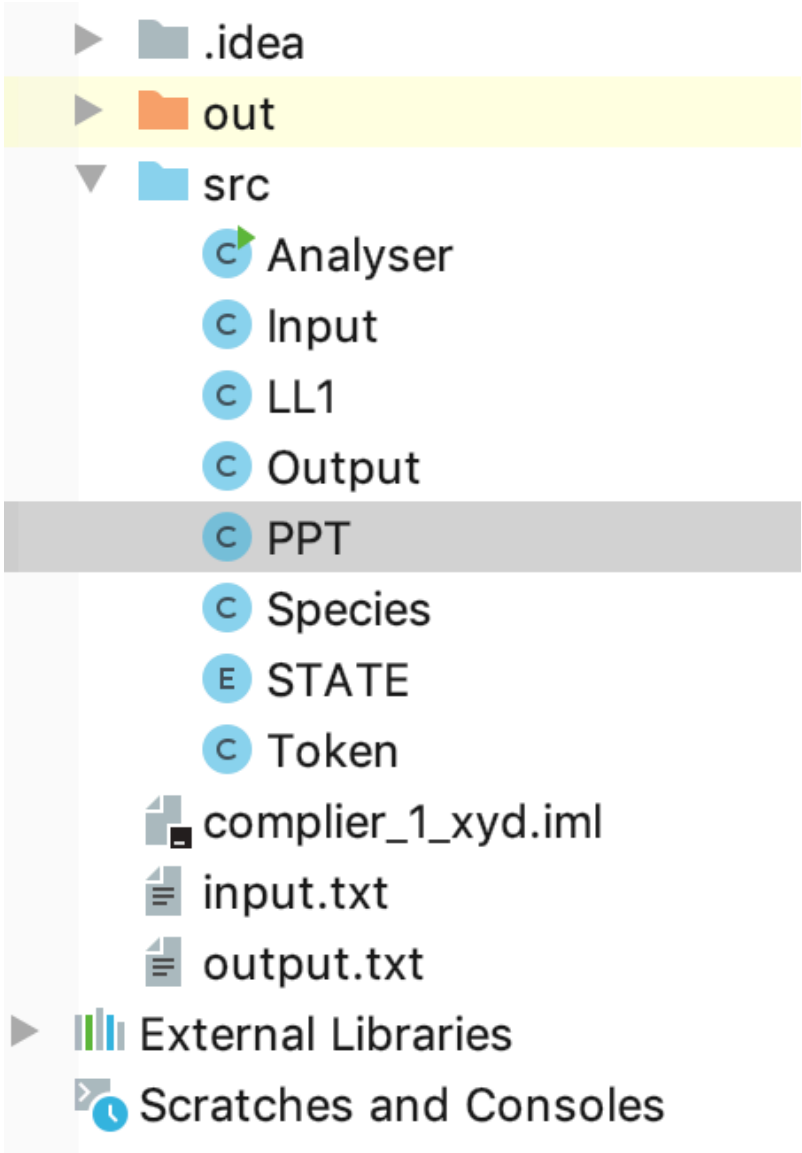
语法分析实验报告

夏雨笛 171250011

类：



输入：

```
while(x==0&&y==1){
if(y==3&&(y==3&&y+2==14)){
x=x+2*y;
}else{
y=y+2*(x+4);
}
}
```

输出：

```
S→while(C){S}S
C→D C'
D→G CMP G
G→id
CMP → ==
G→number
C'→&&DC'
D→G CMP G
G→id
CMP → ==
G→number
C'→ε
S→if(C){S}else{S}S
C→D C'
D→G CMP G
G→id
CMP → ==
G→number
C'→&&DC'
D→(C)
C→D C'
D→G CMP G
G→id
CMP → ==
G→number
C'→&&DC'
D→G CMP G
G→id
```

1. Motivation/Aim

设计了一个 java 语法的分析程序，对部分程序进行语法分析，此时实验采用的是 LL1 分析表的方法。

2. Content description

从 Input.txt 中先读取文件，然后先利用 LAB1 中的代码进行词法分析，然后再利用语法分析程序进行语法分析，输出结果到 Output.txt 中。

3. Ideas

a) 定义出要识别的文法

b) 消除左递归

c) 构造预测分析表。

d) 根据预测分析表编写程序

4. Assumptions

假设输入的文件内容是正常的 JAVA 程序，即包含合法的保留字和运算符。if 语句后都有唯一的 else 语句。id 代表变量，number 代表数字（int , double）。

