语法分析器的设计与实现

姓名：黄昱衡 学号：2017210445（02） 班级：2017211301

一. 实验目的

理解语法分析器在编译器中的作用。

通过手动实现语法分析器，来加深对语法分析部分各算法的理解。

二. 实验要求

编写语法分析程序，实现对算术表达式的语法分析。要求所分析算术表达式由如下的文法产生：

E → E+T | E-T | T

T → T*F | T/F | F

F → (E) | num

实验要求在对输入的算术表达式进行分析的过程中，依次输出所采用的产生式。

三. 实验环境

Operating system：macOS Mojave 10.14.5

IDE：Clion

Language standard：CMAKE_CXX_STANDARD 14

四. 实验内容

本次实验将编写语法分析程序实现自底向上的分析。要完成：

(1) 构造识别文法所有活前缀的DFA

(2) 构造该文法的LR分析表

(3) 编写实现算法4.3， 构造LR分析程序。

经过手动计算，该实验的文法是SLR(1) 文法，仅需要构造SLR(1) 分析表以及相应的分析程序即可。为了加深自己对语法分析器的理解，本次实验**构造DFA，构造分析表，LR分析程序**这三个步骤**均用代码实现了**。虽然为了简化调试，程序里已经写好题目要求的文法，但只要稍微改一下输入，就可以通过调用函数接口，针对所有的SLR(1)文法进行分析。即只要求输入SLR(1) 文法，不要求输入DFA和LR分析表。实现这整个过程的工作量要远大于仅仅实现算法4.3。因为需要完成如下几个过程：

(1) 程序根据所输入的文法，计算每个非终结符的First集和Follow集。这样做是因为从DFA构造LR分析表时，对于规约项目需要查询相应的产生式左端符号的Follow集。

(2) 程序根据所输入的文法，计算识别活前缀的DFA。DFA的构建过程，实际上是一个图(树)结构的构建过程。其中包含节点和边。

(3) 程序根据生成的DFA，构造相应的LR分析表。

(4) 程序根据构造的LR分析表，生成相应的分析程序。

下面详细解释程序的实现

五. 实现过程

首先是对输入的转义。NUM在文法中表示数字0-9，ε为空。这两个字符难以用ASCII码表示出来。因此对其进行转义:

```
#define NUM '@'
#define null '#'
```

若以上符号在文法中有实际意义，可以根据宏定义进行更改。

接下来是对各模块的概述:

(1) 计算First集和Follow集

结构体:

```
struct symbolSet
{
    char nonterminal;
    int symbolCount;
    char terminal[terminalNum];
};
```

用以表示一个非终结符的first集或follow集。nonterminal表示非终结符，symbolCount表示该集合中有几个非终结符，terminal存储终结符。

First集

对于First集，其计算过程如下所示:

```
若X∈VT,则FIRST(X)={X};
若X∈VN,且有产生式X→a…,其中 a∈VT,则把a加入到FIRST(X)中;
若X→ε也是产生式,则 ε 也加入到FIRST(X)中。
若X→Y…是产生式,且Y∈VN,则把FIRST(Y)中的所有非ε元素加入到FIRST(X)中;

若X→Y1Y2…Yk是产生式,如果对某个i, FIRST(Y1)、FIRST(Y2)、…、FIRST(Yi-1)都含有ε, 即
Y1Y2…Yi-1    ε, 则把FIRST(Yi)中的所有非ε元素加入到FIRST(X)中;
若所有FIRST(Yi)均含有ε,其中i=1、2、…、k,则把 ε 加入到FIRST(X)中。
```

代码中使用:

```
void get_firstSet(int num)
```

实现，其中num表示产生式的数量。

Follow集

对于Follow集，其计算过程如下所示：

```
对文法开始符号S，置$于FOLLOW(S)中，$为输入符号串的右尾标志。

若A→αBβ是产生式，则把FIRST(β)中的所有非ε元素加入到FOLLOW(B)中。

若A→αB是产生式，或A→αBβ是产生式，并且β最终能推出ε，则把FOLLOW(A)中的所有元素加入到FOLLOW(B)中。

重复此过程，直到所有集合不再变化为止。
```

Follow集的计算比First集要略微复杂一些。具体实现时分为两步：

1. 首先看产生式右端是否是非终结符号。如果是，将左端的Follow集赋给右端。并且检查右端末尾符号是否能推出空，若可以，则再对右端倒数第二个符号进行检查，如此递归，直到将产生式右端全部遍历完，或者遇到终结符，或者非终结符无法推出空为止。
2. 对产生式右端进行一次扫描，如果扫描到非终结符，检查它的后一个符号的First集（如果有了话），然后将该First集加入到对应的Follow集当中。

