编译原理PA3实验报告

张钰晖 计55 2015011372 yuhui-zh15@mails.tsinghua.edu.cn

任务描述

PA1阶段，我们完成了**词法分析**、**语法分析**，生成了**抽象语法树(AST)**。

PA2阶段，我们要基于PA1的抽象语法树，实现**构造符号表**、**静态语义检查**。

PA3阶段，我们在PA2的基础上，实现**语法制导的中间代码翻译**。

文件说明

在本阶段，以下文件非常重要，主要需要修改以下文件。

文件名	含义	说明
translate/Transpass2	第二趟扫描	对新增特性进行中间代码翻译
translate/Translater	翻译工作的辅助类	增加复数接口
frontend/*	编译器最前端	拷贝PA2文件
typecheck/*	语义检查部分	拷贝PA2文件
tree/*	抽象语法树的各种节点	沿用PA2修改
error/*	编译错误的类	增加新的错误类
type/BaseType	类型定义	增加复数类型
symbol/Class	符号定义	增加类型映射表

实验内容及实现

本次实验分为7个阶段，复用PA2结果、实现5个新文法特性、增加运行错误检查，主要修改列在下方。

步骤零：复用结果

本阶段需要复用PA2词法分析、语法分析、静态语义检查结果，实现以下新的文法特性：

- 整复数类型的支持：本阶段需要增加整复数类型，增加识别复数常量虚部功能，增加获取复数实部、虚部及强制转换复数表达式，增加复数打印语句。
- case表达式的支持：本阶段需要支持case表达式，语法为case(表达式) {常量1:表达式1,..., default:表达式}。
- super表达式的支持：本阶段需要支持super表达式，类似this表达式。

- 对象复制的支持：本阶段需要支持深复制dcopy()和浅复制scopy()表达式。
- 串行循环卫士的支持：本阶段需要支持串行循环卫士语句，语法为do E1:S1|||E2:S2... od。

本阶段修改文件如下：

文件名	修改
BaseLexer.java	复用修改
BaseParser.java	复用修改
Lexer.l	复用修改
Parser.y	复用修改
SemValue.java	复用修改
Tree.java	沿用修改
BaseType.java	沿用修改
BuildSym.java	复用修改
TypeCheck.java	复用修改
LocalScope.java	沿用修改
BadCopyArgError.java	复用修改
BadDoStmtArgError.java	复用修改
BadPrintCompArgError.java	复用修改
DifferentCaseTypeError.java	复用修改
DuplicateConditionError.java	复用修改
FieldNotSupportedError.java	复用修改
IncompatCaseExprError.java	复用修改
IncompatCopyAssignError.java	复用修改
NoParentClassError.java	复用修改
SuperInStaticFuncError.java	复用修改

> 注：复用修改=直接全部复制 沿用修改=仅复制改动部分

步骤一：支持整复数类型

- 修改类translate/Translater.java

增加基本类型Complex接口。

```
public class Translater {
    ...
    public Temp genNewComplex() {
        return genNewArray(genLoadImm4(3));
    }

    public Temp genLoadReal(Temp base) {
        return genLoad(base, 4);
    }

    public Temp genLoadImg(Temp base) {
        return genLoad(base, 8);
    }

    public void genStoreReal(Temp src, Temp base) {
        genStore(src, base, 4);
    }

    public void genStoreImg(Temp src, Temp base) {
        genStore(src, base, 8);
    }

    public Temp genToComplex(Temp src) {
        Temp dst = genNewComplex();
        genStoreReal(src, dst);
        genStoreImg(genLoadImm4(0), dst);
        return dst;
    }
    ...
}
```

