《编译原理与技术》课程实验报告

语法分析程序设计与实现

黎昱彤

学号：2022211414

班级：2022211312

计算机科学与技术

目录

LL(1)语法分析程序 1

1 背景 1

1.1 实验介绍 1

1.2 实验任务 1

2 程序设计说明 1

2.1 实验原理 1

2.2 程序设计 2

2.2.1 构造分析表 2

2.2.2 构造分析程序 5

3 测试报告 6

4 实验总结 7

LR(1)语法分析程序 8

1 背景 8

1.1 实验介绍 8

1.2 实验任务 8

2 程序设计说明 8

2.1 实验原理 8

2.2 程序设计 9

2.2.1 构造分析表 9

2.2.2 构造分析程序 11

3 测试报告 12

4 实验总结 14

LL(1)语法分析程序

1 背景

1.1 实验介绍

《编译原理与技术（第 2 版）》第 4 章的 程序设计 2 (方法 2)

1.2 实验任务

编写一个 LL(1)语法分析程序，能对算数表达式进行语法分析。要求在给定的消除左递归的文法 G'的基础上：

编号	产生式
1	E→TA
2	A→+TA
3	A→-TA
4	A→ε
5	T→FB
6	B→*FB
7	B→/FB
8	B→ε
9	F→(E)
10	F→num

- 实现算法 4.2，为给定文法自动构造预测分析表；
- 实现算法 4.1，构造 LL(1)预测分析程序。

2 程序设计说明

2.1 实验原理

语法分析阶段是编译过程的核心部分，输入为词法分析生成的记号序列，输出为语法分析树。LL(1)分析器是一种自顶向下的递归下降分析器，能够高效地解析符合LL(1)

文法的上下文无关语法。LL(1)文法的关键是：对每一步的分析，仅需查看栈顶符号和下一个输入符号（Lookahead 1）即可确定唯一的推导规则。

2.2 程序设计

递归调用分析的限制是无二义文法。任务给定的文法 G'已消除左递归，也不存在左公因式。本程序不做文法的预处理。

数据结构：

FIRST 集和 FOLLOW 集是 first 和 follow，用二维字符数组存储，每行对应一个非终结符；LL(1)分预测分析表是一个 5 行 9 列的二维数组 ll1table，行对应非终结符，列对应终结符，值为产生式编号；结构体 AnalysisStep 记录分析栈和输入栈，用于跟踪过程。

全局变量：

non_terms 为非终结符；terms 为终结符；productions 为文法的产生式集合；

核心模块：

first()求解 FIRST 集；follow()求解 FOLLOW 集；ll1table()构造 LL(1)预测分析表；ll1analyze(char* expr) 进行 LL(1)预测分析。

2.2.1 构造分析表

预测分析表主要依赖FIRST集和FOLLOW集。实现求解FIRST集和FOLLOW集中的闭包运算，共同结构为：add_to_set函数用来把元素加入集合，while循环来实现连续不断地扫描与构建，设置一个变量update来标记在当前循环中集合是否发生变化。

求解FIRST集

```
Function first()
    update=1
    while(update){
        update=0
```

```
        for each production A -> α:
            if the first symbol of α is a terminal:
                add the terminal to FIRST(A), update?
            else:
                for each symbol X in α:
                    if X is a terminal:
                        add X to FIRST(A), update?
                        break
                    else:
                        add all symbols in FIRST(X) (excluding 'ε') to FIRST(A), update?
                        if 'ε' is not in FIRST(X):
                            break
                if all symbols in α derive 'ε':
                    add 'ε' to FIRST(A)
    }
```

End Function

求解FOLLOW集

Function follow()

```
    add '$' to FOLLOW(E)
    update=1
    while(update){
```

```
        for each production A -> αBβ:
            if B is a non-terminal:
                if β is non-empty:
                    if the first symbol of β is a terminal:
                        add it to FOLLOW(B), update?
                    else:
                        add FIRST(β) (excluding 'ε') to FOLLOW(B), update?
                        if 'ε' is in FIRST(β):
                            add FOLLOW(A) to FOLLOW(B), update?
                else:
                    add FOLLOW(A) to FOLLOW(B), update?
    }
```

End Function

本程序中没有处理表项冲突（默认就是LL(1)），表项初始化为0表示ERROR，实现算法4.2填入产生式编号。

构造预测分析表

```
Function ll1table()
    initialize LL(1) table with ERROR
    for each production A -> α:
        for each terminal a in FIRST(α):
            if a is not 'ε':
                LL1Table[A, a] = production number
        if 'ε' is in FIRST(α):
```

```
        for each terminal b in FOLLOW(A):
            LL1Table[A, b] = production number
End Function
```

2.2.2 构造分析程序

分析过程主要是三部分，分析栈、输入栈、分析动作，实现算法4.1。用结构体AnalysisStep记录分析栈和输入栈，初始化：$压栈，起始符号E压栈，top为分析栈栈顶指针；expr$放入输入栈，向前指针ip指向第一个符号。然后程序根据分析表ll1table对输入符号串expr作出自顶向下分析。