注意无论first集还是follow集的求解，其求解子函数都会返回一个bool变量，用以确定各集合是否还有变化，以此来作为循环的退出条件。

代码中使用：

```
void get_follow_set(int num)
```

实现，其中num表示产生式个数。

运行程序，分析题目要求的文法，其First集和Follow集的结果如下：

```
First集
symbol:S    ( NUM
symbol:E    ( NUM
symbol:T    ( NUM
symbol:F    ( NUM
Folllow集
symbol:S    $
symbol:E    $ + - )
symbol:T    $ + - * / )
symbol:F    $ + - * / )
```

(2) 识别活前缀的DFA

结构体：

```
struct item
{
    expression expr;
    int point_index = 0;
    bool checked = 0;
};
```

存储LR(0)项目。expr是产生式，point_index表示产生式圆点的位置。checked表示该项目在计算DFA时有没有被遍历到。其中，产生式是由结构体：

```
struct expression
{
    char left[2];
    char right[8];
};
```

表示的。

LR(0)有效项目集由结构体：

```
struct itemset
{
    item kernel[maxItemNum];
    int kernelSize = 0;
    item closure[maxItemNum];
    int closureSize = 0;
};
```

表示。其中kernel存储了项目集的核（即LALR(1)文法中的概念），closure存储了包含核在内整个项目集的闭包。存储核的好处在于。当比较两个项目集是否相等时，只需比较核是否相等即可。可以提高效率。

由于DFA可看做是一个图，因此还需要节点和边的数据结构：

```
struct relation
{
    char gotoToken;
    int gotoEntity;
};
struct entity
{
    relation relations[8];
    int relationNum = 0;
};
```

relation表示边，entity表示节点。entity中存储了节点指向外部的边。

计算闭包

构造DFA时，首先需要计算每个项目集的闭包，其计算过程大致如下：

```
输入:项目集合I。
输出:集合J=closure(I)。
方法:
J=I;
do {
 J_new=J;
 for(J_new中的每一个项目A→α·Bβ 和 文法G的每个产生式B→η)
   if (B→·η∉J)
     把B→·η加入J;
} while (J_new!=J).
```

该过程由函数：

```
void get_closure(int tempIndex)
```

实现。其中tempIndex表示项目集的编号。

计算DFA

寻找能够识别文法所有活前缀的DFA，就是计算文法的LR(0)项目集规范族。再加上每个项目集之间的转移关系。

该过程由函数：

```
void get_DFA()
{
    int index = 0;
    do{
        get_closure(index);
        while(get_new_itemSet(index) != -1);
        index ++;
    }while(index < itemSetIndex);
}
```

