepsilon$

$M[S, a] = \{ S \rightarrow aSa, S \rightarrow \varepsilon \}$ ,  $M[S, b] = \{ S \rightarrow bSb, S \rightarrow \varepsilon \}$ , 所以, 不是 LL(1)文法。

**A7** 按照本讲介绍的消除一般左递归算法消除下面文法  $G[S]$  中的左递归 (要求依非终结符的排序  $S$ 、 $Q$ 、 $P$  执行该算法) :

$$\begin{aligned} S &\rightarrow PQ \mid a \\ P &\rightarrow QS \mid b \\ Q &\rightarrow SP \mid c \end{aligned}$$

### 参考解答:

按照非终结符的特定顺序排列各规则:

$$\begin{aligned} S &\rightarrow PQ \mid a \\ Q &\rightarrow SP \mid c \\ P &\rightarrow QS \mid b \end{aligned}$$

第一步, 得:

$$\begin{aligned} S &\rightarrow PQ \mid a \\ Q &\rightarrow PQP \mid aP \mid c \\ P &\rightarrow QS \mid b \end{aligned}$$

第二步, 得:

$$\begin{aligned} S &\rightarrow PQ \mid a \\ Q &\rightarrow PQP \mid aP \mid c \\ P &\rightarrow PQPS \mid aPS \mid cS \mid b \end{aligned}$$

消去  $P \rightarrow PQPS$  的左递归得:

$$\begin{aligned} S &\rightarrow PQ \mid a \\ Q &\rightarrow PQP \mid aP \mid c \\ P &\rightarrow aPSR \mid cSR \mid bR \\ R &\rightarrow QPSR \mid \varepsilon \end{aligned}$$

经检查, 此时得到的文法已经不含左递归, 可结束消除左递归过程。