5. specific methods

要识别的文法：

0 S→id=ES;| if(C){S}else{S}S | while(C){S}S | ε

1 E→E+T | E-T | T

2 T→T*F | T/F | F

3 F→(E) | id | number

4 C→C||D | C&&D | D

5 D→(C)

6 D→G CMP G

7 G→number | id

8 CMP → > | >= | < | <= | == | !=

对于部分文法需要消除左递归：

E→E+T | E-T | T 变成：

E→T E'

E'→+TE' | -TE' | ε

T→T * F | T/F | F 变成：

T→F T'

T'→* FT' | /FT' | ε

C→C||D | C&&D | D 变成：

C→D C'

C'→||DC' | &&DC' | ε

最终得到的 CFG 为：

0. S→id=E;S

1. S→if(C){S}else{S}S

2. S→while(C){S}S

3. S→ε

4. E→T E'

5. E'→+TE'

6. E'→-TE'

7. E'→ε

8. T→F T'

9. T'→* FT'

10. T'→/ FT'

11. T'→ε
12. F→(E)
13. F→id
14. F→number
15. C→D C'
16. C'→||DC'
17. C'→&&DC'
18. C'→ε
19. D→(C)
20. D→G CMP G
21. G→number
22. G→id
23. CMP → >
24. CMP → >=
25. CMP → <
26. CMP → <=
27. CMP → ==
28. CMP → !=

根据这个 CFG 建立预测分析表：

Vn\Vt	id	if	while	+	-
S	0	1	2		
E	4				
E'				5	6
T	8				
T'				11	11
F	13				
C	15				
C'					

D	20				
G	22				
CMP					

Vn\Vt	*	/	;	=	||
S					
E					
E'			7		
T					
T'	9	10	11		
F					
C					16
C'					
D					
G					
CMP					

Vn\Vt	&&	(	)	{	}
S					3
E		4			
E'					
T		8			
T'					
F		12			
C		15			
C'	17		18		
D		19			
G					

CMP					

Vn\Vt	else	==	>	>=	<
S					
E					
E'					
T					
T'					
F					
C					
C'					
D					
G					
CMP		27	23	24	25

Vn\Vt	<=	!=	number	$r
S				3
E			4	
E'				
T			8	
T'				
F			14	
C			15	
C'				
D			20	
G			21	
CMP	26	28		

6. Description of important Data Structures

构建了一个 PPT 的表：

```
public class PPT {
    private static int[][] ppt = new int[][]{
            {0,   1,  2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,  3, -1,-1,-1,-1,-1,-1,-1,-1, 3},//S
            {4,  -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,  4, -1, -1, -1, -1,-1,-1,-1,-1,-1, 4,-1},//E
            {-1, -1, -1,  5,  6, -1, -1,  7, -1, -1, -1, -1, -1, -1, -1, -1, -1,-1,-1,-1,-1,-1,-1,-1},//E'
            {8,  -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,  8, -1, -1, -1, -1,-1,-1,-1,-1,-1, 8,-1},//T
            {-1, -1, -1,  8,  8,  9, 10,  8, -1, -1, -1, -1, -1, -1, -1, -1, -1,-1,-1,-1,-1,-1,-1,-1},//T'
            {13, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 12, -1, -1,-1,-1,-1,-1,-1,14,-1},//F
            {15, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 15, -1, -1, -1, -1,-1,-1,-1,-1,-1,15,-1},//C
            {-1, -1, -1, -1, -1, -1, -1, -1, -1, 16, 17, -1, 18, -1, -1, -1,-1,-1,-1,-1,-1,-1,-1},//C'
            {20, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 19, -1, -1, -1, -1,-1,-1,-1,-1,-1,20,-1},//D
            {22, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,-1,-1,-1,-1,-1,-1,21,-1},//G
            {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,27,23,24,25,26,28,-1,-1}//CMP
    };

    public static int[][] getPPT() {
        return ppt;
    }
}
```