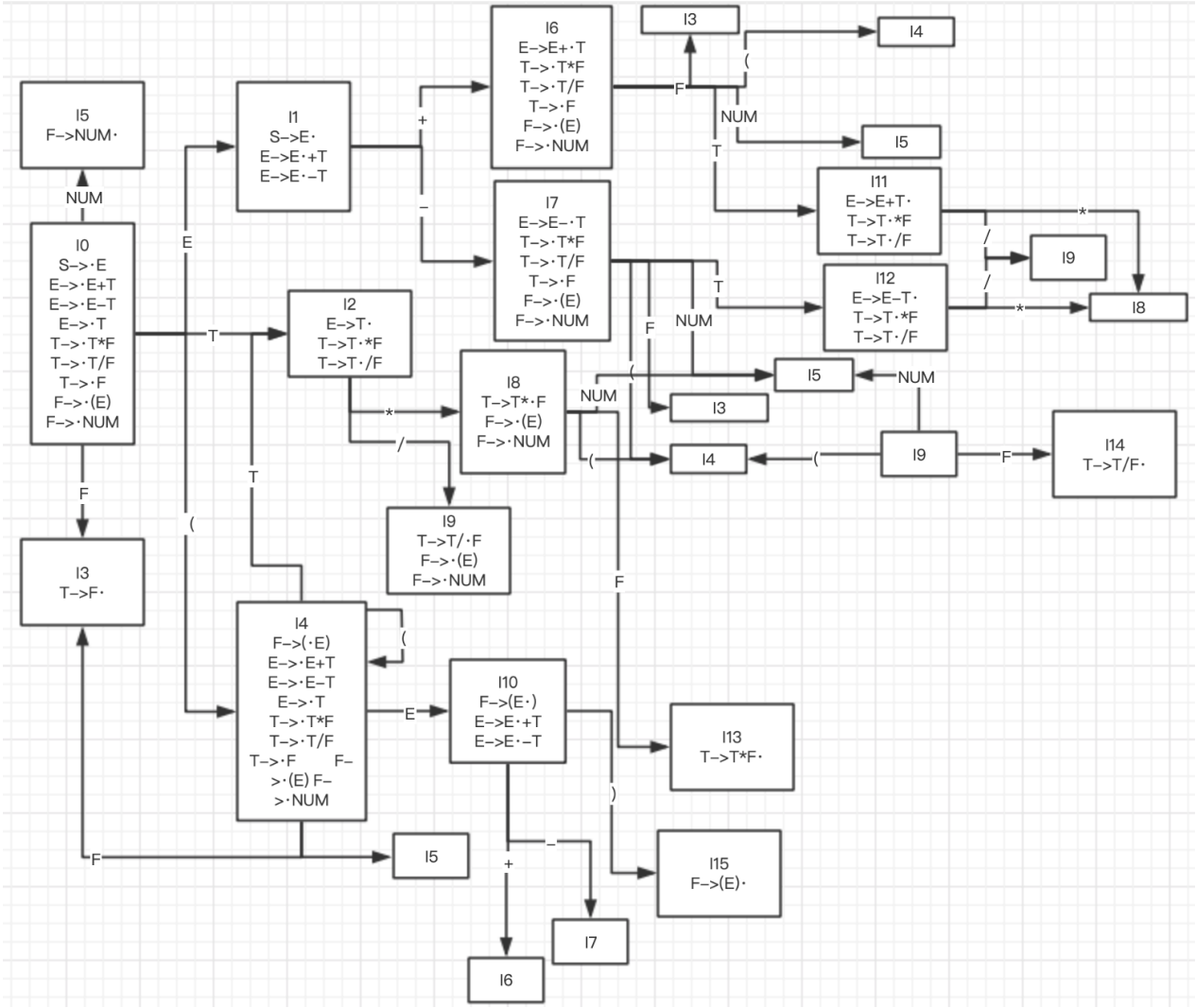
实现。其中itemSetIndex是全局变量，用以记录目前项目集的数量。index = itemSetIndex时就是没有新项目加入的时候，此时退出循环。

结果如下：

```
项目规范集:0  S->·E   E->·E+T E->·E-T E->·T   T->·T*F T->·T/F T->·F   F->·(E) F->·NUM
项目规范集:1  S->E    E->E·+T E->E·-T
项目规范集:2  E->T    T->T·*F T->T·/F
项目规范集:3  T->F
项目规范集:4  F->(·E) E->·E+T E->·E-T E->·T   T->·T*F T->·T/F T->·F   F->·(E) F->·NUM
项目规范集:5  F->NUM
项目规范集:6  E->E+·T T->·T*F T->·T/F T->·F   F->·(E) F->·NUM
项目规范集:7  E->E-·T T->·T*F T->·T/F T->·F   F->·(E) F->·NUM
项目规范集:8  T->T*·F F->·(E) F->·NUM
项目规范集:9  T->T/·F F->·(E) F->·NUM
项目规范集:10     F->(E·) E->E·+T E->E·-T
项目规范集:11     E->E+T  T->T·*F T->T·/F
项目规范集:12     E->E-T  T->T·*F T->T·/F
项目规范集:13     T->T*F
项目规范集:14     T->T/F
项目规范集:15     F->(E)
```

```
项目规范集0的goto关系如下:0--E--1  0--T--2 0--F--3 0--(--4 0--NUM--5
项目规范集1的goto关系如下:1--+--6  1-----7
项目规范集2的goto关系如下:2--*--8  2--/--9
项目规范集3的goto关系如下:
项目规范集4的goto关系如下:4--E--10 4--T--2 4--F--3 4--(--4 4--NUM--5
项目规范集5的goto关系如下:
项目规范集6的goto关系如下:6--T--11 6--F--3 6--(--4 6--NUM--5
项目规范集7的goto关系如下:7--T--12 7--F--3 7--(--4 7--NUM--5
项目规范集8的goto关系如下:8--F--13 8--(--4 8--NUM--5
项目规范集9的goto关系如下:9--F--14 9--(--4 9--NUM--5
项目规范集10的goto关系如下:10--)--15   10--+--6    10-----7
项目规范集11的goto关系如下:11--*--8    11--/--9
项目规范集12的goto关系如下:12--*--8    12--/--9
项目规范集13的goto关系如下:
项目规范集14的goto关系如下:
项目规范集15的goto关系如下:
```

