LR 语法分析器实验报告

学号-20142112280-谢非

2016 年 11 月 22 日

目录

1 实验目的

1. 熟悉 LR 文法的实现原理
2. 理解分析表的具体工作流程

2 实验要求

1. 构造出 LR(0) 项目集规范族及 DFA
2. 构造 SLR(1) 分析表
3. 构造 LR(1) 分析程序

3 实验环境

实验程序用 python 代码实现, 实际程序运行时需要 python3.5 环境以及 python 的 prettytable 包支持

4 实验原理

4.1 DFA 的构造

初始项目集规范族为 $State_0$ (IS_0), 添加 E'->.E, 之后循环拓展（如果点后面是非终结符，则需要把该非终结符的产生式也添加到该项目中. 之后统计该项目小点后面的终结符和非终结符，便于之后由该状态发射边指向下一个状态.

```
class grammerState:
    '''define a state for dfa'''
    def __init__(self,num):
        self.generator = {} # 该状态具有的所有项目
        self.number=num     #  该状态编号
        self.ableTerminal=set()  #  包含所有小点后面的终结符
        self.ableNoterminal=set()  # 包含所有小点后面的非终结符
```

有了第一个状态之后，根据其具有的射出边（终结符和非终结符)，构造出新的状态. 再根据当前状态的项目分别移动小点，把对应项目添加到这些新产生的状态中. 同时把新产生的状态添加到一个集合中，下一次根据这些新状态来射出新的边，产生新的状态. 最终直到没有新的状态产生则构造结束. 不过，这个过程中产生的新状态中可能有重复的状态，所以在扩展时要注意不要添加这些已经有的状态. 也因为这个原因，最终生成的状态会有重复，再去重后，状态的编号可能并不是一个顺序序列.（只是序号不一样，实际对应的 DFA 还是一样的.

```
                #根据新状态的小点后面的终结符和非终结符构造新状态
                for m in extenS.ableTerminal:
                    newState = self.createState() # 构造新状态时会自动产生一个编
号
                    #dfa中,extenS状态经m指向newState(dfa中用number代指状态)
                    self.dfa[extenS.number][m] = newState.number
                    newExtenState.append(newState)
                for i in extenS.ableNoterminal:
                    newState = self.createState()
                    self.dfa[extenS.number][i] = newState.number
                    newExtenState.append(newState)
```

4.2 LR 分析表的构造

LR 分析表的构造实际上在构造 DFA 时就开始构造了，由于构造新状态的过程中实际上就可以知道当前状态可以遇到哪些符号. 所以在产生了新状态的同时，可以往分析表里面添加具体操作.

```
#遍历当前状态的所有项目
for noter in allGeneor:
    for genor in allGeneor[noter]:
        pos = genor.find('.') #查找小点的位置
        tempSymbol = genor[pos + 1:pos + 2]  # 取小点后面的一个字符
```

根据字符是终结符，非终结符或空来往分析表里添加信息

如果小点后面是终结符则移进，是非终结符则直接跳转到某个状态，是空则添加规约动作.

4.3 分析程序构造

分析表构造出来之后，分析程序很容易就可以构造出来. 初始时栈内为 0 状态，之后根据栈顶的状态和输入串的第一个字符在分析表中查找动作. 再根据动作的类型对栈进行相应操作. 同时记录动作，最后输出.

```
            # 根据栈顶状态topState和输入的第一个字符firstTarget获取动作
            action=self.analyzeTable[topState][firstTarget]
        # 根据动作的类型分别进行相应操作
        if action[0]=='s':
            stack.append(tempTarget)  #移进输入字符
            stack.append(int(action[1:]))  # 移进状态
            inputS=inputS[1:]
        elif action[0]=='r':
            # stack.pop()
            num=int(action[1:])
            genor=self.geneList[num]
            genorFormat=genor.split("->")
            removeLen=len(genorFormat[1])*2
                #弹出栈内的规约产生式
            stack=stack[0:len(stack)-removeLen]

                # 获取栈顶状态
            lastState=stack[len(stack)-1]
            # judge next state
                # 根据栈顶状态和规约式左端获取下个状态
            nextState=self.analyzeTable[lastState][genorFormat[0]]
                # 规约式左端和新状态入栈
            stack.append(genorFormat[0])
            stack.append(nextState)
```

直到 acc 时或报错程序停止，分析结束.

