**实验二：语法分析程序的设计与实现**

**实验内容**

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

**实验要求**

* 基本要求：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
* 语言：C/C++
* 方法：
* 方法1：编写递归调用程序实现自顶向下的分析
* **方法2：编写LL(1)语法分析程序，要求如下（必做）**
* **基础要求（必做）**
* **编程实现算法4.2，为给定文法自动构造预测分析表，给出程序运行结果**
* **编程实现算法4.1，构造LL(1)预测分析程序，在分析过程中依次输出所用的产生式**
* **方法3：编写语法分析程序实现自底向上的分析，要求如下（必做）**
* **基础要求（必做）：**
* **构造识别该文法所有活前缀的DFA**
* **构造该文法的LR(1)分析表**
* **编程实现算法4.3，构造LR分析程序**

**文法信息**

文法含有左递归，故不是LL(1)文法

改写文法： 消除左递归

FIRST和FOLLOW集合：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 计算FIRST | FIRST | 计算FOLLOW | FOLLOW |
| E | FIRST(T) | (, num | {$, )} | $, ) |
| E' | {+, -, ε} | ε, +, - | FOLLOW(E) | $, ) |
| T | FIRST(F) | (, num | FIRST(E') + FOLLOW(E) + FOLLOW(E') | $, ), +, - |
| T' | {\*, /, ε} | ε, \*, / | FOLLOW(T) | $, ), +, - |
| F | {(, num} | (, num | FIRST(T') + FOLLOW(T) + FOLLOW(T') | $, ), +, -, \*, / |

LL(1)分析表：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | num | + | - | \* | / | ( | ) | $ |
| E |  |  |  |  |  |  |  |  |
| E' |  |  |  |  |  |  |  |  |
| T |  |  |  |  |  |  |  |  |
| T' |  |  |  |  |  |  |  |  |
| F |  |  |  |  |  |  |  |  |

