

编译原理实验三

——语义分析器——

19335253 葉琚明

目录

编译原理实验三

——语义分析器——

一 实验目的

二 实验要求

三 实验设计

3.1 算术表达式LL(1)文法

3.1.1 LL(1)文法设计

3.1.2 LL(1)程序

四 实验代码

4.1 LL(1)翻译法实现算术表达式四元式翻译

五 实验结果

5.1 LL(1)翻译法实验结果

六 实验心得

附录A

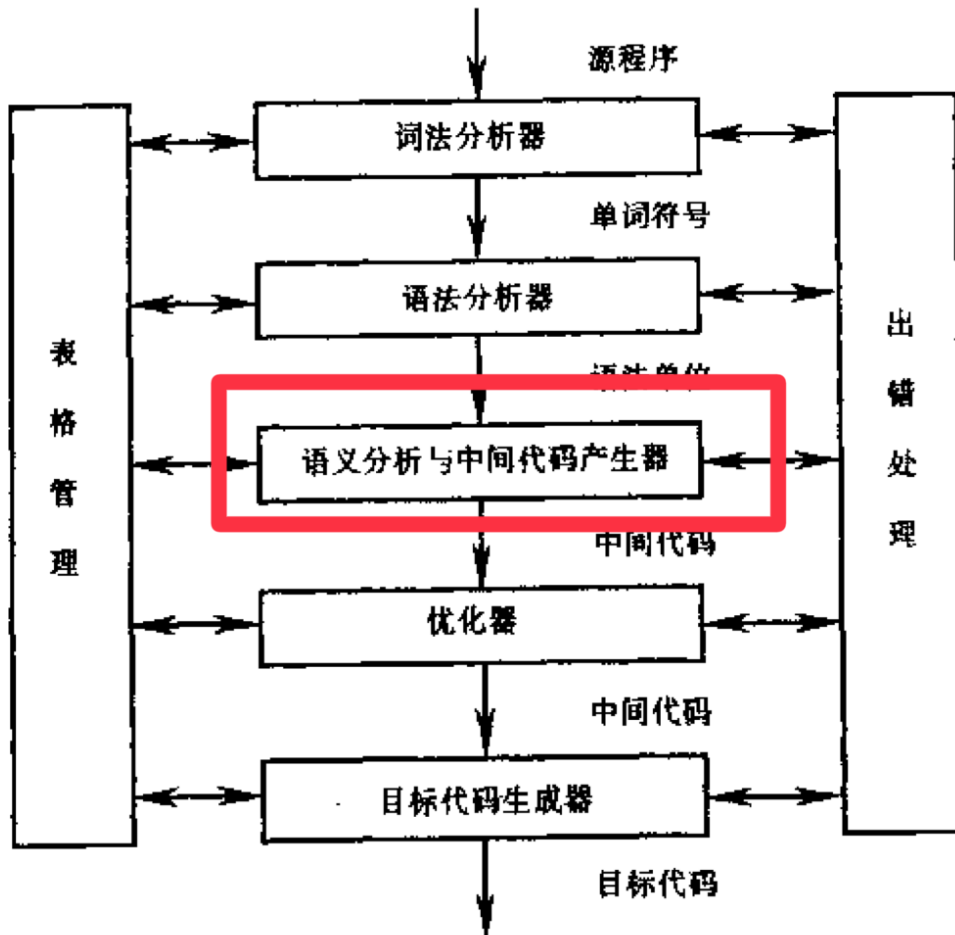
A.1 LL(1)翻译文法实现代码：

所有文件源码地址：[click here](#)

一 实验目的

- 设计至少支持加减乘除以及括号操作的算术表达式语义分析器
- 实现语义分析器功能：将算术表达式翻译成四元式的中间语言
- 使用LL(1)翻译法实现

编译程序的总体处理过程：红色框选部分为本次实验要求实现的部分。



二 实验要求

- 用LL(1)翻译法设计实现算术表达式的语分析器

三 实验设计

3.1 算术表达式LL(1)文法

3.1.1 LL(1)文法设计

- 算术表达式的文法：

$$\begin{aligned}E &\rightarrow T|E + T|E - T \\T &\rightarrow F|T * F|T / F \\F &\rightarrow id|(E)\end{aligned}$$

