

Decaf_PA2 实验报告

计 31 班 刘智峰 2013011427

【实验内容】

本次实验将给出 decaf 基本框架，其中已经完成了《decaf 语言规范》中所描述语言特征的词法、语法及语义分析。现在，你需要在前一阶段的词法语法分析（同时生成抽象语法树）的基础上，继续针对 decaf 语言新增的语言特性进行语义分析。开始时，你需要将前一阶段的工作复制到本次实验的框架中，替换掉框架中的词法语法分析（同时生成抽象语法树）的功能。建议你首先要充分理解基本框架中语义分析的代码结构以及功能，参考框架中对相似语言特征的语义处理过程，根据自己对语言的理解完成新增语言特性的语义分析。

【实验一】

- 1) 实验描述：实现自增、自减单操作算子的语义分析。自增、自减单操作算子形如 `i++`, `++i`, `i--`, `--i`，其语义解释与 C 语言中一致。
- 2) 实现思路：通过观察 TestCase 中给的 `test_q1_decaf` 和 `test_q1_error.def` 以及相应的结果，可以发现，形如 `r=a++` 这样的运算，如果 `r` 和 `a` 的类型不一样，会报 “incompatible operand” 的错误。由于在 PA1 的实验中，我将 `++`、`--` 归入了 Unary 操作符，在观察 `TypeCheck.java` 文件中的 `visitUnary` 函数后，我也将 `++`、`--` 的实现添加到了 `visitUnary` 函数中。
- 3) 实现说明：
`++`、`--` 没有内部作用域，所以不用在 `BuildSym.java` 中添加，直接在 `TypeCheck.java` 中进行检查即可。具体做法直接参照 `visitUnary` 函数即可。

```

else if(expr.tag == Tree.POSTINC || expr.tag == Tree.PREINC){ //decaf_pa2 ++
    if (expr.expr.type.equal(BaseType.ERROR)
        || expr.expr.type.equal(BaseType.INT)) {
        expr.type = expr.expr.type;
    } else {
        issueError(new IncompatUnOpError(expr.getLocation(), "++",
            expr.expr.type.toString()));
        expr.type = BaseType.ERROR;
    }
}
else if(expr.tag == Tree.POSTDEC || expr.tag == Tree.PREDEC){ //decaf_pa2 --
    if (expr.expr.type.equal(BaseType.ERROR)
        || expr.expr.type.equal(BaseType.INT)) {
        expr.type = expr.expr.type;
    } else {
        issueError(new IncompatUnOpError(expr.getLocation(), "--",
            expr.expr.type.toString()));
        expr.type = BaseType.ERROR;
    }
}
}
}

```

【实验二】

- 1) 问题描述 :实现三操作数算子 $A ? B : C$ 的语义分析。三操作数算子形如 $A ? B : C$ 。
- 2) 实现思路 : “ $A ? B : C$ ” 对应规则 $\text{Expr} \rightarrow \text{Expr} ? \text{Expr} : \text{Expr}$ 为。正确的三元运算符需要满足 “?” 前的数据类型为 `bool` , 并且 `B` 与 `C` 的数据类型要一致。所以首先应对 `A` 进行类型检查。如果 `B` 与 `C` 中任何一个的数据类型已经为 `Error` , 则整个三元运算符的数据类型应赋为 `Error`。当两者具有非 `Error` 的数据类型后再判断两者数据类型是否一致, 不一致则报错, 且将整个三元运算符的数据类型赋为 `Error`。
- 3) 实现说明 :
?:操作符没有内部作用域, 所以不用在 `BuildSym.java` 中添加, 直接在 `TypeCheck.java` 中进行检查即可。通过观察 `test_q2_error.def` 以及相应的结果后, 我发现中间和后面这两个 `Expr` 不同时, 输出的语句在已有的报错文件中未曾出现, 所以我新加了一个报错处理的 `java` 文件 **`DifferentTypeError`** 来处理这类错误。

```

DifferentTypeError.java
1 package decaf.error;
2 import decaf.Location;
3
4 public class DifferentTypeError extends DecafError{ // decaf_pa2 A?B:C
5
6     private String middleType;
7     private String rightType;
8     public DifferentTypeError(Location location,String middleType, String rightType) {
9         super(location);
10        this.middleType = middleType;
11        this.rightType = rightType;
12    }
13
14    @Override
15    protected String getErrMsg() {
16        return "incompatible condition operates: " + middleType + " and " + rightType;
17    }
18 }
19
20
// decaf_pa2 A?B:C
@Override
public void visitQuestionAndColon(Tree.QuestionAndColon questionAndColon){
    checkTestExpr(questionAndColon.left);
    questionAndColon.middle.accept(this);
    questionAndColon.right.accept(this);
    Type middleType = questionAndColon.middle.type;
    Type rightType = questionAndColon.right.type;
    if(middleType.equal(BaseType.ERROR) || rightType.equal(BaseType.ERROR)){ // 后面两个表达式类型本省出错
        questionAndColon.type = BaseType.ERROR;
        return;
    }
    else{
        if(middleType.equal(rightType)){ //两表达式类型一致
            questionAndColon.type = middleType;
        }
        else{
            issueError(new DifferentTypeError(questionAndColon.loc,
                middleType.toString(), rightType.toString()));
            questionAndColon.type = BaseType.ERROR;
        }
    }
}
}

