编译原理PA3实验报告

计63 肖朝军 2016011302

一、实验概述

本次实验内容为将AST翻译为TAC三地址码的一种表示。TAC三地址码的中间表示与汇编语言十分接近，因此在将AST翻译为TAC时，需要按照汇编的思想来生成中间表示的TAC码。这也是PA3最大的难点。

代码框架主要有两个部分，也就对应着对AST的两次扫描，第一次扫描为每一个类计算类的大小、构建虚函数表、计算每一个类成员的偏移量，为每一个方法的形参计算偏移量等。第二趟扫描完成所有TAC语句的翻译生成。在写代码时，需要转换思维，将思维转换至汇编风格，才能顺畅地写完本次PA。

二、实验内容

1. 类的浅复制(scopy(ident, E))

   该项特性实现思路为：先初始化一个类的实例temp，将E中内容一个字节一个字节拷贝到temp对应的内存空间中，最后将temp赋值给ident对应的变量。

```
@Override
public void visitScopy(Tree.Scopy scopy) {
    scopy.ident.accept( v: this);
    scopy.expr.accept( v: this);

    Temp tttmp = tr.genDirectCall(((ClassType)scopy.expr.type).getSymbol().getNewFuncLabel(), BaseType.INT);

    tr.copyClass(scopy.expr.val, tttmp, ((ClassType)scopy.expr.type).getSymbol().getSize());
    scopy.ident.symbol.setTemp(tttmp);
    scopy.ident.val = tttmp;
}
```

   其中copyClass为两个类之间的浅复制接口，

```
public void copyClass(Temp src, Temp dst, int size){
    int time = size / OffsetCounter.WORD_SIZE - 1;
    Temp sizeTmp = genLoadImm4(size);
    if (time != 0){
        if (time < 5){
            for (int i = 0; i < time; ++ i){
                Temp value = genLoad(genAdd(src, genLoadImm4(OffsetCounter.WORD_SIZE * (i + 1))), offset: 0);
                genStore(value, genAdd(dst, genLoadImm4(OffsetCounter.WORD_SIZE * (i + 1))), offset: 0);
            }
        } else {
            Temp unit = genLoadImm4(OffsetCounter.WORD_SIZE);
            Label loop = Label.createLabel();
            Label exit = Label.createLabel();
            dst = genAdd(dst, sizeTmp);
            genMark(loop);
            genAssign(dst, genSub(dst, unit));
            genAssign(sizeTmp, genSub(sizeTmp, unit));
            genBeqz(sizeTmp, exit);
            genStore(genAdd(src, sizeTmp), dst, offset: 0);
            genBranch(loop);
            genMark(exit);
        }
    }
}
```

2. sealed的支持

   这一步在PA2中的类型检查已经完成，在TAC码的生成中，不需要有额外的工作。

3. 支持条件卫士语句

   条件卫士语句实现思路与if的实现思路类似，不同地方在于，条件卫士语句中不存在else分支，且有一连串的if。

```
@Override
public void visitGuard(Tree.Guard guard){
    for (Tree.IfSub ifSub : guard.iflist){
        ifSub.accept( v: this);
    }
}

@Override
public void visitIfSub(Tree.IfSub ifSub) {
    ifSub.expr.accept( v: this);
    Label exit = Label.createLabel();
    tr.genBeqz(ifSub.expr.val, exit);
    ifSub.statementBody.accept( v: this);
    tr.genMark(exit);
}
```

4. 支持类型推导

   主要为修改第二次扫描的visitAssign函数，增加了对左值为var变量的处理，当左值为var变量时，增加赋值语句，并为变量的symbol设置变量。

```
@Override
public void visitAssign(Tree.Assign assign) {
    assign.left.accept( v: this);
    assign.expr.accept( v: this);

    if (assign.left.tag == Tree.VARSTMT){
        Temp t = Temp.createTempI4();
        Tree.VarStmt varstmt = (Tree.VarStmt) assign.left;
        t.sym = varstmt.symbol;
        varstmt.symbol.setTemp(t);

        tr.genAssign(((Tree.VarStmt) assign.left).symbol.getTemp(), assign.expr.val);
        return;
    }
```