以上关系所表示的DFA即：



(3) 构造SLR(1)分析表

分析表的构造，即算法4.6：

输入:拓广文法G′　　输出:G′的SLR分析表

1.构造G′的LR(0)项目集规范族C={I0,I1,…In}。
2.对于状态i(对应于项目集Ii的状态)的分析动作如下

- a) 若A→α·aβ∈Ii,且go(Ii,a)=Ij,则置action[i,a]=sj
- b) 若A→α·∈Ii,则对所有a∈FOLLOW(A), 置action[i,a]=r A→α
- c) 若S′→S· ∈Ii,则置action[i,$]=acc,表示分析成功

3.若go(Ii,A)=Ij,A为非终结符号,则置goto[i,A]=j

4.分析表中凡不能用规则(2)、(3)填入信息的空白表项,均置为出错标志error。

```
5.分析程序的初态是包含项目S'→S的有效项目集所对应的状态。
```

实际实现时，首先处理移进-待约项目，再处理规约项目。该过程由函数：

```
void get_SLR_table()
```

实现。

结果如下：

SLR(1)分析表：

状态	+	-	*	/	(	)	NUM	$	|E	T	F
状态0					s4		s5		1	2	3
状态1	s6	s7						acc	0	0	0
状态2	r3	r3	s8	s9		r3		r3	0	0	0
状态3	r6	r6	r6	r6		r6		r6	0	0	0
状态4					s4		s5		10	2	3
状态5	r8	r8	r8	r8		r8		r8	0	0	0
状态6					s4		s5		0	11	3
状态7					s4		s5		0	12	3
状态8					s4		s5		0	0	13
状态9					s4		s5		0	0	14
状态10	s6	s7				s15			0	0	0
状态11	r1	r1	s8	s9		r1		r1	0	0	0
状态12	r2	r2	s8	s9		r2		r2	0	0	0
状态13	r4	r4	r4	r4		r4		r4	0	0	0
状态14	r5	r5	r5	r5		r5		r5	0	0	0
状态15	r7	r7	r7	r7		r7		r7	0	0	0

经过手动验算结果正确。

(4) 分析程序

LR分析控制程序，即算法4.3：

```
输入:文法G的一张分析表和一个输入符号串ω
输出:若ω∈L(G),得到ω的自底向上的分析,否则报错
方法:开始时,初始状态S0在栈顶,ω$在输入缓冲区中。

置ip指向ω$的第一个符号;
do {
  令S是栈顶状态,a是ip所指向的符号
    if (action[S,a]==shift S') {
      把a和S'分别压入符号栈和状态栈;
      推进ip,使它指向下一个输入符号;
    } else if (action[S,a]==reduce by A→β) {
      从栈顶弹出|β|个符号; (令S'是现在的栈顶状态)
      把A和goto[S',A]分别压入符号栈和状态栈;
      输出产生式A→β;
    }else if (action[S,a]==accept)
      return;
  else
```

```
    error();
} while(1).
```

该算法由函数：

```
bool LR_analysis(char* str)
```

实现。

六. 程序测试

现对整个程序进行检测：

(1)3+5

```
please input your string
3+5
栈   输入 分析动作
0
    3+5$    s5
05
 3  +5$ reduce by F->@  goto 3
03
 F  +5$ reduce by T->F  goto 2
02
 T  +5$ reduce by E->T  goto 1
01
 E  +5$ s6
016
 E+ 5$  s5
0165
 E+5    $   reduce by F->@  goto 3
0163
 E+F    $   reduce by T->F  goto 11
01611
 E+T    $   reduce by E->E+T    goto 1
01
 E  $   acc
accept

Process finished with exit code 0
```

(1) 3*(3+5)

```
please input your string
3*(3+5)
栈   输入 分析动作
0
    3*(3+5)$    s5
05
 3  *(3+5)$ reduce by F->@  goto 3
03
 F  *(3+5)$ reduce by T->F  goto 2
02
 T  *(3+5)$ s8
028
 T* (3+5)$  s4
0284
 T*(    3+5)$   s5
02845
 T*(3   +5)$    reduce by F->@  goto 3
02843
 T*(F   +5)$    reduce by T->F  goto 2
02842
 T*(T   +5)$    reduce by E->T  goto 10
028410
 T*(E   +5)$    s6
0284106
 T*(E+  5)$ s5
02841065
 T*(E+5 )$  reduce by F->@  goto 3
02841063
 T*(E+F )$  reduce by T->F  goto 11
028410611
 T*(E+T )$  reduce by E->E+T    goto 10
028410
 T*(E   )$  s15
02841015
 T*(E)  $   reduce by F->(E)    goto 13
02813
 T*F    $   reduce by T->T*F    goto 2
02
 T  $   reduce by E->T  goto 1
01
 E  $   acc
accept

Process finished with exit code 0
```