5 错误处理

输入不合法的串时会报错，如下图所示:

```
Please input your analyze string:(like 4*6+3 , input 'end' can quit): (4*5
error for this string

Please input your analyze string:(like 4*6+3 , input 'end' can quit): (5*4)-3/2-
error for this string

Please input your analyze string:(like 4*6+3 , input 'end' can quit): 9-5*-3/
error for this string
```

6 实验样例

实验可以打印出优美的表格 [1], 程序主体自带几个测试样例，输出见附件. [2]下面附上几张截图:

```
Analyze Action Table for n+n*n is  here:
+-------------------------------------------------+--------+------------------+
|                      stack                      | input  |  analyze action  |
+-------------------------------------------------+--------+------------------+
|                       [0]                       | n+n*n$ |     Shift 2      |
|                  [0, 'n', 2]                    | +n*n$  |  Reduce by F->n  |
|                  [0, 'F', 4]                    | +n*n$  |  Reduce by T->F  |
|                  [0, 'T', 5]                    | +n*n$  |  Reduce by E->T  |
|                  [0, 'E', 3]                    | +n*n$  |     Shift 12     |
|              [0, 'E', 3, '+', 12]               |  n*n$  |     Shift 2      |
|          [0, 'E', 3, '+', 12, 'n', 2]           |  *n$   |  Reduce by F->n  |
|          [0, 'E', 3, '+', 12, 'F', 4]           |  *n$   |  Reduce by T->F  |
|          [0, 'E', 3, '+', 12, 'T', 25]          |  *n$   |     Shift 14     |
|      [0, 'E', 3, '+', 12, 'T', 25, '*', 14]     |   n$   |     Shift 2      |
|  [0, 'E', 3, '+', 12, 'T', 25, '*', 14, 'n', 2] |   $    |  Reduce by F->n  |
| [0, 'E', 3, '+', 12, 'T', 25, '*', 14, 'F', 31] |   $    | Reduce by T->T*F |
|          [0, 'E', 3, '+', 12, 'T', 25]          |   $    | Reduce by E->E+T |
|                  [0, 'E', 3]                    |   $    |       acc        |
+-------------------------------------------------+--------+------------------+
```

```
Analyze Action Table for 6-3*(2+7) is  here:
+-----------------------------------------------------------------------------+-----------+------------------+
|                                    stack                                    |   input   |  analyze action  |
+-----------------------------------------------------------------------------+-----------+------------------+
|                                     [0]                                     | 6-3*(2+7)$ |     Shift 1      |
|                                [0, '6', 1]                                  | -3*(2+7)$  |  Reduce by F->n  |
|                                [0, 'F', 5]                                  | -3*(2+7)$  |  Reduce by T->F  |
|                                [0, 'T', 4]                                  | -3*(2+7)$  |  Reduce by E->T  |
|                                [0, 'E', 3]                                  | -3*(2+7)$  |     Shift 11     |
|                           [0, 'E', 3, '-', 11]                              |  3*(2+7)$  |     Shift 1      |
|                       [0, 'E', 3, '-', 11, '3', 1]                          |  *(2+7)$   |  Reduce by F->n  |
|                       [0, 'E', 3, '-', 11, 'F', 5]                          |  *(2+7)$   |  Reduce by T->F  |
|                    [0, 'E', 3, '-', 11, 'T', 20]                            |  *(2+7)$   |     Shift 14     |
|                [0, 'E', 3, '-', 11, 'T', 20, '*', 14]                       |   (2+7)$   |     Shift 2      |
|             [0, 'E', 3, '-', 11, 'T', 20, '*', 14, '(', 2]                  |   2+7)$    |     Shift 1      |
|          [0, 'E', 3, '-', 11, 'T', 20, '*', 14, '(', 2, '2', 1]            |    +7)$    |  Reduce by F->n  |
|          [0, 'E', 3, '-', 11, 'T', 20, '*', 14, '(', 2, 'F', 5]            |    +7)$    |  Reduce by T->F  |
|          [0, 'E', 3, '-', 11, 'T', 20, '*', 14, '(', 2, 'T', 4]            |    +7)$    |  Reduce by E->T  |
|          [0, 'E', 3, '-', 11, 'T', 20, '*', 14, '(', 2, 'E', 8]            |    +7)$    |     Shift 12     |
|     [0, 'E', 3, '-', 11, 'T', 20, '*', 14, '(', 2, 'E', 8, '+', 12]        |    7)$     |     Shift 1      |
| [0, 'E', 3, '-', 11, 'T', 20, '*', 14, '(', 2, 'E', 8, '+', 12, '7', 1]    |     )$     |  Reduce by F->n  |
| [0, 'E', 3, '-', 11, 'T', 20, '*', 14, '(', 2, 'E', 8, '+', 12, 'F', 5]    |     )$     |  Reduce by T->F  |
| [0, 'E', 3, '-', 11, 'T', 20, '*', 14, '(', 2, 'E', 8, '+', 12, 'T', 24]   |     )$     | Reduce by E->E+T |
|          [0, 'E', 3, '-', 11, 'T', 20, '*', 14, '(', 2, 'E', 8]            |     )$     |     Shift 17     |
|     [0, 'E', 3, '-', 11, 'T', 20, '*', 14, '(', 2, 'E', 8, ')', 17]        |     $      | Reduce by F->(E) |
|             [0, 'E', 3, '-', 11, 'T', 20, '*', 14, 'F', 31]                |     $      | Reduce by T->T*F |
|                    [0, 'E', 3, '-', 11, 'T', 20]                            |     $      | Reduce by E->E-T |
|                                [0, 'E', 3]                                  |     $      |       acc        |
+-----------------------------------------------------------------------------+-----------+------------------+
```