其中 id 代表数字或字母

- 消除上述文法的左递归

$$\begin{aligned}E &\rightarrow T\{GEQ(+)\}E' & \textcircled{1} \\E' &\rightarrow +T\{GEQ(-)\}E' & \textcircled{2} \\E' &\rightarrow -TE' & \textcircled{3} \\E' &\rightarrow \epsilon & \textcircled{4} \\T &\rightarrow FT' & \textcircled{5} \\T' &\rightarrow *F\{GEQ(*)\}T' & \textcircled{6} \\T' &\rightarrow /F\{GEQ(/)\}T' & \textcircled{7} \\T' &\rightarrow \epsilon & \textcircled{8} \\F &\rightarrow id\{PUSH(i)\} & \textcircled{9} \\F &\rightarrow (E) & \textcircled{10}\end{aligned}$$

其中 id 代表数字或字母

- 构造LL(1)的预测分析表

	id	+	-	*	/	()	#
E	①					①		
E'		②	③				④	④
T	⑤					⑤		
T'		⑧	⑧	⑥	⑦		⑧	⑧
F	⑨					⑩		

3.1.2 LL(1)程序

- 语法栈SYN，语义栈SEM，四元式区QT
- 具体实现：
 - 当产生式(逆序)压栈时，动作符号也不例外；
 - 当动作符号位于栈顶时，执行之；
- 伪代码：

```

1  if syn.peek() == 'geq':
2      syn.pop()
3      s1 = sem.peek()
4      sem.pop()
5      s2 = sem.peek()
6      sem.pop()
7      s3 = 't'+str(j)
8      sem.push(s3)
9      j = j+1
10     s4 = '(' + '+' + ',' + s2 + ',' + s1 + ',' + s3 + ')'
11     qt.push(s4)
12     continue
13     ...
14
15 if oper == 1:
16     status.push(1)
17     syn.pop()
18     syn.push('E1')
19     syn.push('T')
20 elif oper == 2:
21     status.push(2)
22     syn.pop()
23     syn.push('E1')
24     syn.push('geq+')
25     syn.push('T')
26     syn.push('+')
27     ...

```

四 实验代码

4.1 LL(1)翻译法实现算术表达式四元式翻译

- 部分核心代码:

```

1  for line in exps:
2      syn = Stack()
3      syn.push('E')
4      status = Stack()
5      status.push(0)
6      sem = Stack()
7      qt = Stack()
8
9      while syn.size() != 0:
10         token = line[i]
11         mark = token
12         regular = r'\d+\.\d+|\d+|\w+(\w|\d)*'
13         t = re.fullmatch(regular, token)
14         if t != None:
15             mark = 'id'
16
17         if syn.peek() == token:
18             syn.pop()
19             i = i+1

```

```

20         continue
21
22     if syn.peek() == 'geq+':
23         syn.pop()
24         s1 = sem.peek()
25         sem.pop()
26         s2 = sem.peek()
27         sem.pop()
28         s3 = 't'+str(j)
29         sem.push(s3)
30         j = j+1
31         s4 = '(' + '+' + ',' + s2 + ',' + s1 + ',' + s3 + ')'
32         qt.push(s4)
33         continue
34     ...
35
36     if row.get(syn.peek()) == None:
37         print("INCORRECR EXPRESSION!!")
38         break
39     oper = table[row[syn.peek()]][col[mark]]
40     if oper == 1:
41         status.push(1)
42         syn.pop()
43         syn.push('E1')
44         syn.push('T')
45     elif oper == 2:
46         status.push(2)
47         syn.pop()
48         syn.push('E1')
49         syn.push('geq+')
50         syn.push('T')
51         syn.push('+')
52     ...

