

(若发现问题, 请及时告知)

A1 根据你的判断, 针对以下文法是否可以设计一个自顶向下预测分析过程? 如果可以, 需要向前察看多少个输入符号?

(1) 文法 $G_1[S]$:

$$\begin{aligned} S &\rightarrow A \mid B \\ A &\rightarrow aAb \mid c \\ B &\rightarrow aBbb \mid d \end{aligned}$$

(2) 文法 $G_2[S]$:

$$\begin{aligned} S &\rightarrow \varepsilon \mid abA \\ A &\rightarrow Saa \mid b \end{aligned}$$

参考解答:

(1) 不可以。

(2) 可以, 需要向前察看 2 个输入符号。

A2 给定某类表达式文法 $G[E]$:

$$\begin{aligned} E &\rightarrow +ER \mid -ER \mid \underline{positive} R \\ R &\rightarrow *ER \mid \varepsilon \end{aligned}$$

其中, $+$ 和 $-$ 分别代表一元正和一元负运算, $*$ 代表普通的二元乘法运算, positive 为代表正整数 (非0) 的单词。

(1) 针对文法 $G[E]$, 下表给出各产生式右部文法符号串的 *First* 集合, 各产生式左部非终结符的 *Follow* 集合, 以及各产生式的预测集合 *PS*。试填充其中空白表项 (共3处) 的内容:

$G[E]$ 的规则 r	$First(rhs(r))$	$Follow(lhs(r))$	$PS(r)$
$E \rightarrow +ER$	$+$		$+$
$E \rightarrow -ER$	$-$	此处不填	$-$
$E \rightarrow \underline{positive} R$	<u>positive</u>	此处不填	<u>positive</u>
$R \rightarrow *ER$	$*$		$*$
$R \rightarrow \varepsilon$	ε	此处不填	

表中, $rhs(r)$ 为产生式 r 右部的文法符号串, $lhs(r)$ 为产生式 r 左部的非终结符。

- (2) $G[E]$ 不是 LL(1) 文法，试解释为什么？
- (3) 虽然 $G[E]$ 不是 LL(1) 文法，但可以采用一种强制措施，使得常规的 LL(1) 分析算法仍然可用。针对含5个单词的输入串 $+ - 20 * 18$ ，以下基于这一措施以及上述各产生式的预测集合（或预测分析表）的一个表驱动 LL(1) 分析过程：

步骤	下推栈	余留符号串	下一步动作
1	# E	$+ - 20 * \underline{18} \#$	应用产生式 $E \rightarrow + E R$
2	# $R E +$	$+ - \underline{20} * \underline{18} \#$	匹配栈顶和当前输入符号
3	# $R E$	$- \underline{20} * \underline{18} \#$	应用产生式 $E \rightarrow - E R$
4	# $R R E -$	$- \underline{20} * \underline{18} \#$	匹配栈顶和当前输入符号
5	# $R R E$	$\underline{20} * \underline{18} \#$	应用产生式 $E \rightarrow \underline{positive} R$
6	# $R R R \underline{positive}$	$\underline{20} * \underline{18} \#$	匹配栈顶和当前输入符号
7	# $R R R$	$* \underline{18} \#$	
8			
9			
10			
11			
12	# $R R R$	#	应用产生式 $R \rightarrow \epsilon$
13	# $R R$	#	应用产生式 $R \rightarrow \epsilon$
14	# R	#	应用产生式 $R \rightarrow \epsilon$
15	#	#	结束

试填写上述分析过程中第7步时使用的产生式，以及第 8~11 步的分析过程，共计13处空白；并指出采用了什么样的强制措施。

参考解答：

(1)

$G[E]$ 的规则 r	$First(rhs(r))$	$Follow(lhs(r))$	$PS(r)$
$E \rightarrow + E R$	+	# *	+
$E \rightarrow - E R$	-	# *	-
$E \rightarrow \underline{positive} R$	<u>positive</u>	# *	<u>positive</u>
$R \rightarrow * E R$	*	# *	*
$R \rightarrow \epsilon$	ϵ	# *	# *

表中的 $rhs(r)$ 表示产生式 r 右部的文法符号串, $lhs(r)$ 表示产生式 r 左部的非终结符。

(2)