匹配终结符：检查输入串和栈顶是否一致，匹配成功则前移指针；

产生式推导：从分析表中查找对应规则，逐个将右部符号逆序入栈；

错误检测：若栈顶非终结符和输入符号无匹配规则则报错；

构造预测分析程序

```
Function ll1analyze(char* expr)
    initialize stack with ['$' and E]
    append '$' to the end of expr
    do{
        X = the top of the stack, a = current input symbol
        if X is a terminal or '$':
            if X==a:
                Pop stack
                Move ip forward
            else:
                Error
```

```
        else:
            if ll1table[X, a]=X->Y1…Yk :
                Pop stack
                Push Yk…Y1
                Output
            else:
                Error
    }while(X!='$')
```

End Function

3 测试报告

输入为一个算术表达式

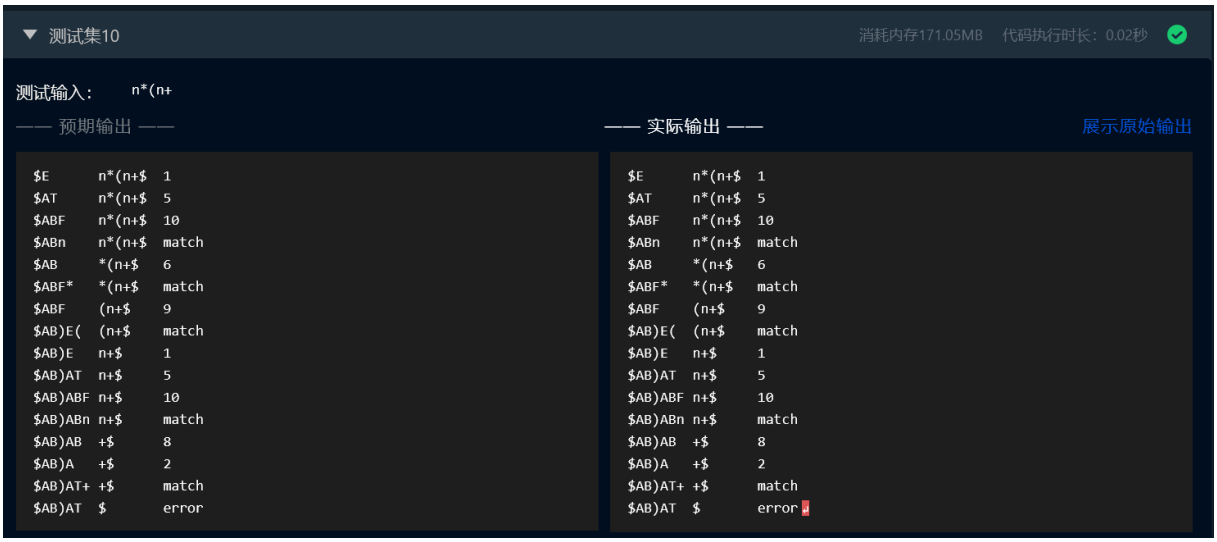
构成该算术表达式的字符有：{'n'， '+'， '-'， '*'， '/'， '('， ')' }

输出

分析栈：以 $ 符号表示栈底的字符串(左侧为栈底；由终结符和非终结符构成)

输入栈：以 $ 符号表示栈底的字符串(右侧为栈底)

分析动作：产生式编号 或 match 或 error 或 accept(表示当前步骤应执行的动作)



头歌实验平台测试用例全部通过。

4 实验总结

在本次实验中，将书中的LL(1)相关算法翻译为代码比较顺利。在实验平台测试有固定的LL(1)文法，无需对文法进行预处理（消除左递归、提取左公因式），也无需在构造分析表时处理冲突，大幅降低了难度。唯一一处卡了半天的地方竟然是被自己一开始LL1Table数组大小设置背刺，因为我当时是想直接不算e那列后来写完忘了……让我深刻体会到，算法的理论推导和代码实现之间的差异往往体现在一些细节上，尤其是边界条件。

LR(1)语法分析程序

1 背景

1.1 实验介绍

《编译原理与技术（第 2 版）》第 4 章的 程序设计 2 (方法 3)

1.2 实验任务

编写一个 LR(1)语法分析程序，能对算数表达式进行语法分析。要求在给定文法：

编号	产生式
0	E’→E
1	E→E+T
2	E→E-T
3	E→T
4	T→T*F
5	T→T/F
6	T→F
7	F→(E)
8	F→num

- 实现构造该文法的 LR(1)分析表；
- 实现算法 4.3，构造 LR(1)分析程序。

2 程序设计说明

2.1 实验原理

大多数无二义性的CFG都可以用LR分析法分析。LR(1)分析方法是一种自底向上的分析技术，是移进-归约分析法。这种分析方法从左到右扫描输入，按最右推导，是规范归约的过程。根据当前分析栈中的符号串和向右顺序查看输入串的1个符号就可以唯一确定分析器的动作。