```

五 实验结果

5.1 LL(1)翻译法实验结果

测试文件为`tokens.txt`，内容是算术表达式的`tokens`。

读取文件`tokens.txt`，语义分析器直接输出分析结果：

- 翻译算术表达式： $98.0 + 32.44 - 0.4\#$ ：

['98.0', '+', '32.44', '-', '0.4', '#']			
SYN			
['E']	TOKEN	SEM	STATUS
['E1', 'T']	98.0	['98.0']	0
['E1', 'T1', 'F']	98.0	['98.0']	1
['E1', 'T1', 'push', '98.0']	98.0	['98.0']	5
['E1', 'T1', 'push']	98.0	['98.0']	9
['E1', 'T1']	+	['98.0']	9
['E1']	+	['98.0']	9
['E1', 'geq+', 'T', '+']	+	['98.0']	8
['E1', 'geq+', 'T']	32.44	['98.0']	2
['E1', 'geq+', 'T1', 'F']	32.44	['98.0']	5
['E1', 'geq+', 'T1', 'push', '32.44']	32.44	['98.0']	9
['E1', 'geq+', 'T1', 'push']	-	['98.0']	9
['E1', 'geq+', 'T1']	-	['98.0', '32.44']	9
['E1', 'geq+']	-	['98.0', '32.44']	8
['E1']	-	['t0']	8
['E1', 'geq-', 'T', '-']	-	['t0']	3
['E1', 'geq-', 'T']	0.4	['t0']	3
['E1', 'geq-', 'T1', 'F']	0.4	['t0']	5
['E1', 'geq-', 'T1', 'push', '0.4']	0.4	['t0']	9
['E1', 'geq-', 'T1', 'push']	#	['t0']	9
['E1', 'geq-', 'T1']	#	['t0', '0.4']	9
['E1', 'geq-']	#	['t0', '0.4']	8
['E1']	#	['t1']	8
CORRECT EXPRESSION!!			
The result is: ['(+,98.0,32.44,t0)', '(-,t0,0.4,t1)']			

- 分析：第一行代表需要分析的`tokens`，SYN列为语法栈，可以看到翻译过程中，程序能够根据LL(1)分析表进行状态的转化（文法中表达式的选择），这一点也可以从最后一列的状态看出，其最终结果也正确输出；

- 翻译算术表达式：123/((a * b) - 5#

['123', '/', '(', '(', 'a', '*', 'b', ')', '-', '5', '#']			
SYN	TOKEN	SEM	STATUS
['E1']	123	['123']	0
['E1', 'T1']	123	['123']	1
['E1', 'T1', 'F']	123	['123']	5
['E1', 'T1', 'push', '123']	123	['123']	9
['E1', 'T1', 'push']	/	['123']	9
['E1', 'T1']	/	['123']	9
['E1', 'T1', 'geq', 'F', '/']	/	['123']	7
['E1', 'T1', 'geq', 'F']	(['123']	7
['E1', 'T1', 'geq', ')', 'E', '(']	(['123']	10
['E1', 'T1', 'geq', ')', 'E']	(['123']	10
['E1', 'T1', 'geq', ')', 'E1', 'T']	(['123']	1
['E1', 'T1', 'geq', ')', 'E1', 'T1', 'F']	(['123']	5
['E1', 'T1', 'geq', ')', 'E1', 'T1', ')', 'E', '(']	(['123']	10
['E1', 'T1', 'geq', ')', 'E1', 'T1', ')', 'E1', 'T']	a	['123']	10
['E1', 'T1', 'geq', ')', 'E1', 'T1', ')', 'E1', 'T1', 'F']	a	['123']	1
['E1', 'T1', 'geq', ')', 'E1', 'T1', ')', 'E1', 'T1', 'push', 'a']	a	['123']	5
['E1', 'T1', 'geq', ')', 'E1', 'T1', ')', 'E1', 'T1', 'push']	a	['123']	9
['E1', 'T1', 'geq', ')', 'E1', 'T1', ')', 'E1', 'T1', 'push']	*	['123']	9
['E1', 'T1', 'geq', ')', 'E1', 'T1', ')', 'E1', 'T1', 'push']	*	['123', 'a']	9
['E1', 'T1', 'geq', ')', 'E1', 'T1', ')', 'E1', 'T1', 'geq', 'F', '*']	*	['123', 'a']	6
['E1', 'T1', 'geq', ')', 'E1', 'T1', ')', 'E1', 'T1', 'geq', 'F']	b	['123', 'a']	6
['E1', 'T1', 'geq', ')', 'E1', 'T1', ')', 'E1', 'T1', 'geq', 'push', 'b']	b	['123', 'a']	9
['E1', 'T1', 'geq', ')', 'E1', 'T1', ')', 'E1', 'T1', 'geq', 'push'])	['123', 'a']	9
['E1', 'T1', 'geq', ')', 'E1', 'T1', ')', 'E1', 'T1', 'geq'])	['123', 'a', 'b']	9
['E1', 'T1', 'geq', ')', 'E1', 'T1', ')', 'E1', 'T1'])	['123', 't0']	9
['E1', 'T1', 'geq', ')', 'E1', 'T1', ')', 'E1'])	['123', 't0']	8
['E1', 'T1', 'geq', ')', 'E1', 'T1', ')'])	['123', 't0']	4
['E1', 'T1', 'geq', ')', 'E1', 'T1']	-	['123', 't0']	4
['E1', 'T1', 'geq', ')', 'E1']	-	['123', 't0']	8
['E1', 'T1', 'geq', ')', 'E1', 'geq', 'T', '-']	-	['123', 't0']	3
['E1', 'T1', 'geq', ')', 'E1', 'geq', 'T']	5	['123', 't0']	3
['E1', 'T1', 'geq', ')', 'E1', 'geq', 'T1', 'F']	5	['123', 't0']	5
['E1', 'T1', 'geq', ')', 'E1', 'geq', 'T1', 'push', '5']	5	['123', 't0']	9
['E1', 'T1', 'geq', ')', 'E1', 'geq', 'T1', 'push']	#	['123', 't0']	9
['E1', 'T1', 'geq', ')', 'E1', 'geq', 'T1']	#	['123', 't0', '5']	9
['E1', 'T1', 'geq', ')', 'E1', 'geq']	#	['123', 't0', '5']	8
['E1', 'T1', 'geq', ')', 'E1']	#	['123', 't1']	8
['E1', 'T1', 'geq', ')']	#	['123', 't1']	4
INCORRECR EXPRESSION!!			
The result is: ['(*,a,b,t0)', '(-,t0,5,t1)']			

