《编译原理》第一次书面作业

截止日期：2024 年 10 月 22 日

若发现问题，或有任何想法（改进题目、调整任务量等等），请及时联系助教

Q1. 针对以下文法 *G[S]* 是否可以设计一个确定的自顶向下预测分析过程？如果可以，需要向前查看多少个输入符号？

1.
$$S \to A \mid B$$
$$A \to aAb \mid c$$
$$B \to aBbb \mid d$$

2.
$$S \to abA \mid \epsilon$$
$$A \to Saa \mid b$$

参考答案：

1. 不可以。

2. 可以，向前查看 2 个输入符号。

Q2. 考虑下列文法 *G[S]*：

$$S \to T < U \mid b > U$$
$$T \to aS < S \mid cU \mid \ > b$$
$$U \to > Ta \mid \ < Sb$$

1. 计算每个非终结符的 *First* 集合、*Follow* 集合。

2. 计算每个产生式的预测集合（*PS*）。该文法是否是 *LL(1)* 文法？说明原因。

3. 画出该文法的 *LL(1)* 分析预测表。

4. 现有输入符号串 $> b <> > ba$，运行表驱动 *LL(1)* 分析程序，请画出该过程的下推栈。

5. 设计该文法的递归下降分析程序，下面给出了相关函数的声明，其中 `matchToken(char)` 的实现与课件一致。请写出 `parseS()` 的定义。

```
static char lookahead;
void parseS();
void parseT();
void parseU();
void matchToken(char expected);
```

参考答案：

1.

$$First(S) = \{a, b, c, >\}$$
$$First(T) = \{a, c, >\}$$
$$First(U) = \{<, >\}$$
$$Follow(S) = \{a, b, <, \#\}$$
$$Follow(T) = \{a, <\}$$
$$Follow(U) = \{a, b, <, \#\}$$

2.

$$PS(S \to T < U) = \{a, c, >\}$$
$$PS(S \to b > U) = \{b\}$$
$$PS(T \to aS < S) = \{a\}$$
$$PS(T \to cU) = \{c\}$$
$$PS(T \to > b) = \{>\}$$
$$PS(U \to > Ta) = \{>\}$$
$$PS(U \to < Sb) = \{<\}$$

该文法是 LL(1) 文法，因为：（注意 T 的三条产生式的 PS 是两两相交为空，不是三个相交为空）

$$PS(S \to T < U) \cap PS(S \to b > U) = \emptyset$$
$$PS(T \to aS < S) \cap PS(T \to cU) = \emptyset$$
$$PS(T \to aS < S) \cap PS(T \to > b) = \emptyset$$
$$PS(T \to cU) \cap PS(T \to > b) = \emptyset$$
$$PS(U \to > Ta) \cap PS(U \to < Sb) = \emptyset$$

3. 如下表所示。

	a	b	c	$<$	$>$	$\#$
S	$T < U$	$b > U$	$T < U$		$T < U$	
T	$aS < S$		cU		$> b$	
U				$< Sb$	$> Ta$	

4. 如下表所示。

下推栈	余留符号串	下一步动作
$\#S$	$>b<>>ba\#$	应用产生式 $S \to T < U$
$\#U < T$	$>b<>>ba\#$	应用产生式 $T \to > b$
$\#U < b >$	$>b<>>ba\#$	匹配栈顶和当前输入符号
$\#U < b$	$b<>>ba\#$	匹配栈顶和当前输入符号
$\#U <$	$<>>ba\#$	匹配栈顶和当前输入符号
$\#U$	$>>ba\#$	应用产生式 $U \to > Ta$
$\#aT >$	$>>ba\#$	匹配栈顶和当前输入符号
$\#aT$	$>ba\#$	应用产生式 $T \to > b$
$\#ab >$	$>ba\#$	匹配栈顶和当前输入符号
$\#ab$	$ba\#$	匹配栈顶和当前输入符号
$\#a$	$a\#$	匹配栈顶和当前输入符号
$\#$	$\#$	结束

5. 如下所示。（不必拘泥于语法细节）

```
void parseS() {
    switch lookahead {
        case 'a':
        case 'c':
        case '>':
            parseT();
            matchToken('<');
            parseU();
            break;
        case 'b':
            matchToken('b');
            matchToken('>');
            parseU();
            break;
        default:
            printf("syntax error\n");
            exit(1);
    }
}
```

Q3. 考虑下列 *LL(1)* 文法 *G[S]*:

$$S \to P$$
$$P \to \wedge PP \mid \vee PP \mid \neg P \mid \underline{id}$$

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。现有输入符号串：$\vee \vee a \wedge bc \vee \neg a \wedge cb$。

1. 在针对该符号串的表驱动 LL(1) 分析过程中，分析栈中最多会出现几个 S、几个 P ？

2. 若因误操作使输入串多了一个符号，变为 $\vee\vee a\wedge bcc\vee\neg a\wedge cb$，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

3. 如果想要从该出错位置恢复分析，可以进行什么操作？

参考答案：

1. 最多会出现 1 个 S, 3 个 P。

2. 缺运算数。说明：读入第二个 c 时出错，可以直接报“多运算数”（多第二个 c）；若不遇到第二个 c，而是遇到一个运算符，则不会出错，所以可以报“缺运算符”；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”。

3. 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。

Q4. 按要求进行文法变换：

1. 提取左公因子：

$$S\to T\mid T+T\mid T*T$$
$$T\to Ta\mid Tb\mid cU$$
$$U\to U0\mid U1\mid\epsilon$$

2. 按照 S,U,T 的顺序消除左递归：

$$S\to S+S\mid(S)\mid T$$
$$T\to UUb\mid Ta$$
$$U\to TTc\mid c$$

参考答案：

1.

$$S\to TS'$$
$$S'\to +T\mid *T\mid\epsilon$$
$$T\to TT'\mid cU$$
$$T'\to a\mid b$$
$$U\to UU'\mid\epsilon$$
$$U\to 0\mid 1$$

2. 对于 S，消除直接左递归：

$$S\to(S)S'\mid TS'$$
$$S'\to +SS'\mid\epsilon$$
$$T\to UUb\mid Ta$$
$$U\to TTc\mid c$$

对于 U，暂时无需消除；
对于 T，首先用 $T \to TTcUb \mid cUb$ 替代 $T \to UUb$，再消除直接左递归：

$$
\begin{aligned}
S &\to (S)S' \mid TS' \\
S' &\to +SS' \mid \epsilon \\
T &\to cUbT' \\
T' &\to TcUbT' \mid aT' \mid \epsilon \\
U &\to TTc \mid c
\end{aligned}
$$