

Compilers Principles Lab

Lab 2 Yacc

71118415 叶宏庭

1. Motivation

该实验的目的是为了自行编写一个语法分析器 Yacc，可以针对输入的字符流进行语法分析，返回结果的步骤分析表。

2. Content description

- 1) Input
 - Stream of characters
 - CFG(Combination of CFGs of some classes of sentence)
- 2) Output
 - Sequence of derivations if top-down syntax analyzing methods are used.
 - Sequence of reductions if bottom-up syntax analyzing methods are used.
- 3) Classes of sentences are defined by yourself
- 4) Error handling may be included

3. Ideas/Methods

- 1) 定义文法产生式
- 2) 由文法计算 First(), Follow()
- 3) 构造预测分析表 LL(1)
- 4) 基于 LL(1) PPT 进行编码
- 5) 执行，输入语句，输出分析结果

4. Assumptions

- 1) 本分析器指定文法为：

```
文法：
E -> E + T | T
T -> T * F | F
F -> ( E ) | i
```

2) 经过消除左递归，文法为：

```
消除左递归：
E->TH      (H代替E')
```

$$H \rightarrow +TH \mid e \quad (e \text{ 替代空})$$

$$T \rightarrow FY \quad (Y \text{ 代替 } T')$$

$$Y \rightarrow *FY \mid e$$

$$F \rightarrow (E) \mid i$$

3) 相关符号定义：

```
非终结符：
E, H, T, Y, F
终结符：
i, +, *, (, ), #
```

5. PPT of the Grammer

	i	+	*	()	#
E	E->TH			E->TH		
H		H->+TH			H->e	H->e
T	T->FY			T->FY		
Y		Y->e	Y->*FY		Y->e	Y->e
F	F->i			F->(E)		

其中 H 表示 E'，e 表示空。

6. Description of important Data Structures

1) dists: 预测分析表

构造预测分析表

```
dists = {
    ('E', 'i'): 'TH', ('E', '('): 'TH', ('H', '+'): '+TH',
    ('H', ')'): 'e', ('H', '#'): 'e', ('T', 'i'): 'FY',
    ('T', '('): 'FY', ('Y', '+'): 'e', ('Y', '*'): '*FY',
    ('Y', ')'): 'e', ('Y', '#'): 'e', ('F', 'i'): 'i',
    ('F', '('): '(E)',
}
```

2) V_t , V_h 分别表示终结符集合, 非终结符集合

构造终结符集合

```
Vt = ('i', '+', '*', '(', ')')
```

构造非终结符集合

```
Vh = ('E', 'H', 'T', 'Y', 'F')
```

7. Description of core Algorithms

总控模块:

```
60 # 总控程序
61 def masterctrl(str):
62     """
63     总控程序, 用于进文法的判断
64     """
65     # 用列表模拟栈
66     stack = []
67     location = 0
68     # 将#号入栈
69     stack.append(str[location])
70
71     # 将文法开始符入栈
72     stack.append('E')
73     # 将输入串第一个字符读进a中
74     location += 1
75     a = str[location]
76
77     flag = True
78     count = 1 # 计算步骤
79     table.add_row([count, printstack(stack), a, printstr(str, location), ''])
80     while flag:
81         if count == 1:
82             pass
83         else:
84             if x in Vt:
85                 table.add_row([count, printstack(stack), a, printstr(str, location), ''])
86             else:
87                 temp = x + '->' + s
88                 table.add_row([count, printstack(stack), a, printstr(str, location), temp])
89             x = stack.pop()
90             if x in Vt: # 栈顶是终结符
91                 if x == str[location]: # 该字符匹配, 输入串向后挪一位
92                     location += 1
93                     a = str[location]
94             else: # 否则错误
95                 error()
```

```
96         elif x == '#': # 栈顶是结束符
97             if x == a: # 当前输入字符也是结束符, 分析结束
98                 flag = False
99             else: # 否则错误
100                 error()
101         elif (x, a) in dists.keys(): # M[x,a]是产生式
102             s = dists[(x, a)]
103             for i in range(len(s) - 1, -1, -1): # 倒序入栈
104                 if s[i] != 'e':
105                     stack.append(s[i])
106         else:
107             error()
108         count += 1
```