- 分析：该算术表达式式不正确的，其圆括号没有匹配上，程序最终也输出“INCORRECT EXPRESSION”提示该表达式是不正确的，前面 $(a * b)$ ，与 $(a * b) - 5$ 的部分是正确输出其四元式的表达，该语义分析在翻译遇到错误时能够正确判断并退出。

六 实验心得

本次实验中，主要是实现语义分析器的功能，在此之前需要熟悉语法分析的LL(1)和逆波兰式，结合之前所学的语法分析器和词法分析器，根据所得token进行分析。

本次主要实现LL(1)分析表的逻辑，难度不高。通过设置语义栈和语法栈，token来确定当前状态和下一步的状态转化，对于PUSH和GEQ的部分则需要做出相应的操作，PUSH需要把id存入语法栈中，GEQ则需要生成对应的四元式，和语法栈的更新。在实验过程中需要注意的是按照产生式压栈，即逆序分析。

附录A

A.1 LL(1)翻译文法实现代码：

```
1  from email.policy import default
2  import re
3  #      i, +, -, *, /, (, ), #
4  table = [[1, 0, 0, 0, 0, 1, 0, 0],
5           [0, 2, 3, 0, 0, 0, 4, 4],
6           [5, 0, 0, 0, 0, 5, 0, 0],
7           [0, 8, 8, 6, 7, 0, 8, 8],
8           [9, 0, 0, 0, 0, 10, 0, 0],
9           [11, 11, 11, 11, 11, 11, 11, 11]]
10
11
12  row = { 'E' : 0 , 'E1' : 1 , 'T' : 2 , 'T1' : 3, 'F' : 4 , 'push' : 5}
13  col = { 'id' : 0 , '+' : 1 , '-' : 2 , '*' : 3, '/' : 4 ,
14         '(' : 5 , ')' : 6 , '#' : 7 }
15
16
17  exps = []
18
19  class Stack:          # 定义一个栈
20      def __init__(self): # 初始化栈为空列表
21          self.items = []
22
23      def isempty(self):
24          return self.items == []
25
26      def push(self,item):
27          self.items.append(item)
28
29      def pop(self):
30          return self.items.pop()
31
32      def peek(self):
33          return self.items[len(self.items)-1]
34
35      def size(self):
36          return len(self.items)
37
38      def all(self):
39          return self.items
40
41  def get_tokens(filepath):
42      file = open(filepath, 'r')
43      line = file.readlines()
44      exp = []
45      for i in line:
46          i = i.strip()
47          exp.append(i)
48          if i == '#':
49              exps.append(exp)
50              exp = []
51
```



```

52
53 get_tokens("tokens.txt")
54
55
56 for line in exps:
57     print(line)
58
59     syn = Stack()
60     syn.push('E')
61     status = Stack()
62     status.push(0)
63     sem = Stack()
64     qt = Stack()
65     i = 0
66
67     print('%-80s%-10s%-20s%-20s'%( 'SYN', 'TOKEN', 'SEM', 'STATUS'))
68
69     j = 0
70
71     while syn.size() != 0:
72
73
74         token = line[i]
75         print('%-80s%-10s%-20s%-20s'%(syn.all(), token, sem.all(),
status.peek()))
76
77         mark = token
78         regular = r'\d+\.\d+|\d+|\w+(\w|\d)*'
79         t = re.fullmatch(regular, token)
80         if t != None:
81             mark = 'id'
82
83         if syn.peek() == token:
84             syn.pop()
85             i = i+1
86             continue
87
88         if syn.peek() == 'geq+':
89             syn.pop()
90             s1 = sem.peek()
91             sem.pop()
92             s2 = sem.peek()
93             sem.pop()
94             s3 = 't'+str(j)
95             sem.push(s3)
96             j = j+1
97             s4 = '(' + '+' + ',' + s2 + ',' + s1 + ',' + s3 + ')'
98             qt.push(s4)
99             continue
100         elif syn.peek() == 'geq-':
101             syn.pop()
102             s1 = sem.peek()
103             sem.pop()
104             s2 = sem.peek()
105             sem.pop()
106             s3 = 't'+str(j)
107             sem.push(s3)
108             j = j+1