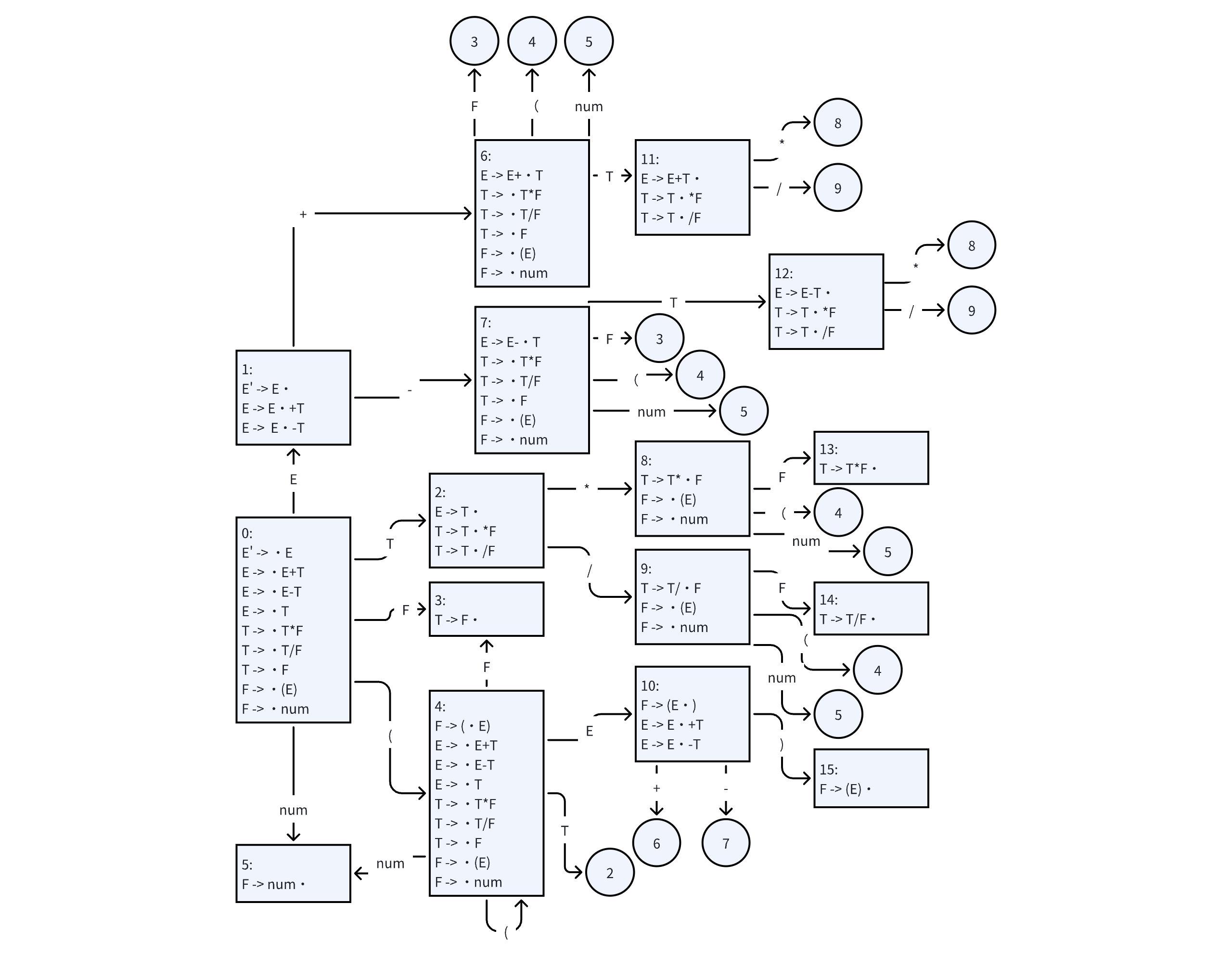
拓广文法：

对其编号：

FIRST和FOLLOW集合：

|  |  |  |  |
| --- | --- | --- | --- |
|  | FIRST | 计算FOLLOW | FOLLOW |
| E | (, num | {$, +, -, )} | $, +, -, ) |
| T | (, num | FOLLOW(E) + {\*, /} | $, +, -, ), \*, / |
| F | (, num | FOLLOW(T) | $, +, -, ), \*, / |

构造识别所有活前缀的DFA



SLR(1)分析表：

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | action | | | | | | | | goto | | |
|  | + | - | \* | / | ( | ) | num | $ | E | T | F |
| 0 |  |  |  |  | S4 |  | S5 |  | 1 | 2 | 3 |
| 1 | S6 | S7 |  |  |  |  |  | ACC |  |  |  |
| 2 | R3 | R3 | S8 | S9 |  | R3 |  | R3 |  |  |  |
| 3 | R6 | R6 | R6 | R6 |  | R6 |  | R6 |  |  |  |
| 4 |  |  |  |  | S4 |  | S5 |  | 10 | 2 | 3 |
| 5 | R8 | R8 | R8 | R8 |  | R8 |  | R8 |  |  |  |
| 6 |  |  |  |  | S4 |  | S5 |  |  | 11 | 3 |
| 7 |  |  |  |  | S4 |  | S5 |  |  | 12 | 3 |
| 8 |  |  |  |  | S4 |  | S5 |  |  |  | 13 |
| 9 |  |  |  |  | S4 |  | S5 |  |  |  | 14 |
| 10 | S6 | S7 |  |  |  | S15 |  |  |  |  |  |
| 11 | R1 | R1 | S8 | S9 |  | R1 |  | R1 |  |  |  |
| 12 | R2 | R2 | S8 | S9 |  | R2 |  | R2 |  |  |  |
| 13 | R4 | R4 | R4 | R4 |  | R4 |  | R4 |  |  |  |
| 14 | R5 | R5 | R5 | R5 |  | R5 |  | R5 |  |  |  |
| 15 | R7 | R7 | R7 | R7 |  | R7 |  | R7 |  |  |  |