Erroe 模块:

```

53
54 # 定义error函数
55 def error():
56     print('Error')
57     exit()
58

```

部分函数：

```

37
38 # 获取输入栈中的内容
39 def printstack(stack):
40     rtu = ''
41     for i in stack:
42         rtu += i
43     return rtu
44
45
46 # 得到输入串剩余串
47 def printstr(str, index):
48     rtu = ''
49     for i in range(index, len(str), 1):
50         rtu += str[i]
51     return rtu
52

```

8. Use cases on running

```

(base) E:\学习资料\课件\程序猿\编译原理\Lab\Yacc>python yacc.py
Enter your string to analysis:i+i*i

```

步骤	分析栈	当前输入a	剩余输入串	所用产生式
1	#E	i	i+i*i#	
2	#HT	i	i+i*i#	E->TH
3	#HYF	i	i+i*i#	T->FY
4	#HYi	i	i+i*i#	F->i
5	#HY	+	+i*i#	
6	#H	+	+i*i#	Y->e
7	#HT+	+	+i*i#	H->+TH
8	#HT	i	i*i#	
9	#HYF	i	i*i#	T->FY
10	#HYi	i	i*i#	F->i
11	#HY	*	*i#	
12	#HYF*	*	*i#	Y->*FY
13	#HYF	i	i#	
14	#HYi	i	i#	F->i
15	#HY	#	#	
16	#H	#	#	Y->e
17	#	#	#	H->e

分析成功!

```
(base) E:\学习资料\课件\程序猿\编译原理\Lab\Yacc>python yacc.py
Enter your string to analysis:i+i*i+i+i
```

步骤	分析栈	当前输入a	剩余输入串	所用产生式
1	#E	i	i+i*i+i+i#	
2	#HT	i	i+i*i+i+i#	E->TH
3	#HYF	i	i+i*i+i+i#	T->FY
4	#HYi	i	i+i*i+i+i#	F->i
5	#HY	+	+i*i+i+i#	
6	#H	+	+i*i+i+i#	Y->e
7	#HT+	+	+i*i+i+i#	H->+TH
8	#HT	i	i*i+i+i#	
9	#HYF	i	i*i+i+i#	T->FY
10	#HYi	i	i*i+i+i#	F->i
11	#HY	*	*i*i+i#	
12	#HYF*	*	*i*i+i#	Y->*FY
13	#HYF	i	i+i*i+i#	
14	#HYi	i	i+i*i+i#	F->i
15	#HY	+	+i*i+i#	
16	#H	+	+i*i+i#	Y->e
17	#HT+	+	+i*i+i#	H->+TH
18	#HT	i	i*i+i#	
19	#HYF	i	i*i+i#	T->FY
20	#HYi	i	i*i+i#	F->i
21	#HY	*	*i+i#	
22	#HYF*	*	*i+i#	Y->*FY
23	#HYF	i	i+i#	
24	#HYi	i	i+i#	F->i
25	#HY	+	+i#	
26	#H	+	+i#	Y->e
27	#HT+	+	+i#	H->+TH
28	#HT	i	i#	
29	#HYF	i	i#	T->FY
30	#HYi	i	i#	F->i
31	#HY	#	#	
32	#H	#	#	Y->e
33	#	#	#	H->e

分析成功!

```
(base) E:\学习资料\课件\程序猿\编译原理\Lab\Yacc>
```

9. Problems occurred and related solutions

- 1) 编写的代码没有准确预测优先级, 造成分析错误, 最终通过修改 PPT 的结构, 成功修复这个错误。
- 2) 在定义数据结构时, 没有很好组织结构关系, 可以进行更好的结构优化。

10. Your feelings and comments

通过本次实验,更加深入理解了 Yacc 语法分析器的工作原理,同时完成了自己的一个简易的语法分析器,更好的掌握了相关的理论知识,在未来的学习和工作中都会有很大的帮助,也希望我能够有时间继续去完善这个语法分析器