```

```

109         s4 = '(' + '-' + ',' + s2 + ',' + s1 + ',' + s3 + ')'
110         qt.push(s4)
111         continue
112     elif syn.peek() == 'geq*':
113         syn.pop()
114         s1 = sem.peek()
115         sem.pop()
116         s2 = sem.peek()
117         sem.pop()
118         s3 = 't'+str(j)
119         sem.push(s3)
120         j = j+1
121         s4 = '(' + '*' + ',' + s2 + ',' + s1 + ',' + s3 + ')'
122         qt.push(s4)
123         continue
124     elif syn.peek() == 'geq/':
125         syn.pop()
126         s1 = sem.peek()
127         sem.pop()
128         s2 = sem.peek()
129         sem.pop()
130         s3 = 't'+str(j)
131         sem.push(s3)
132         j = j+1
133         s4 = '(' + '/' + ',' + s2 + ',' + s1 + ',' + s3 + ')'
134         qt.push(s4)
135         continue
136
137     if row.get(syn.peek()) == None:
138         print("INCORRECR EXPRESSION!!")
139         break
140
141     oper = table[row[syn.peek()]] [col[mark]]
142
143     if oper == 1:
144         status.push(1)
145         syn.pop()
146         syn.push('E1')
147         syn.push('T')
148     elif oper == 2:
149         status.push(2)
150         syn.pop()
151         syn.push('E1')
152         syn.push('geq+')
153         syn.push('T')
154         syn.push('+')
155     elif oper == 3:
156         status.push(3)
157         syn.pop()
158         syn.push('E1')
159         syn.push('geq-')
160         syn.push('T')
161         syn.push('-')
162     elif oper == 4:
163         status.push(4)
164         syn.pop()
165     elif oper == 5:
166         status.push(5)

```

```
167         syn.pop()
168         syn.push('T1')
169         syn.push('F')
170     elif oper == 6:
171         status.push(6)
172         syn.pop()
173         syn.push('T1')
174         syn.push('geq*')
175         syn.push('F')
176         syn.push('*')
177     elif oper == 7:
178         status.push(7)
179         syn.pop()
180         syn.push('T1')
181         syn.push('geq/')
182         syn.push('F')
183         syn.push('/')
184     elif oper == 8:
185         status.push(8)
186         syn.pop()
187     elif oper == 9:
188         status.push(9)
189         syn.pop()
190         syn.push('push')
191         syn.push(token)
192     elif oper == 10:
193         status.push(10)
194         syn.pop()
195         syn.push(')')
196         syn.push('E')
197         syn.push('(')
198     elif oper == 11:
199         syn.pop()
200         sem.push(line[i-1])
201     elif oper == 0:
202         print("ERROR!")
203         break
204 if syn.size() == 0:
205     print("CORRECT EXPRESSION!!")
206 print("The result is: " , qt.all())
207 print('')
208
209
```