**程序设计说明**

**LL(1)语法分析程序**

**为给定文法自动构造预测分析表**

* **读入文法**

存储文法的数据结构：哈希表

|  |
| --- |
| C typedef **struct** production {   **char** left; *// 左部非终结符*  **char**\* right; *// 右部符号串*  **struct** production \*next; *// 相同非终结符下一产生式* } production;  #define BUCKET\_SIZE 52 *// 假设哈希表的大小为26，即非终结符的个数* #define MAX\_LENGTH 10 *// 假设产生式右部的最大长度为10*  production\* hash\_table[BUCKET\_SIZE]; *// 哈希表是一个指向产生式的指针数组* |

相关函数

|  |
| --- |
| C void read\_into\_grammar(); // 读入文法 **int** hash(**char** c); *// 哈希函数定义* **void** init\_hash\_table(); *// 初始化哈希表，将所有指针设为NULL* **void** insert\_production(**char** left, **char**\* right); *// 插入一个产生式到哈希表中，如果已经存在，则忽略* production\* find\_production(**char** left, **char**\* right); *// 查找一个产生式是否在哈希表中，如果存在，则返回指向该产生式的指针，否则返回NULL* **void** delete\_production(**char** left, **char**\* right); *// 删除一个产生式从哈希表中，如果不存在，则忽略* **void** print\_hash\_table(); *// 打印哈希表的内容* |

输入文法的输出：

|  |
| --- |
| C Read the grammar successfully: [E -> E+T]  [E -> E-T] [E -> T] [F -> (E)] [F -> num] [T -> T\*F] [T -> T/F] [T -> F] |

* **消除左递归**

消除逻辑：判断有左递归则消除，没有则直接打印出文法

|  |
| --- |
| C if(has\_direct\_left\_recursion()){  printf("The grammer has direct left recursion!\n");  remove\_direct\_left\_recursion(); } printf("The grammer's left recursion has been removed:\n"); print\_hash\_table(); |

相关函数：

|  |
| --- |
| C **int** has\_direct\_left\_recursion(); *// 判断一个文法是否含有直接左递归* **void** remove\_direct\_left\_recursion(); *// 消除一个文法的直接左递归* |

消除左递归过程的输出：

|  |
| --- |
| C The grammer has direct left recursion! The grammer's left recursion has been removed: [E -> Te] [F -> (E)] [F -> num] [T -> Ft] [e -> +Te] [e -> -Te] [e -> ] [t -> \*Ft] [t -> /Ft] [t -> ] |

* **求FIRST集合**

|  |  |  |
| --- | --- | --- |
|  | 计算FIRST | FIRST |
| E | FIRST(T) | (, num |
| E' | {+, -, ε} | ε, +, - |
| T | FIRST(F) | (, num |
| T' | {\*, /, ε} | ε, \*, / |
| F | {(, num} | (, num |

结构及函数定义：

|  |
| --- |
| C **char** FIRST[52][10]; *//FIRST集合* **void** compute\_first(); *// 计算文法的FIRST集合* **void** print\_first(); *//打印FIRST集合* **void** init\_first(); //FIRST集合初始化 **void** get\_first(**char** symbol, **char** toadd); // 将要symbol的FIRST集合加入tooadd的FIRST集合 |

求FIRST集合过程输出：

|  |
| --- |
| C This is the first set: FIRST[E]: (n FIRST[F]: (n FIRST[T]: (n FIRST[e]: +-~ FIRST[t]: \*/~ |

* **求FOLLOW集合**

|  |  |  |
| --- | --- | --- |
|  | 计算FOLLOW | FOLLOW |
| E | {$, )} | $, ) |
| E' | FOLLOW(E) | $, ) |
| T | FIRST(E') + FOLLOW(E) + FOLLOW(E') | $, ), +, - |
| T' | FOLLOW(T) | $, ), +, - |
| F | FIRST(T') + FOLLOW(T) + FOLLOW(T') | $, ), +, -, \*, / |

结构及函数定义：

|  |
| --- |
| C **char** FOLLOW[52][10]; // FIRST集合 **void** compute\_follow(); // 计算follow集合 **void** get\_follow(**char** symbol, **char** toadd); // 将symbol的follow集合加入tooadd的follow集合 **void** add\_first\_to\_follow(**char** symbol, **char** toadd); // 将symbol的first加入tooadd的follow **void** print\_follow(); // 打印follow集合 |

