编译原理 PA1-B 实验报告

2017011620 计 73 李家昊

2019 年 10 月 29 日

1 工作内容

1.1 新特性实现

1.1.1 Abstract 和 Var 关键字的实现

新增关键字的实现思路和方法与 PA1-A 完全相同，这里不再赘述。

1.1.2 Lambda 表达式的实现

按照实验说明，Lambda 表达式的优先级最低，此外，为了避免冲突，需要将两种 Lambda 表达式的左公因子提取出来，即在 `Expr` 的产生式中添加 `FUN '(' VarList ')' LambdaExprBody`，然后使 `LambdaExprBody` 产生 Lambda 表达式和 Lambda 语句块，并用一个布尔值标记，使得父表达式可根据标记生成不同的 Lambda 表达式。

对于 Lambda 类型，原产生式为

```
Type ::= AtomType | Type '[' ']' | Type '(' TypeList ')'
AtomType ::= INT | BOOL | STRING | VOID | CLASS Id
```

上式包含左递归，消除左递归后，得到

```
Type ::= AtomType ArrayType
ArrayType ::= '[' ']' ArrayType | '(' TypeList ')' ArrayType | empty
```

利用 `SemValue` 提供的 `thunkList` 作为栈，每次递归时将当前的 `SemValue` 压入栈，返回父产生式时依次出栈，根据每个 `SemValue` 的类型分别处理。

1.1.3 Call 的实现

个人认为这是本实验中最难的一部分。观察 `Decaf.spec` 源码以及 Decaf 优先级表，可知 `Expr8` 对应的优先级为函数调用的优先级。需要特别注意的是，对于产生式 `Expr ::= Expr '(' ExprList ')'`，等号右边的 `Expr` 并不是任意表达

式，而是 Expr8。然后消除左递归，得到 ExprT8 ::= '(' ExprList ')' ExprT8，最后将 ExprListOpt 消除即可。

1.1.4 New 语句对 Lambda 表达式的支持

原来的 New 语句只能生成 AtomType 的数组，并不支持 Lambda 类型数组，例如 var a = new int[](int[])[10];。仿照 Lambda 类型的实现，注意提取左公因子，即可解决此问题。

1.2 错误恢复

对于错误恢复算法，按照实验指导实现即可。具体来说，分析非终结符 A 时，若当前输入符号 $a \notin Begin(A)$，则先调用 yyerror("syntax error"); 报告语法错误，然后循环读取输入符号，直到当前输入符号 $b \in Begin(A) \cup End(A)$，对于下列两种情况分别处理。

- 若 $b \in Begin(A)$ ，则跳出循环，恢复分析。
- 若 $b \in End(A)$，则分析失败，返回 null。

在代码中，$Begin(A)$ 为 LLTable.java 中的 beginSet(symbol) ，$End(A)$ 为 LLTable.java 中的 followSet(symbol) 和参数 follow 的并集。每次递归下降时，需要将 $End(A)$ 作为 follow 参数传入下一层函数继续分析。

2 遇到的困难及解决方案

一开始没有意识到 Expr ::= Expr '(' ExprList ')' 中等号右边的 Expr 是 Expr8，浪费了很多时间，后来通过分析优先级表和代码的关系，得出这个结论，解决了函数调用的问题。

3 PA1-B 相关问题

Q1. 本阶段框架是如何解决空悬 else (dangling-else) 问题的？

A1. 在编译生成的 LLTable.java 中有这样一段代码

```
case ElseClause: {
    switch (lookahead) {
        case ELSE:
            return new AbstractMap.SimpleEntry<>(37, Arrays.asList(
                ELSE, Stmt));
```

```
        ...
    }
}
```

可以看出，当非终结符为 ElseClause 时，若当前输入符号为 else，则直接返回。即相比于空分支，框架赋予 else 分支更高的优先级，因此 else 永远与最近的未匹配的 if 匹配。

Q2. 使用 LL(1) 文法如何描述二元运算符的优先级与结合性？请结合框架中的文法，举例说明。

A2. 可先产生较低优先级的运算符，再生成较高优先级的运算符，同一优先级下，对于左结合的运算符，则将每次解析的 SemValue 依次压入栈，在父表达式中依次出栈，对于右结合的运算符，则将每次解析的 SemValue 依次入队列，在父表达式中依次出队列。例如，对于左结合的较低优先级的运算符 +，-，以及左结合的较高优先级的运算符 *，/，%，LL(1) 文法如下：

```
Op5     ::= '+' | '-'
Op6     ::= '*' | '/' | '%'
Expr5   ::= Expr6 ExprT5
ExprT5  ::= Op5 Expr6 ExprT5 | empty
Expr6   ::= Expr7 ExprT6
ExprT6  ::= Op6 Expr7 ExprT6 | empty
```

其中在解析每一个运算符时，有压栈代码：

```
$$.thunkList.add(0, sv);
```

在函数 buildBinaryExpr 中，将 SemValue 依次出栈，进行相应操作：

```
for (var sv : rest) {
    expr = new Tree.Binary(Tree.BinaryOp.values()[sv.code], expr, sv.
        expr, sv.pos);
}
```

Q3. 无论何种错误恢复方法，都无法完全避免误报的问题。请举出一个具体的 Decaf 程序（显然它要有语法错误），用你实现的错误恢复算法进行语法分析时会带来误报。并说明该算法为什么无法避免这种误报。

A3. 下面代码会产生误报

```
class Main {
    static abc main() {

```

```
    }
}
```

报错信息为

```
*** Error at (2,12): syntax error
*** Error at (2,16): syntax error
```

实际上只有第一行报错信息是有用的（用 PA1-A 的代码测试也只会输出第一行）。根据产生式

```
FieldList ::= STATIC Type Id '(' VarList ')' Block FieldList
```

容易看出，分析到 `abc` 的时候，非终结符为 `Type`，而 `abc` 为 `Id` 属于 $End(Type)$，因此返回 `null`，然后分析非终结符 `Id`，与 `abc` 匹配，接下来分析 `'('` ，却遇到 `main`，于是再一次报错，一直读取到 `main` 后面的 `'('`，属于 $Begin('(')$，继续分析，此后恢复正常，但此算法已经误报了一条错误信息。