PA1-A 实验报告

李潇
计 63
学号: 2016011303

2018 年 10 月 19 日

1 实验总述

本次实验是完成词法分析和语法分析部分，利用语法分析器 Parser 调用词法分析器 Lexer, 完成抽象语法树的构建。

具体地，需要在 lex.l 中识别加入信息识别新的标识符，在 Parser.y 中加入新的语法规则，并对于部分规则，在 Tree.java 加入支持语法规则处理的新节点。下面将分别说明每条新增语法规则的实现思路。

2 具体实现

2.1 对象浅复制语句

此语句支持较为简单，按照要求在 lex.l 中新增关键字 scopy，这样就可以利用词法分析器识别出该 token, 并按照给定的参考语法在 Parser.y 中加入语法规则进行处理。

至于该条语法规则的处理方式，我在 Tree 类中创建了一个继承自 Tree 本身的 Scopy 类。除此之外，我在建立之前创建了新的检测实例类别的标识符，以方便后续的工作。

2.2 禁止类继承

由于 sealed 类和普通类并无本质不同，故我创建了继承自 ClassDef 的 SealedClassDef 类，以复用之前的处理类的代码。

2.3 串行条件卫士语句

这条总体实现较为复杂，具体地，我按照给定的语法规则，创建了 Guarded 和 IfBranch 节点，分别处理 GuardedStmt ::= if <IfBranch* IfSubStmt> 和该语法规则内部的 * 展

开。

需要注意的是这里有一个递归结构，对应于文法中的一个递归产生式，这里应该使用左递归，以保证及时从栈顶取出需要的结构归约掉。

2.4 简单自动类型推导

这里 var x 是整体作为左值存在的，故我创建了继承自 LValue 的 Var 类。

2.5 数字相关操作

对于数组常量，其也应该算作一种常量，故我修改了 Literal 类，在其中加入了 ARRAYCONSTANT 的处理方式。这里要主要防止和后续规则的 ArrayComp 语法发生移进归约冲突。

对于数组初始化和数组拼接，其均可看作一个二元操作表达式，故我在 Binary 类中增加了对它们的处理。除此之外，务必要显式定义操作符的优先级和结合性，否则会产生大量的移进归约冲突。

对于取子数组表达式，其括号中的 Expr:Expr 表达式也可以看作一个二元操作，故也可在 Binary 中进行处理。

最后的下标动态访问，按照要求将整个 default 语句看作一个三元操作符，故我仿照 Binary 的格式，构造了 Trinary 类，这里要注意将 DEFULT 标注为%nonassoc, 以满足优先级要求。

2.6 Python 风格数组

这里我新建了 ArrayComp 类，用于处理该语句，注意这里容易与数组常量语句发生移进归约冲突，在我重新设计了数组常量语句的处理语法后这问题得以解决。

2.7 数组迭代语句

这一语句和上面的 Python 风格数组是类似的，我创建了新的类 ArrayForeach 来处理该语句。对于 Var 和 VarDef 两种不同类型，我通过类型检测来进行判断，分别进行处理，而没有再创建新的类型。

3 特殊自测用例

这里我对 var 的自动类型推导对于新增数组常量的支持，以及数组常量中元素对数组常量的支持 (即嵌套的数组类型常量) 进行了测试，这在给出的测试样例中并没有涉及。

我构造的测试用例如下：

```
class Main {
    static void main() {
        var xs = [[[true], ["true"]]];
    }
}
```

最终的输出结果如下:

```
program
    class Main <empty>
        static func main voidtype
            formals
            stmtblock
                assign
                    var xs
                    array const
                        array const
                            array const
                                boolconst true
                            array const
                                stringconst "true"
```

可见，程序仍能正确的处理上述情形。

4 实验总结

最终本地测试的样例全部通过。

总的来说，本次实验就是主要是给我们熟悉下编译作业的环境和代码结构，本身任务量并不大，但是由于会对后续作业造成影响，我还是做的相当谨慎的。

期待下一次的编译作业。