求FOLLOW集合过程输出：

|  |
| --- |
| C This is the follow set: FOLLOW[E]: $) FOLLOW[F]: $)+-\*/ FOLLOW[T]: $)+- FOLLOW[e]: $) FOLLOW[t]: $)+- |

* **构造预测分析表**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | num | + | - | \* | / |
| E |  |  |  |  |  |
| E' |  |  |  |  |  |
| T |  |  |  |  |  |
| T' |  |  |  |  |  |
| F |  |  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | ( | ) | $ |
| E |  |  |  |
| E' |  |  |  |
| T |  |  |  |
| T' |  |  |  |
| F |  |  |  |

使用算法：

|  |
| --- |
| C 输入：文法G 输出：文法G的预测分析表M 方法： for (文法G的每个产生式 A-> α ) {  for (每个终结符号 a∈FIRST(α))  把 A->α 放入 M[A, a] 中;  if (ε∈FIRST(α))  for (任何 b∈FOLLOW(A))  把 A->α 放入 M[A, b] 中;  }； for (所有无定义的M[A, a]) 标上错误标志。 |

|  |
| --- |
| 调试日志：11月23日完成分析表构造。 |

分析表构造过程输出：（用e代替E'）

|  |
| --- |
| C This is the analysis table: M[E]: M[E][n]:E->Te M[E][(]:E->Te  M[F]: M[F][n]:F->num M[F][(]:F->(E)  M[T]: M[T][n]:T->Ft M[T][(]:T->Ft  M[e]: M[e][+]:e->+Te M[e][-]:e->-Te M[e][)]:e-> M[e][$]:e->  M[t]: M[t][+]:t-> M[t][-]:t-> M[t][\*]:t->\*Ft M[t][/]:t->/Ft M[t][)]:t-> M[t][$]:t-> |

**LL(1)预测分析程序**

首先进行词法分析，从input.txt读入输入字符串，例如：1+2-37\*(2+4)，得到记号流：n+n-n\*(n+n)。

|  |
| --- |
| C lexical\_analysis(); printf("This is the token flow:\n"); print\_token(); |

词法分析过程输出：

|  |
| --- |
| C This is the string: 1\*2-(3/4+(543-321)) This is the token flow: n\*n-(n/n+(n-n)) |

预测分析程序使用算法：

|  |
| --- |
| C 输入：输入符号串Ω，文法G的一张预测分析表M。 输出：若Ω在L(G)中，则输出Ω的最左推导，否则报告错误。 方法：分析开始时，$在栈底，文法开始符号S在栈顶，Ω$在输入缓冲区中  置ip指向Ω$ 的第一个符号；  do {  令X是栈顶符号，a是ip所指向的符号；  if (X是终结符号或$) {  if (X==a) {  从栈顶弹出X; ip前移一个位置;  };  else error();  else /\* X是非终结符号 \*/  if (M[X, a]=XY1Y2…Yk) {  从栈顶弹出X;  把Yk、Yk-1、…、Y2、Y1压入栈，Y1在栈顶;  输出产生式X->Y1Y2…Yk;  };  else error();  } while(X!=$) /\* 栈不空，继续 \*/ |