[1]依赖 python 库 prettytable

[2]样例输出.txt

Analyze Action Table for 8*(5/3+2) is here:

stack	input	analyze action
[0]	8*(5/3+2)$	Shift 1
[0, '8', 1]	*(5/3+2)$	Reduce by F->n
[0, 'F', 5]	*(5/3+2)$	Reduce by T->F
[0, 'T', 4]	*(5/3+2)$	Shift 14
[0, 'T', 4, '*', 14]	(5/3+2)$	Shift 2
[0, 'T', 4, '*', 14, '(', 2]	5/3+2)$	Shift 1
[0, 'T', 4, '*', 14, '(', 2, '5', 1]	/3+2)$	Reduce by F->n
[0, 'T', 4, '*', 14, '(', 2, 'F', 5]	/3+2)$	Reduce by T->F
[0, 'T', 4, '*', 14, '(', 2, 'T', 4]	/3+2)$	Shift 13
[0, 'T', 4, '*', 14, '(', 2, 'T', 4, '/', 13]	3+2)$	Shift 1
[0, 'T', 4, '*', 14, '(', 2, 'T', 4, '/', 13, '3', 1]	+2)$	Reduce by F->n
[0, 'T', 4, '*', 14, '(', 2, 'T', 4, '/', 13, 'F', 28]	+2)$	Reduce by T->T/F
[0, 'T', 4, '*', 14, '(', 2, 'T', 4]	+2)$	Reduce by E->T
[0, 'T', 4, '*', 14, '(', 2, 'E', 8]	+2)$	Shift 12
[0, 'T', 4, '*', 14, '(', 2, 'E', 8, '+', 12]	2)$	Shift 1
[0, 'T', 4, '*', 14, '(', 2, 'E', 8, '+', 12, '2', 1]	)$	Reduce by F->n
[0, 'T', 4, '*', 14, '(', 2, 'E', 8, '+', 12, 'F', 5]	)$	Reduce by T->F
[0, 'T', 4, '*', 14, '(', 2, 'E', 8, '+', 12, 'T', 24]	)$	Reduce by E->E+T
[0, 'T', 4, '*', 14, '(', 2, 'E', 8]	)$	Shift 17
[0, 'T', 4, '*', 14, '(', 2, 'E', 8, ')', 17]	$	Reduce by F->(E)
[0, 'T', 4, '*', 14, 'F', 31]	$	Reduce by T->T*F
[0, 'T', 4]	$	Reduce by E->T
[0, 'E', 3]	$	acc

7 实验总结

本次实验过程中，我对分析表的构成和分析程序的工作过程有了更深的体会，我对 LR 文法的分析过程有了更加深入的认识和理解. 同时感觉到分析表具体化了各种操作之后，程序很容易实现出来，分析表的与程序的贴合度很高.