因为 $PS(R \rightarrow *ER)$ 与 $PS(R \rightarrow \epsilon)$ 相交不为空。

(3)

步骤	下推栈	余留符号串	下一步动作
1	# E	+ - 20 * <u>18</u> #	应用产生式 $E \rightarrow +ER$
2	# RE +	+ - <u>20</u> * <u>18</u> #	匹配栈顶和当前输入符号
3	# RE	- <u>20</u> * <u>18</u> #	应用产生式 $E \rightarrow -ER$
4	# RE -	- <u>20</u> * <u>18</u> #	匹配栈顶和当前输入符号
5	# RE	<u>20</u> * <u>18</u> #	应用产生式 $E \rightarrow \underline{positive} R$
6	# $RR \underline{positive}$	<u>20</u> * <u>18</u> #	匹配栈顶和当前输入符号
7	# RRR	* <u>18</u> #	应用产生式 $R \rightarrow *ER$
8	# $RRRE$ *	* <u>18</u> #	匹配栈顶和当前输入符号
9	# $RRRE$	<u>18</u> #	应用产生式 $E \rightarrow \underline{positive} R$
10	# $RRRR \underline{positive}$	<u>18</u> #	匹配栈顶和当前输入符号
11	# $RRRR$	#	应用产生式 $R \rightarrow \epsilon$
12	# RRR	#	应用产生式 $R \rightarrow \epsilon$
13	# RR	#	应用产生式 $R \rightarrow \epsilon$
14	# R	#	应用产生式 $R \rightarrow \epsilon$
15	#	#	结束

采用的强制措施：如果在下一输入符号为 *（未到结束符 #）时，不使用 $E \rightarrow \epsilon$ ，而是使用 $R \rightarrow *ER$ 。

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。

A4 分析表达式文法 (Parsing Expression Grammar, PEG) 是一个四元组 (N, Σ, P, S) , 其中

- N 为非终结符 (nonterminal) 集合;
- Σ 为终结符 (terminal) 集合;
- P 为产生式 (parsing rules) 集合;
- S 为开始符号 (start symbol)。

各产生式形如 $A \rightarrow e$, 其中 A 为非终结符, e 为分析表达式 (parsing expression)。每个分析表达式都会用来匹配输入串的一个前缀。

本题考虑一种简单的 PEG, 其分析表达式定义如下:

$$e ::= t \mid n \mid e_1 e_2 \mid e_1 > e_2 \mid e^* \mid (e)$$

其中 $t \in \Sigma$ 只能匹配终结符 t 自身, $n \in N$ 只能匹配非终结符 n 自身。其他复合的表达式及分析过程为 (分析失败则表明输入串不能被该文法识别):

- 序列 “ $e_1 e_2$ ” 表示: 先用输入串匹配 e_1 , 若成功再用剩下的串匹配 e_2 ;
- 有序选择 “ $e_1 > e_2$ ” 表示: 先尝试用输入串匹配 e_1 , 若成功则分析结束 (忽略 e_2); 若失败, 重新用输入串匹配 e_2 ;
- 星闭包 “ e^* ” 表示: 用输入串匹配 e 零次或多次, 直至匹配失败 (即消耗掉输入串中尽可能多的符号);
- 特别地, 括号用于显式指定优先级。

可以发现, PEG 由于其分析过程允许回溯, 因而无需像传统的 LL 分析那样要向后查看符号。

根据以上设定, 回答下列问题。

- (1) 什么样的输入串可以匹配 (指完全匹配, 即匹配完以后没有余留的串) 分析表达式 “ a^* ”

ab ”，其中 a 和 b 为终结符？

(2) 类似于 “ e^* ”，我们可以扩展出 “ e^+ ”——它表示用输入串匹配 e 一次或多次，直至匹配失败。请利用以上给出的这些分析表达式，写出 “ e^+ ” 的定义。