```

【实验三】

- 1) 问题描述：实现反射运算 numinstances 的语义分析。反射运算 numinstances 形如 numinstances (A), 其语义解释为：计算结果返回类 A 当前实例对象的个数。
- 2) 实现思路：在 PA1 中，numinstances()的实现是对照 instanceof()函数来做的。所以在进行语义分析时，也可以直接参照着 instanceof()的分析函数来做。通过观察 test_q3_decaf 和 test_q3_error.def 以及相应的结果可知，

对于 numinstances(A)所做的报错处理，就是判断 A 是否为一个 Class。若不是，则报错。

3) 实现说明：

Numinstances()没有内部作用域，所以不用在 BuildSym.java 中添加，直接在 TypeCheck.java 中进行检查即可。

```
@Override
public void visitTypeTest(Tree.TypeTest instanceofExpr) {
    instanceofExpr.instance.accept(this);
    if (!instanceofExpr.instance.type.isClassType()) {
        issueError(new NotClassError(instanceofExpr.instance.type
            .toString(), instanceofExpr.getLocation()));
    }
    Class c = table.lookupClass(instanceofExpr.className);
    instanceofExpr.symbol = c;
    instanceofExpr.type = BaseType.BOOL;
    if (c == null) {
        issueError(new ClassNotFoundError(instanceofExpr.getLocation(),
            instanceofExpr.className));
    }
}

//decaf_pa2 numinstances 仿造visitTypeTest
public void visitNumTest(Tree.NumTest numinstancesExpr) {
    Class c = table.lookupClass(numinstancesExpr.numinstance);
    numinstancesExpr.type = BaseType.INT; // 在这不定义的话会出现空指针 由计33的郭志芃同学提示
    if (c == null) {
        issueError(new ClassNotFoundError(numinstancesExpr.getLocation(),
            numinstancesExpr.numinstance));
    }
}
```

蓝色部分为参考的 instanceof()的报错处理，下面有注释的为自己实现的 numinstanceof()的出错处理。刚开始我没有设置 type 老出空指针的错误，最后请教了计 33 的郭志芃同学。

【实验四、五】

1) 问题描述：实现串行条件卫士语句和串行循环卫士语句的语义分析。语意见实验说明。

实现思路：串行条件卫士语句和串行循环卫士语句有内部域，所以需要在 BuildSym.java中构造符号表。刚开始不知道怎么下手，但看到实验说明中老师

提示的“参照If、While语句”后，依葫芦画瓢地构造了符号表，并在TypeCheck.java中实现了出错处理。由于在PA1的实验中，我为E : S定义了一个类型GuardedES，然后串行卫士和循环卫士都维护一个GuardedES类型的列表。所以我需要在BuildSym.java和TypeCheck.java中为GuardedES、GuardedIfStmt、GuardedDoStmt这三者建立立对应的语法。然后进行判断E是不是一个Bool类型的表达式、表达式的正确性等。而这些判断仿照If、While就能实现。同时，循环卫士和条件卫士有一点不同。循环卫士中允许出现break语句，而条件卫士中没有。break的实现直接照搬while的例子即可。

2) 实现说明：

BuildSym.java：

```
//decaf_pa2 并行语句
@Override
public void visitGuardedES(Tree.GuardedES guardedES) {
    if(guardedES.tree != null){
        guardedES.tree.accept(this);
    }
};

@Override
public void visitGuardedIFStmt(Tree.GuardedIFStmt guardedIFStmt) {
    for(Tree.GuardedES i : guardedIFStmt.guardedES){
        i.accept(this);
    }
};

@Override
public void visitGuardedDOStmt(GuardedDOStmt guardedDOStmt) {
    for(Tree.GuardedES i : guardedDOStmt.guardedES){
        i.accept(this);
    }
}
```