(符号@表示NUM)

(2) (5+6)/(3*7)*2

```
please input your string
(5+6)/(3*7)*2
栈   输入 分析动作
0
    (5+6)/(3*7)*2$  s4
04
 (  5+6)/(3*7)*2$   s5
045
 (5 +6)/(3*7)*2$    reduce by F->@  goto 3
043
 (F +6)/(3*7)*2$    reduce by T->F  goto 2
042
 (T +6)/(3*7)*2$    reduce by E->T  goto 10
0410
 (E +6)/(3*7)*2$    s6
04106
 (E+    6)/(3*7)*2$ s5
041065
 (E+6   )/(3*7)*2$  reduce by F->@  goto 3
041063
 (E+F   )/(3*7)*2$  reduce by T->F  goto 11
0410611
 (E+T   )/(3*7)*2$  reduce by E->E+T    goto 10
0410
 (E )/(3*7)*2$  s15
041015
 (E)    /(3*7)*2$   reduce by F->(E)    goto 3
03
 F  /(3*7)*2$   reduce by T->F  goto 2
02
 T  /(3*7)*2$   s9
029
 T/ (3*7)*2$    s4
0294
 T/(    3*7)*2$ s5
02945
 T/(3   *7)*2$  reduce by F->@  goto 3
02943
 T/(F   *7)*2$  reduce by T->F  goto 2
```

```
02942
 T/(T   *7)*2$  s8
029428
 T/(T*  7)*2$   s5
0294285
 T/(T*7 )*2$    reduce by F->@  goto 13
02942813
 T/(T*F )*2$    reduce by T->T*F    goto 2
02942
 T/(T   )*2$    reduce by E->T  goto 10
029410
 T/(E   )*2$    s15
02941015
 T/(E)  *2$ reduce by F->(E)    goto 14
02914
 T/F    *2$ reduce by T->T/F    goto 2
02
 T  *2$ s8
028
 T* 2$  s5
0285
 T*2    $   reduce by F->@  goto 13
02813
 T*F    $   reduce by T->T*F    goto 2
02
 T  $   reduce by E->T  goto 1
01
 E  $   acc
accept
```

(3) (5+6)/

```
please input your string
5+6/
栈  输入 分析动作
0
    5+6/$   s5
05
 5  +6/$    reduce by F->@  goto 3
03
 F  +6/$    reduce by T->F  goto 2
02
 T  +6/$    reduce by E->T  goto 1
01
 E  +6/$    s6
016
 E+ 6/$ s5
0165
 E+6    /$  reduce by F->@  goto 3
0163
 E+F    /$  reduce by T->F  goto 11
01611
 E+T    /$  s9
016119
 E+T/   $   Not accept
```

(4) (5-6)*(7+

```
please input your string
(5-6)*(7+
栈   输入 分析动作
0
    (5-6)*(7+$  s4
04
 (  5-6)*(7+$   s5
045
 (5 -6)*(7+$    reduce by F->@  goto 3
043
 (F -6)*(7+$    reduce by T->F  goto 2
042
 (T -6)*(7+$    reduce by E->T  goto 10
0410
 (E -6)*(7+$    s7
04107
 (E-    6)*(7+$ s5
041075
 (E-6   )*(7+$  reduce by F->@  goto 3
041073
 (E-F   )*(7+$  reduce by T->F  goto 12
0410712
 (E-T   )*(7+$  reduce by E->E-T    goto 10
0410
 (E )*(7+$  s15
041015
 (E)    *(7+$   reduce by F->(E)    goto 3
03
 F  *(7+$   reduce by T->F  goto 2
02
 T  *(7+$   s8
028
 T* (7+$    s4
0284
 T*(    7+$ s5
02845
 T*(7   +$  reduce by F->@  goto 3
02843
 T*(F   +$  reduce by T->F  goto 2
02842
 T*(T   +$  reduce by E->T  goto 10
028410
 T*(E   +$  s6
0284106
 T*(E+  $   Not accept

Process finished with exit code 0
```

程序运行良好。

七. 实验总结

本次语法分析器的设计从First，Follow集的求解，到最终分析程序的构造，完整实现了整个SLR(1)的分析过程。工作量比预想的要大，大概有560行代码。但是实现的相对顺利，完成后调试BUG的时间比较少。这得益于一开始良好的设计过程，以及在编写代码时对每个函数良好的模块化测试。此次试验收获很多，既加深了自己对SLR(1)分析器的理解，又加强了自己的编程能力，是一次非常不错的体验。