2.2 程序设计

数据结构和全局变量：

AnalysisStep：分析栈和输入栈；

Item： LR(1)分析中的项目，包括左部、右部、点位置和向前看符号；

State：一个LR(1)状态，包括项目数组和项目数量；

states：一个State数组，记录所有状态；

s_cnt：状态数量；

action：分析表中的一个动作，包括动作类型（移入、规约、接受）和数字（状态或产生式编号）；

ACTION：一个action二维数组；

GOTO：状态转移表，记录每个状态和非终结符的状态转移；

non_terms，terms，productions，FIRST和FOLLOW与LL(1)同理；

核心模块：

first()求解FIRST集；follow()求解FOLLOW集；add_item(State *state, char left, const char *right, int dot, const char *lookahead)向状态中添加项目；closure(State *set)求解闭包扩展状态集；dfa()构造DFA得分析表；fill_tables()构建ACTION和GOTO表；lr1analyze(char* expr) 进行LR(1)预测分析。

2.2.1 构造分析表

LR(1)分析表是分析器的核心，包括action和goto两部分，实现算法4.9。

构造LR(1)分析表的第一步是构造拓广文法。任务给定文法的起始符号E’仅出现在一个产生式（0号产生式）的左边，从而使得分析器只有一个接受状态。因此本程序不做文法的拓广。

构造LR(1)项目集规范族及DFA，结合FIRST集和FOLLOW集列出分析表。

构造分析表

```
Function dfa()
    Initialize states with closure of S -> .E, $
    update=1
    while (update):
        update=0
          For each state:
              For each item in the state:
                 If there is a symbol after the dot:
                      Create a new state with all items that shift over this symbol
                      closure(new state)
                      If new state already exists in states:
                          Add a transition to ACTION or GOTO for the symbol
                      Else:
                          Add the new state to states
                          Update ACTION or GOTO for the symbol
End Function
Function fill_tables()
      Extend grammar to include augmented production S -> .E, $
      Initialize ACTION and GOTO tables as empty
       dfa()
      For each state in DFA:
        a. For each item in the state:
```

i. If it is a reduction item (dot at the end of the production):

If the production is S -> E:

Set ACTION[state][lookahead] = Accept

Else:

For each lookahead symbol:

Set ACTION[state][lookahead] = Reduce with production

End Function

2.2.2 构造分析程序

LR(1)分析程序根据分析表确定分析动作（移入、规约、接受、错误），实现算法4.3。用结构体AnalysisStep记录分析栈和输入栈，初始化：0压栈，top为分析栈栈顶指针；expr$放入输入栈，ip指向第一个符号。然后程序根据ACTION和GOTO表进行自底向上分析。

构造分析程序

Function ll1analyze(char* expr)

initialize stack with ['$' and E]

append '$' to the end of expr

do{

S = the top of the stack, a = current input symbol

If ACTION[S, a] = shift S’:

Push a and S’

Move ip forward

Else if ACTION[S, a] = reduce by A-> β :

Pop | β |*2 elements

```
                Set S’ as top
                Push A and GOTO[S’, A]
                Output
        Else if ACTION[S, a] = accept
                Return
        Else
                Error()
    }while(1)
```

End Function

3 测试报告

输入为一个算术表达式

构成该算术表达式的字符有：{'n'， '+'， '-'， '*'， '/'， '('， ')' }

输出

分析动作：归约使用的产生式编号 或 shift 或 error 或 accept(表示当前步骤应执行的动作)

DEBUG日志

打印分析表，发现有很多state1，完全没有error的情况

ACTION:

state	+	-	*	/	(	)	n	$
0	S1	S1	S1	S1	S1	S1	S1	S1
1	S1	S1	S1	S1	S1	S1	S1	S1
2	S1	S1	S1	S1	S1	S1	S1	S1
3	R8	R8	R8	R8	S1	S1	S1	R8
4	S1	S1	S1	S1	S1	S1	S1	A

打印每个状态和转移，发现是遍历符号时导致的空状态

```
State 0 (from state -1, symbol ''):
  S -> .E, $
  E -> .E+T, $+-
  E -> .E-T, $+-
  E -> .T, $+-
  T -> .T*F, $+-*/
  T -> .T/F, $+-*/
  T -> .F, $+-*/
  F -> .(E), $+-*/
  F -> .n, $+-*/

State 1 (from state 0, symbol '+'):
```

打印填写ACTION表格的每一步发现了根据lookahead填表的覆盖问题，修好后获得了正确的分析表。

头歌实验平台测试用例全部通过。

4 实验总结

在本次实验中，将书中的LR(1)相关的较为复杂的算法翻译为代码过程有些坎坷。实验平台提供的LR(1)文法无需拓广。一开始ACTION的数据结构设的三维字符数组，一开始，我为ACTION数据结构设计了一个三维字符数组，后来发现状态太多了，问题变得复杂，于是将其改为结构体，这让我深刻体会到数据结构设计的重要性。为了方便提交写了一个c文件，导致代码太长了看起来很困难。Debug过程中几乎是打印了整个DFA，逐个填表步骤检查，最终得到正确的ACTION和GOTO。通过手写LR(1)分析器，我对“移进-归约”语法分析有了更深的理解，挺有成就感的，并且从中获得了宝贵的经验。