(3) 利用 PEG 可以消除悬挂 `else` 的二义性。请简要说明如何消除？

(4) 考查下列 CFG（其中 x 为终结符）：

$$S \rightarrow x S x \mid x$$

它对应的语言记作 L 。问：

a. 是否可以直接对上述文法采用递归下降方法来分析？请说明理由。

b. 是否存在某个能做递归下降分析的 PEG，其对应的语言也为 L ？若存在，请给出此 PEG；否则，请说明理由。

参考解答：

(1) 没有任何输入串可以匹配 “ $a^* a b$ ”，因为 “ a^* ” 将消耗掉该输入串任何连续的 a 构成的前缀，导致下一个 “ a ” 无法被匹配。

(2) $e^+ = e e^*$

(3) 通过有序选择，大致文法如下：

$$stmt \rightarrow \dots > \underline{\text{if expr stmt}} \underline{\text{else stmt}} > \underline{\text{if expr stmt}}$$

(4)

a. 不能。语言 $L = \{x^n \underline{x} x^n\}$ 为以 \underline{x} 为中心的全 x 串，而在分析前我们无法找到中心点的 \underline{x} 位于何处，因此我们无法决定何时采用 $S \rightarrow x S x$ ，何时采用 $S \rightarrow x$ 。

b. 该语言 $L = \{x^{2n+1}\}$ ，即奇数个 x 组成的串。因此可以设计如下接受 L 的 PEG：

$$S \rightarrow x (x x)^*$$

.....

以下是 Lecture03 文档中的题目

.....

5 计算下列文法中每个非终结符的 First 集和 Follow 集，以及每个产生式的预测集合，并判断该文法是否 LL(1) 文法（说明原因）：

(2) 文法 $G_2[S]$ ：

$$\begin{aligned} S &\rightarrow TP \\ T &\rightarrow +PT \mid \varepsilon \\ P &\rightarrow (S) \mid a \end{aligned}$$

(3) 文法 $G_3[S]$ ：

$$\begin{aligned} S &\rightarrow A1B \\ A &\rightarrow 0A \mid \varepsilon \\ B &\rightarrow 0B \mid 1B \mid \varepsilon \end{aligned}$$

参考解答:

(2) 计算非终结符的 FIRST 集和 FOLLOW 集, 结果如下:

$$\text{FIRST}(S) = \{ +, (, a \}$$

$$\text{FOLLOW}(S) = \{ \#,) \}$$

$$\text{FIRST}(T) = \{ +, \varepsilon \}$$

$$\text{FOLLOW}(T) = \{ (, a \}$$

$$\text{FIRST}(P) = \{ (, a \}$$

$$\text{FOLLOW}(P) = \{ \#, +, (, a,) \}$$

$$\text{PS}(S \rightarrow TP) = \{ +, (, a \}$$

$$\text{PS}(T \rightarrow +PT) = \{ + \}$$

$$\text{PS}(T \rightarrow \varepsilon) = \{ (, a, + \}$$

$$\text{PS}(P \rightarrow (S)) = \{ (\}$$

$$\text{PS}(P \rightarrow a) = \{ a \}$$

因为, $\text{PS}(T \rightarrow +PT) \cap \text{PS}(T \rightarrow \varepsilon) = \{ + \} \cap \{ (, a \} = \Phi$, 所以, $G(S)$ 是 LL(1) 文法。

(3) 计算 FOLLOW 集, 以及每个产生式的预测集合, 结果如下:

G 中的规则 r	$\text{First}(rhs(r))$	$\text{Follow}(lhs(r))$	$\text{PS}(r)$
$S \rightarrow A1B$	0, 1	#	0, 1
$A \rightarrow 0A$	0	1	0
$A \rightarrow \varepsilon$	ε	此处不填	1
$B \rightarrow 0B$	0	#	0
$B \rightarrow 1B$	1	此处不填	1
$B \rightarrow \varepsilon$	ε	此处不填	#

表中的 $rhs(r)$ 表示产生式 r 右部的文法符号串, $lhs(r)$ 表示产生式 r 左部的非终结符。

非终结符的 FIRST 集可从上表中第2列对应符号的并集得到, 这里省略。

因为 $\text{PS}(A \rightarrow 0A) \cap \text{PS}(A \rightarrow \varepsilon) = \emptyset$ 及 $\text{PS}(B \rightarrow 0B)$ 、 $\text{PS}(B \rightarrow 1B)$ 和 $\text{PS}(B \rightarrow \varepsilon)$ 两两互不相交, 所以文法 $G[S]$ 是 LL(1) 文法。

