# 编译原理 AP_1

## ——实验报告

闫力敏*

## 1 概述

本次试验在 PA_1 的基础上，消除了 Case 语句以及循环卫士的左递归，进一步实现了 LL(1) 语法，然后在 PA_1B 的基础上框架下，实现了错误恢复功能，然后增加了一些特殊的错误样例分析。

### 1.1 实验原理

基础原理仍然通过 Lexer 进行词法分析，然后通过已有的预测表实现分析以及错误恢复过程。

### 1.2 实验过程

首先将转化好的 LL(1) 语法写入.spec 文件。

#### 1.2.1 @ # ¥ 最高运算符优先级

```
Oper8:   '@'
         {
             $$.counter = Tree.RE;
             $$.loc = $1.loc;
         }
         |   '#'
         {
             $$.counter = Tree.TR;
             $$.loc = $1.loc;
         }
         |   '$'
         {
             $$.counter = Tree.IM;
```

*清华大学计算机系. 学号：2015011391. 邮编：100084

```
            $$.loc = $1.loc;
        }
    ;
Expr9:    Oper8 Expr9
        {
            $$.expr = new Tree.Unary($1.counter, $2.expr, $1.loc);
        }
        |   Expr10
        {
            $$.expr = $1.expr;
        }
    ;
```

在原有基础上，继续拉高 @ # ¥ 的优先级。

### 1.2.2 Case 语句

```
DefaultExpr
        :   DEFULT ':' Expr ';'
        {
            $$.dfexpr = new Tree.DfExpr($3.expr,$1.loc);
        }
        ;

ACaseExpr
        :   Constant':' Expr';'
        {
            $$.cexpr = new Tree.CaseExpr($1.expr,$3.expr,$1.loc);
        }
        ;

ACaseExprlist
    :  ACaseExpr ACaseExprlist
    {

        $$.celist = new ArrayList<Tree.CaseExpr>();
        $$.celist.add($1.cexpr);
```

```
        $$.celist.addAll($2.celist);
    }
    |
    {
        $$.celist = new ArrayList<Tree.CaseExpr>();
    }
```

消除左递归后，经检验是符合 LL(1) 的，然后接入 Expr，取最高的优先级。

### 1.2.3 循环卫士

```
DoSubStmt
    :   Expr ':' Stmt
    {
        $$.dstmt = new Tree.DoSubStmt($1.expr,$3.stmt,$2.loc);
    }
    ;
DoBranch
    :   DDD DoSubStmt
    {
        $$.dstmt = $2.dstmt;
    }
    ;

DoBranchlist
    :    DoBranch DoBranchlist
    {
        $$.Dolist = new ArrayList<Tree>();
        $$.Dolist.add($1.dstmt);
        $$.Dolist.addAll($2.Dolist);
    }
    |   /* empty */
    {
        $$.Dolist = new ArrayList<Tree>();
    }
    ;
```

```
DoStmt
    :   DDO DoSubStmt DoBranchlist  OOD
    {
        $$.stmt = new Tree.DoStmt($3.Dolist,$2.dstmt,$1.loc);
    }
    ;
```

消除左递归后，经检验是符合 LL(1) 的，然后接入 Expr，取最高的优先级。

### 1.3 if-else-分析

```
IfStmt
    :   IF '(' Expr ')' Stmt ElseClause
    {
        $$.stmt = new Tree.If($3.expr, $5.stmt, $6.stmt, $1.loc);
    }
    ;

ElseClause
    :   ELSE Stmt // higher priority This
    {
        $$.stmt = $2.stmt;
    }
    |   /* empty */
    ;
```

由于 PS(ElseClause -> ELSE Stmt) 与 PS(ElseClause -> /* empty */) 存在交集，即 ELSE，本实验的冲突处理方法是添加了优先级机制，每次遇到 else 后会优先的采用第一个例子如下

```
class Hoo {
    int printNumbers(int n) {

        if (10 < n)
            if (9 < n)
                n = 2;
        else
             n = 3;

     return n;
```

```
    }
}
```

输出结果如下

```
program
    class Hoo <empty>
        func printNumbers inttype
            formals
                vardef n inttype
            stmtblock
                if
                    les
                        intconst 10
                        varref n
                    if
                        les
                            intconst 9
                            varref n
                        assign
                            varref n
                            intconst 2
                    else
                        assign
                            varref n
                            intconst 3
                return
                    varref n
```

在分析第二个 if 的时候，优先匹配 else，所以第一个 if 是没有匹配到 else。

### 1.4 误判分析

错例如下:

```
class Main {
static string dayOfWeek(int n) {
    return case (n) {
        1: "Monday";
        2: "Tuesday";
```

```
            3: "Wednesday";;
            default: "2333"
        };
    }
}
```

输出如下

```
*** Error at (6,19): syntax error
*** Error at (7,4): syntax error
*** Error at (8,3): syntax error
*** Error at (8,4): syntax error
*** Error at (11,1): syntax error
```

因为';'是在父亲的 follow 中，所以遇到多余的';'会跳过然后根据父亲节点来分析，但是后面的语句为 case 所独有因此在用父节点解析的时候会持续报错，从而产生误判。

## 2 总结

本次试验是后面实验的基础，不仅需要认真完成，而且需要借此机会熟悉框架的运行机理，通过此次试验，切实的掌握了 Lex,Yacc 的使用，也复习了不少自动机的知识。