TypeCheck.java:

```

//decaf_pa2 GuardedIFStmt
@Override
public void visitGuardedES(GuardedES guardedES) {
    guardedES.expr.accept(this);
    guardedES.tree.accept(this);
}

@Override
public void visitGuardedIFStmt(GuardedIFStmt guardedIFStmt) {
    for(Tree.GuardedES i : guardedIFStmt.guardedES){
        i.accept(this);
    }
}

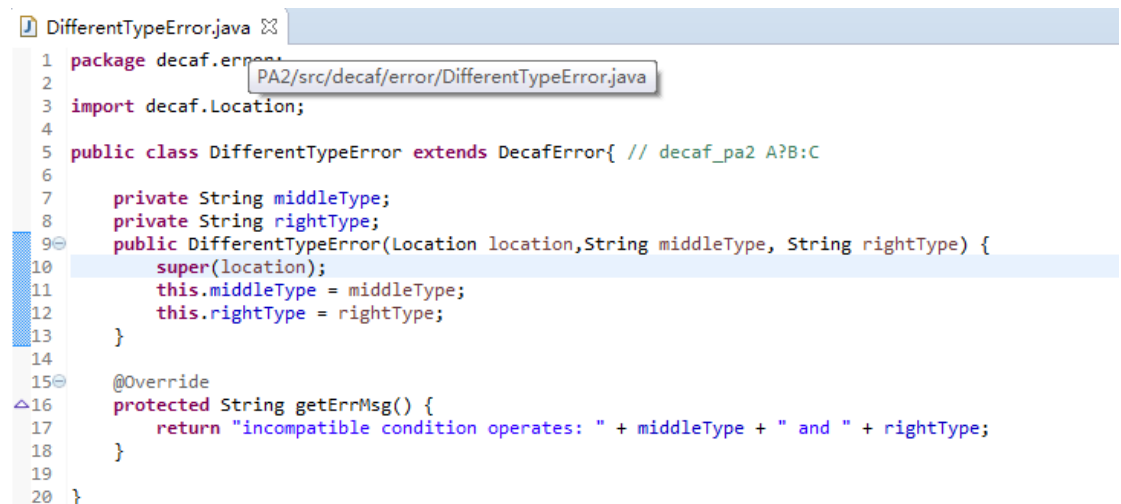
@Override
public void visitGuardedDOSmt(GuardedDOSmt guardedDOSmt) {
    for(Tree.GuardedES i : guardedDOSmt.guardedES){
        breaks.add(i); // 参照 visitwhile
        i.accept(this);
        breaks.pop();
    }
}
}

```

【增加的处理编译错误的类】

在实验二中已经提到过了。形如 A?B:C 的表达式 , 需要保证 B 和 C 为同一类型。

为了在 B、C 类型不同时进行报错，参照了 TestCase 中的结果后，我自己实现了一个 DifferentTypeError.java 文件用于此类错误的报错。



```

DifferentTypeError.java
1 package decaf.error;
2 import decaf.Location;
3
4 public class DifferentTypeError extends DecafError{ // decaf_pa2 A?B:C
5
6     private String middleType;
7     private String rightType;
8     public DifferentTypeError(Location location,String middleType, String rightType) {
9         super(location);
10        this.middleType = middleType;
11        this.rightType = rightType;
12    }
13
14    @Override
15    protected String getErrMsg() {
16        return "incompatible condition operates: " + middleType + " and " + rightType;
17    }
18
19 }
20

```

【实验总结】

本次实验遇到的最大的 bug 就是导入没导入清楚，导致做第 3 个小实验的时候一直出现空指针的问题，调了一个多小时，最后重新导入一遍就过了.....

相比于 PA1 ,本次实验我明显对 decaf 实验更加熟练。通过进一步的联系，我更加深入地学习了解了 decaf 语言语义分析的过程和操作。

本次实验的难度不算太大，但小 bug 也不少。感谢老师在实验说明中提示的可以参考 If、while 的实现方式。如果不参考，自己摸索，要花更多的时间。在实验过程中，我还请教了计 33 班的郭志芃同学。感谢老师、助教和郭志芃同学给我的帮助！