6 验证如下文法是 LL(1) 文法, 并基于该文法构造递归下降分析程序:

(2) 文法 $G'[E]$:

$$E \rightarrow [F] E'$$

$$E' \rightarrow E \mid \varepsilon$$

$$F \rightarrow aF'$$

$$F' \rightarrow aF' \mid \varepsilon$$

参考解答:

(2) 观察文法规则可知, 可能产生规则选择冲突的规则只能是 $E' \rightarrow E \mid \varepsilon$ 和 $F' \rightarrow aF' \mid \varepsilon$ 。我们只要求出这四条规则的 PS 集合(预测集合)即可。欲求这四个 PS 集合, 我们需要先求出:

$$\begin{array}{ll} \text{First}(E) = \{ [\} & \text{Follow}(E') = \{ \# \} \\ \text{First}(aF') = \{ a \} & \text{Follow}(F') = \{] \} \end{array}$$

从而

$$\begin{array}{l} PS(E' \rightarrow E) = \text{First}(E) = \{ [\} \\ PS(E' \rightarrow \varepsilon) = \text{Follow}(E') = \{ \# \} \\ PS(F' \rightarrow aF') = \text{First}(aF') = \{ a \} \\ PS(F' \rightarrow \varepsilon) = \text{Follow}(F') = \{] \} \end{array}$$

因为

$$\begin{array}{l} \text{对于 } E' \rightarrow E \mid \varepsilon \text{ 有: } PS(E' \rightarrow E) \cap PS(E' \rightarrow \varepsilon) = \Phi \\ \text{对于 } F' \rightarrow aF' \mid \varepsilon \text{ 有: } PS(F' \rightarrow aF') \cap PS(F' \rightarrow \varepsilon) = \Phi \end{array}$$

所以, 文法 $G[E]$ 是 LL(1) 文法。

用类似 C 语言写出 $G[E]$ 的递归子程序, 其中 `getToken()` 为取下一单词过程, 变量 `lookahead` 为全局变量, 存放当前单词。

```
void ParseE( ) {
    Match_Token ( [ );
    ParseF( );
    MatchToken ( ] );
    ParseE' ( );
}

void ParseE' ( ) {
    switch (lookahead) {
        case [:
            ParseE( );
            break;
        case #:
            break;
        default:
            printf("syntax error \n" );
            exit(0);
    }
}

void ParseF( ) {
    MatchToken ( a );
    ParseF' ( );
}

void ParseF' ( ) {
    switch (lookahead) {
        case a:
            MatchToken ( a );
            ParseF' ( );
    }
}
```

```

        break;
    case ]:
        break;
    case:
        printf("syntax error \n" );
        exit(0);
}
}

void MatchToken(int expected) { //判别当前单词是否与期望的终结符匹配
    if (lookahead != expected) {
        printf("syntax error \n" );
        exit(0);
    } else // 若匹配, 消费掉当前单词并调用词法分析器读入下一个单词
        lookahead = getToken();
}

```

7 给出习题 5 中所有文法的预测分析表, 并根据分析表指出相应文法是否 LL(1)的, 同时验证习题 5 的结果。

参考解答:

(2) 文法 $G_2[S]$:

$$\begin{aligned}
 S &\rightarrow TP \\
 T &\rightarrow +PT \mid \varepsilon \\
 P &\rightarrow (S) \mid a \\
 PS(S \rightarrow TP) &= \{ +, (, a \} \\
 PS(T \rightarrow +PT) &= \{ + \} \\
 PS(T \rightarrow \varepsilon) &= \{ (, a \} \\
 PS(P \rightarrow (S)) &= \{ (\} \\
 PS(P \rightarrow a) &= \{ a \}
 \end{aligned}$$

	()	a	+	#
S	$S \rightarrow TP$		$S \rightarrow TP$	$S \rightarrow TP$	
T	$T \rightarrow \varepsilon$		$T \rightarrow \varepsilon$	$T \rightarrow +PT$	
P	$P \rightarrow (S)$		$P \rightarrow a$		

每一个表项唯一确定, 所以, 是 LL(1)文法。

(3) 文法 $G_3[S]$:

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

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

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

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

	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。

11 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。