|  |
| --- |
| 11月24日日志：完成预测分析程序构建 |

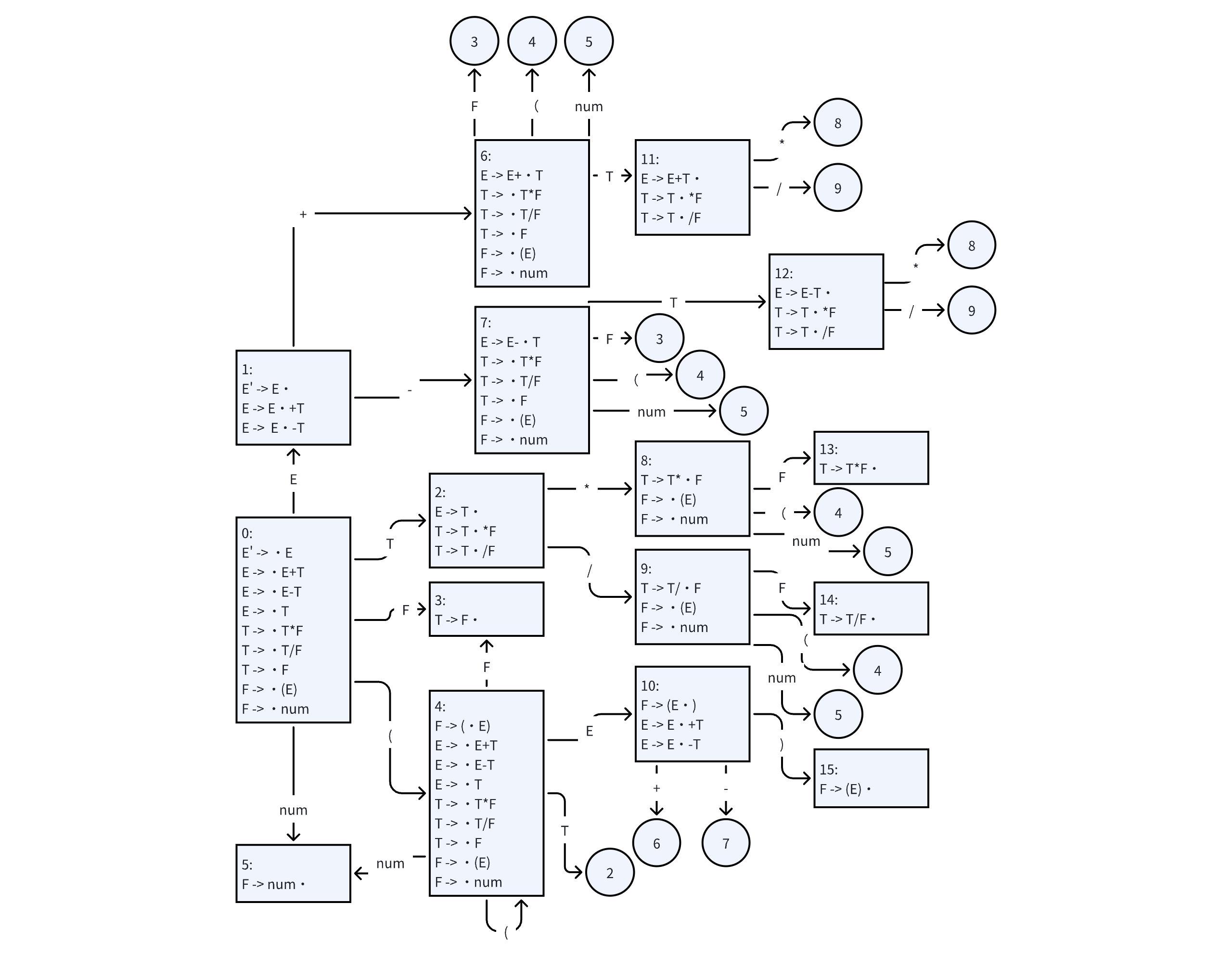
终端输出样例：

|  |
| --- |
| C This is the analysis process: Stack: [$, E] Token: n\*n-(n/n+(n-n))$ E->Te Stack: [$, e, T] Token: n\*n-(n/n+(n-n))$ T->Ft Stack: [$, e, t, F] Token: n\*n-(n/n+(n-n))$ F->num ...... |

产生式序列输出在output.txt中。

**自底向上语法分析程序**

**构造识别该文法所有活前缀的DFA**



自动机结构定义：

|  |
| --- |
| C *// 定义项目* **struct** Item {  **char** left;  **char** right[10];  **int** dotPosition; };  *// 定义状态* **struct** State {  **struct** Item items[10];  **int** numItems; }; |

构造识别该文法所有活前缀DFA状态转移表go，为构造分析表做准备，从下面的语句获取状态 i 的所有项目：

|  |
| --- |
| C *// 遍历所有状态* for(**int** i = 0; i < NUM\_STATES; i++){  *// 对于状态i的分析动作如下*  *// printState(&state[i]);*  for(**int** j = 0; j <(state[i].numItems); j++) {  **struct** Item item = getItem(&state[i], j);  printItem(&state[i], j);  } } |