5. (1) 数组初始化常量表达式

   分为几步，

   a) 检查n的大小，若n为负数则报错。

```
public void genCheckDoubleSize(Temp size){
    Label exit = Label.createLabel();
    Temp cond = genLes(size, genLoadImm4(0));
    genBeqz(cond, exit);
    Temp msg = genLoadStrConst(RuntimeError.NEGATIVE_ARR_SIZE_DOUBLE_MOD);
    genParm(msg);
    genIntrinsicCall(Intrinsic.PRINT_STRING);
    genIntrinsicCall(Intrinsic.HALT);
    genMark(exit);
}
```

   b) 利用translator中的接口产生一个新的数组

   c) 为数组中每个元素赋值

若元素类型为普通类型，则可以直接拷贝；若为类对象，则先调用类的构造函数，再对该变量进行赋值（类似于scopy中对象的浅复制）

需要注意的是，要写一个TAC的循环，这个循环与汇编中循环类似。

(2) 数组下标动态访问（E [E1] default E′）

该特性实现与if类似，判断条件为数组下标是否为负数或者是否越界，若下标不合法，则将返回值赋值为E′，否则赋值为E[E1]。

```
@Override
public void visitArrayDefault(Tree.ArrayDefault arrayDefault){
    arrayDefault.expr1.accept( v: this);
    arrayDefault.expr2.accept( v: this);
    arrayDefault.expr3.accept( v: this);


    Label err = Label.createLabel();
    Label good = Label.createLabel();
    Label exit = Label.createLabel();

    Temp tttmp = Temp.createTempI4();

    Temp length = tr.genLoad(arrayDefault.expr1.val, -OffsetCounter.WORD_SIZE);
    Temp cond = tr.genLes(arrayDefault.expr2.val, length);

    tr.genBeqz(cond, err);
    cond = tr.genLes(arrayDefault.expr2.val, tr.genLoadImm4(0));

    tr.genBeqz(cond, good);
    tr.genMark(err);
    // tr.genStore(arrayDefault.expr3.val, arrayDefault.val, 0);

    tr.genAssign(tttmp, arrayDefault.expr3.val);
    // tr.genAssign(arrayDefault.val, arrayDefault.expr3.val);
    tr.genBranch(exit);
    tr.genMark(good);

    Temp esz = tr.genLoadImm4(OffsetCounter.WORD_SIZE);
    Temp t = tr.genMul(arrayDefault.expr2.val, esz);
    Temp base = tr.genAdd(arrayDefault.expr1.val, t);
    Temp tttmp1 = tr.genLoad(base,  offset: 0);
    tr.genAssign(tttmp, tttmp1);

    tr.genMark(exit);
    arrayDefault.val = tttmp;

}
```

(3) 数组迭代语句

可以仿照while的实现进行实现

```java
@Override
public void visitForeachStmt(Tree.ForeachStmt foreachStmt){
    if (foreachStmt.typel != null)
        foreachStmt.typel.accept( v: this);

    Temp esz = tr.genLoadImm4(OffsetCounter.WORD_SIZE);

    Temp varX = Temp.createTempI4();
    Temp index = tr.genLoadImm4(0);


    Label loop = Label.createLabel();
    Label exit = Label.createLabel();

    foreachStmt.inExpr.accept( v: this);

    Temp length = tr.genLoad(foreachStmt.inExpr.val, -OffsetCounter.WORD_SIZE);

    tr.genMark(loop);
    // tr.genMemo("loop begin");

    Temp cond = tr.genLes(index, length);
    tr.genBeqz(cond, exit);

    varX = tr.genLoad(tr.genAdd(foreachStmt.inExpr.val, tr.genMul(index, esz)),  offset: 0);
    foreachStmt.symbol.setTemp(varX);

    foreachStmt.whileExpr.accept( v: this);
    tr.genBeqz(foreachStmt.whileExpr.val, exit);

    loopExits.push(exit);
    if (foreachStmt.foreachBody != null) {
        foreachStmt.foreachBody.accept( v: this);
    }

    tr.append(Tac.genAdd(index, index, tr.genLoadImm4(1)));
    tr.genBranch(loop);
    loopExits.pop();

    tr.genMark(exit);

}
```

三、实验总结

本次实验难度较大，主要一个原因是这次工作需要将一个AST的中间表示转换成一个TAC码的中间表示，这二者直接有着非常大的区别，因此实现起来有一个比较大的跨度。