编译原理实验报告

计 63 肖朝军 2016011302

一、 本阶段工作内容

1、 错误处理

错误处理方法与 readme 中提到的方法相同。对 parse 函数的修改如下：

若 lookahead 不包含在当前非终结符的 beginSet 中，则报错；报错后，开始跳过字符直到字符属于当前非终结符的 beginSet 或者属于 follow 集合；

1) 若 lookahead 属于当前非终结符的 beginSet，则继续按照正常流程分析程序；
2) 若 lookahead 属于 follow 集合，则返回 null；

follow 集合：follow 需要加上当前节点所有父节点的 followSet

2、 添加新的 LL(1)文法

本次实验要求加入的所有文法都是 LL(1)文法，因此需要按照课件的内容，对着给出的文法消除左公因子以及左递归。需要额外提的是，由于原文法中 Expr 表达式中，涉及到了非常多运算符的优先级，因此 Expr 表达式除了需要消除左递归及左公因子外，需要结合运算符优先级、运算符左结合右结合的性质对文法进行改写。

1) 结合优先级改写文法思路

结合优先级改写文法的大致思路是增加非终结符用来表示不同优先级的表达式，以两个不同优先级的运算符 O_1、O_2 为例（假定运算优先级 $O_1 > O_2$，O_1 和 O_2 可以分别表示一个运算符集合）。原始表达式为：

$$Expr \rightarrow Expr\ O1\ Expr \mid Expr\ O2\ Expr \mid Expr3$$

改写为：

$$Expr \rightarrow Expr1$$
$$Expr1 \rightarrow Expr2\ O2\ Expr2 \mid Expr2$$
$$Expr2 \rightarrow Expr3\ O1\ Expr3$$

再对改写后的表达式进行消除左公因子以及左递归。

2) 左结合 & 右结合

左结合与右结合的主要区别表现在识别完表达式之后的语义动作，右结合的运算符，需要从右至左分析表达式，并产生相应的树节点。

二、 Else 分支冲突说明

Else 分支出现在 IfStmt 中，If 语句为(此处忽略条件卫士语句)

$$IfStmt \rightarrow if\ (\ Expr\ )\ Stmt\ ElseClause$$
$$ElseClause \rightarrow else\ Stmt \mid /*\ empty\ */$$

我们来计算 ElseClause 的两个产生式的预测集：

$$PS(ElseClause \rightarrow else\ Stmt) = \{else\}$$
$$且\ else \in PS(ElseClause \rightarrow /*\ empty\ */)$$

这种情况实际上就是当 if 语句嵌套时，会出现 else 语句匹配哪一个 if 语句的问题，即文法存在二义性（形式语言与自动机课程称之为 悬挂 Else 二义性）。工具解决这个矛盾的方法是给两个产生式一个优先级，产生式优先匹配 ElseClause –> else Stmt 产生式，这样就形成了这样一个效果，else 语句总是匹配和 else 最近的 if 语句。

例：对下面这样一段代码

```
if (expr1)
if (expr2)
    stmt1
else
    stmt2
```

这段代码是符合语法规范的伪代码，有两个 if 却只有一个 else，这样在解析程序时，会产生矛盾，else 是应该作为 if(expr1)的分支还是作为 if(expr2)的分支。在规定了匹配产生式的优先级之后，将使 else 语句匹配为 if(expr2)的分支。

三、 数组 comprehension

该表达式与数组常量的定义产生式会产生冲突。

数组常量文法：

$$\text{Constant} \rightarrow [\text{Constant ConstantList}]$$

$$\text{ConstantList} \rightarrow , \text{Constant ConstantList} \mid /* \text{ empty } */$$

数组 comprehension 文法：

$$\text{Expr} \rightarrow [\text{Expr for Identifier in Expr IfBoolExpr}]$$

$$\text{IfBoolExpr} \rightarrow \text{if Expr} \mid /* \text{ empty } */$$

这两个表达式会产生冲突，两个产生式有着左公因子 [且 Expr 和 Constant 也有重合，如果需要将数组 comprehension 改成 LL(1)文法，则需要将其文法改写，需要将 Expr 中 Constant 和非 Constant 分开，会很复杂。

四、 错误处理误报

对于代码

```
class Main {
    static void main(){
        (true){

        }
        else{

        }
    }
}
```

产生了误报，报错结果为：

```
*** Error at (3,15): syntax error
*** Error at (6,9): syntax error
*** Error at (9,5): syntax error
```

可以看到因为缺少了 if 这样一个关键字，在进行程序分析时，将接下来的 else 以及大括号都进行了报错处理。这是因为这样一个程序缺少了，产生式第一个符号，导致在编译时，无法找到该语句对应的产生式，导致了之后一连串的误报。

五、 实验总结

本次实验与 PA1-A 的主要区别是，加入了错误处理，且要求将文法改写成 LL(1)文法。由于整个程序的文法产生式很多，在改写文法时需要额外注意文法的改写方法，需要考虑到现在已经有的产生式，需要对框架已经给出的产生式进行改写，以达到效果。