**构造该文法的LR(1)分析表**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | action | | | | | | | |  | goto |  |
|  | + | - | \* | / | ( | ) | num | $ | E | T | F |
| 0 |  |  |  |  | S4 |  | S5 |  | 1 | 2 | 3 |
| 1 | S6 | S7 |  |  |  |  |  | ACC |  |  |  |
| 2 | R3 | R3 | S8 | S9 |  | R3 |  | R3 |  |  |  |
| 3 | R6 | R6 | R6 | R6 |  | R6 |  | R6 |  |  |  |
| 4 |  |  |  |  | S4 |  | S5 |  | 10 | 2 | 3 |
| 5 | R8 | R8 | R8 | R8 |  | R8 |  | R8 |  |  |  |
| 6 |  |  |  |  | S4 |  | S5 |  |  | 11 | 3 |
| 7 |  |  |  |  | S4 |  | S5 |  |  | 12 | 3 |
| 8 |  |  |  |  | S4 |  | S5 |  |  |  | 13 |
| 9 |  |  |  |  | S4 |  | S5 |  |  |  | 14 |
| 10 | S6 | S7 |  |  |  | S15 |  |  |  |  |  |
| 11 | R1 | R1 | S8 | S9 |  | R1 |  | R1 |  |  |  |
| 12 | R2 | R2 | S8 | S9 |  | R2 |  | R2 |  |  |  |
| 13 | R4 | R4 | R4 | R4 |  | R4 |  | R4 |  |  |  |
| 14 | R5 | R5 | R5 | R5 |  | R5 |  | R5 |  |  |  |
| 15 | R7 | R7 | R7 | R7 |  | R7 |  | R7 |  |  |  |

分析表分为goto和action两部分，在后续LR分析程序中都有重要作用。

**编程实现算法4.3，构造LR分析程序**

从输入文件input.txt中获取输入字符串，首先进行与LL(1)相同的词法分析，之后对记号流使用算法4.3

算法过程：

|  |
| --- |
| C 输入：文法 G 的一张分析表和一个输入符号串w 输出：若w属于L(G)，得到w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，w$ 在输入缓冲区中。 置 ip 指向 w$ 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d;  };  else if (action[S, a]==reduce by A->b) {  从栈顶弹出 |b| 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->b;  };  else if (action[S, a]==accept) return;  else error(); } while(1). |

终端输出示例：

|  |
| --- |
| C This is the analysis process: Status Stack: [0]Symbol Stack: [-] Token: n\*n-(n/n+(n-n))$ Status Stack: [0, 5]Symbol Stack: [-, n] Token: \*n-(n/n+(n-n))$ F -> n Status Stack: [0, 3]Symbol Stack: [-, F] Token: \*n-(n/n+(n-n))$ T -> F ...... |

产生式序列输出在output.txt中。

**测试报告**

**LL(1)语法分析程序测试报告：**

测试用例1：1+1

output.txt输出：

|  |
| --- |
| C E->Te T->Ft F->num t-> e->+Te T->Ft F->num t-> e-> |

测试用例2：1\*2-(3/4+(543-321))

output.txt输出：

|  |
| --- |
| C E->Te T->Ft F->num t->\*Ft F->num t-> e->-Te T->Ft F->(E) E->Te T->Ft F->num t->/Ft F->num t-> e->+Te T->Ft F->(E) E->Te T->Ft F->num t-> e->-Te T->Ft F->num t-> e-> t-> e-> t-> e-> |

**自底向上语法分析程序测试报告：**

测试用例1：1+1

output.txt输出：

|  |
| --- |
| C F->n T->F E->T F->n T->F E->E+T |

测试用例2：1\*2-(3/4+(543-321))

output.txt输出：

|  |
| --- |
| C F->n T->F F->n T->T\*F E->T F->n T->F F->n T->T/F E->T F->n T->F E->T F->n T->F E->E-T F->(E) T->F E->E+T F->(E) T->F E->E-T |