Species 类存的是每个终结符或者非终结符的种别码：

```
public final static int S_START = 0;
public final static int ID = 1;
public final static int IF = 2;
public final static int WHILE = 3;
public final static int S_ADD = 4;
public final static int S_SUB = 5;
public final static int S_MUL = 6;
public final static int S_DIV = 7;
public final static int SEMICOLON = 8;
public final static int EQUAL = 9;
public final static int S_OR = 10;
public final static int S_AND = 11;
public final static int LEFT_K = 12;
public final static int RIGHT_K = 13;
public final static int LEFT_D = 14;
public final static int RIGHT_D = 15;
public final static int ELSE = 16;
public final static int D_EQUAL = 17;
public final static int BIG = 18;
public final static int BIG_E = 19;
public final static int LESS = 20;
public final static int LESS_E = 21;
public final static int N_EQUAL = 22;
public final static int NUMBER = 23;
public final static int S_END = 24;
public static final int S = 100;
public static final int E = 101;
public static final int E1 = 102;
public static final int T = 103;
public static final int T1 = 104;
public static final int F = 105;
public static final int C = 106;
public static final int C1 = 107;
public static final int D = 108;
public static final int G = 109;
public static final int CMP = 110;
```

对于每一个终结符或者非终结符都保存着种别码，和相印字符的内容。

```
public void addSpecies(){
    String s;
    s="Species : {code : "+this.code+" }"+"  {value : "+this.value+" }";
    Output.species.add(new Species(this.code,this.value));
    Output.species_arr.add(s);
}
```

对于 LL1 文法需要有一个栈和一个队列，以及一个产生式的数组：

```
private static Stack<Species> speciesStack = new Stack<>();//栈
private static Queue<Species> speciesQueue = new LinkedList<>();//队列
private static int[][] ppt = PPT.getPPT();
private Output output=new Output();
private static String[] production={
        "S→id=E;",
        "S→if(C){S}else{S}",
        "S→while(C){S}",
        "S→ε",
        "E→T E'",
        "E'→+TE'",
        "E'→-TE'",
        "E'→ε",
        "T→F T'",
        "T'→* FT'",
        "T'→/ FT'",
        "T'→ε",
        "F→(E)",
        "F→id",
        "F→number",
        "C→D C'",
        "C'→||DC'",
        "C'→&&DC'",
        "C'→ε",
        "D→(C)",
        "D→G CMP G",
        "G→number",
        "G→id",
        "CMP → >",
        "CMP → >=",
        "CMP → <",
        "CMP → <=",
        "CMP → ==",
        "CMP → !="
};
```

7. Description of core Algorithms

利用 LL1 分析的特性，如果栈顶是一个非终结符，就根据预测分析表继续推导，直到变成一个终结符且和队列首部的终结符相同，便匹配成功，同时队列指针向后移动一位，并且栈顶的终结符弹出。

8. example

见文件开始。

另外还有一些测试用例和输出：

test:

```
while(x==0&&y==1){
    a==b;
    c=1;
}
```

print:

```
S→while(C){S}S
C→D C'
D→G CMP G
G→id
CMP → ==
G→number
C'→&&DC'
D→G CMP G
G→id
CMP → ==
G→number
C'→ε
S→id=E;S
```

9. problems

一开始的时候我的循环体{}中只要有东西，就不能成功输出，排查了许久发现是两个应该 break 的地方竟然 return 了。导致无法输出。

另外，一开始由于没有考虑清楚自定义文法，导致每条 while 循环或是 if 判断中只能写一条语句，是因为我的文法中缺少 S→SS 这条语句。但是这时我的代码已经基本成型了，而且时间紧迫，如果重新进行化简左递归，重新制表会要花费很多时间。所以我想到了一个最小修改。

修改了这三条文法：

0. S→id=E;S

1. S→if(C){S}else{S}S

2. S→while(C){S}S

在其后面添加了一个 S，并且修改了少量代码之后解决了问题

10. feelings

感觉这次实验细心很重要，应该实现就要把文法和预测符表思考清楚，画正确。否则一旦有小错误，会给代码带来不小的修改。