- 修改类translate/TransPass2.java

新增基本类型Complex定义、赋值、转换、计算、打印支持。

```
public class TransPass2 extends Tree.Visitor {

    @Override
    public void visitVarDef(Tree.VarDef varDef) {
        if (varDef.symbol.isLocalVar()) {
            Temp t;
            if (varDef.type.type.equal(BaseType.COMPLEX)) {
                t = tr.genNewComplex();
            } else {
                t = Temp.createTempI4();
            }
            ...
        }
    }
```

```
15
16      @Override
17      public void visitBinary(Tree.Binary expr) {
18          ...
19          switch (expr.tag) {
20          case Tree.PLUS:
21              if (expr.left.type.equal(BaseType.COMPLEX) ||
expr.right.type.equal(BaseType.COMPLEX)) {
22                  expr.val = tr.genNewComplex();
23                  Temp left = expr.left.val;
24                  Temp right = expr.right.val;
25                  if (expr.left.type.equal(BaseType.INT)) left =
tr.genToComplex(left);
26                  if (expr.right.type.equal(BaseType.INT)) right =
tr.genToComplex(right);
27                  Temp leftReal = tr.genLoadReal(left);
28                  Temp leftImg = tr.genLoadImg(left);
29                  Temp rightReal = tr.genLoadReal(right);
30                  Temp rightImg = tr.genLoadImg(right);
31                  Temp dstReal = tr.genAdd(leftReal, rightReal);
32                  Temp dstImg = tr.genAdd(leftImg, rightImg);
33                  tr.genStoreReal(dstReal, expr.val);
34                  tr.genStoreImg(dstImg, expr.val);
35              } else {
36                  expr.val = tr.genAdd(expr.left.val, expr.right.val);
37              }
38              break;
39          ...
40          case Tree.MUL:
41              if (expr.left.type.equal(BaseType.COMPLEX) ||
expr.right.type.equal(BaseType.COMPLEX)) {
42                  expr.val = tr.genNewComplex();
43                  Temp left = expr.left.val;
44                  Temp right = expr.right.val;
45                  if (expr.left.type.equal(BaseType.INT)) left =
tr.genToComplex(left);
46                  if (expr.right.type.equal(BaseType.INT)) right =
tr.genToComplex(right);
47                  Temp leftReal = tr.genLoadReal(left);
48                  Temp leftImg = tr.genLoadImg(left);
49                  Temp rightReal = tr.genLoadReal(right);
50                  Temp rightImg = tr.genLoadImg(right);
51                  Temp dstReal = tr.genSub(tr.genMul(leftReal,
rightReal), tr.genMul(leftImg, rightImg));
52                  Temp dstImg = tr.genAdd(tr.genMul(leftReal, rightImg),
tr.genMul(leftImg, rightReal));
53                  tr.genStoreReal(dstReal, expr.val);
54                  tr.genStoreImg(dstImg, expr.val);
55              } else {
```

```
                expr.val = tr.genMul(expr.left.val, expr.right.val);
            }
            break;
        ...
        }
    }

    @Override
    public void visitAssign(Tree.Assign assign) {
        ...
        case LOCAL_VAR:
            tr.genAssign(((Tree.Ident) assign.left).symbol.getTemp(),
            if (assign.left.type.equal(BaseType.COMPLEX)) {
                Temp left = ((Tree.Ident)
assign.left).symbol.getTemp();
                Temp right = assign.expr.val;
                if (assign.expr.type.equal(BaseType.INT)) right =
tr.genToComplex(right);
                Temp rightReal = tr.genLoadReal(right);
                Temp rightImg = tr.genLoadImg(right);
                tr.genStoreReal(rightReal, left);
                tr.genStoreImg(rightImg, left);
            } else {
                tr.genAssign(((Tree.Ident)
assign.left).symbol.getTemp(),
                    assign.expr.val);
            }
            break;
        }
    }

    @Override
    public void visitLiteral(Tree.Literal literal) {
        ...
        case Tree.IMG:
            literal.val = tr.genNewComplex();
            tr.genStoreReal(tr.genLoadImm4(0), literal.val);
            tr.genStoreImg(tr.genLoadImm4((Integer)literal.value),
literal.val);
            break;
        ...
    }

    @Override
    public void visitUnary(Tree.Unary expr) {
        ...
        case Tree.GETREAL:
            expr.val = tr.genLoadReal(expr.expr.val);
            break;
```

```
        case Tree.GETIMG:
            expr.val = tr.genLoadImg(expr.expr.val);
            break;
        case Tree.TOCOMPLEX:
            expr.val = tr.genToComplex(expr.expr.val);
            break;
        ...
    }

    @Override
    public void visitPrintComp(Tree.PrintComp printCompStmt) {
        for (Tree.Expr r : printCompStmt.exprs) {
            r.accept(this);
            tr.genParm(tr.genLoadReal(r.val));
            tr.genIntrinsicCall(Intrinsic.PRINT_INT);
            tr.genParm(tr.genLoadStrConst("+"));
            tr.genIntrinsicCall(Intrinsic.PRINT_STRING);
            tr.genParm(tr.genLoadImg(r.val));
            tr.genIntrinsicCall(Intrinsic.PRINT_INT);
            tr.genParm(tr.genLoadStrConst("j"));
            tr.genIntrinsicCall(Intrinsic.PRINT_STRING);
        }
    }

}
```

阶段二：支持Case表达式

- 修改类translate/TransPass2.java

新增函数visitCaseExpr，将case表达式翻译成TAC。

```
public class TransPass2 extends Tree.Visitor {

    @Override
    public void visitCaseExpr(Tree.CaseExpr caseExpr) {
        caseExpr.val = Temp.createTempI4();

        caseExpr.conditionexpr.accept(this);
        caseExpr.defaultexpr.accept(this);
        tr.genAssign(caseExpr.val, caseExpr.defaultexpr.val);

        Label next = Label.createLabel();
        for (Tree.ACaseExpr aCaseExpr: caseExpr.casesexpr) {
            tr.genMark(next);
            aCaseExpr.literal.accept(this);
            Temp isEqual = tr.genEqu(caseExpr.conditionexpr.val,
aCaseExpr.literal.val);
            tr.genBeqz(isEqual, next = Label.createLabel());
```

```
            aCaseExpr.expr.accept(this);
            tr.genAssign(caseExpr.val, aCaseExpr.expr.val);
        }
        tr.genMark(next); // Guard for end
    }

}
```

阶段三：支持Super表达式

- 修改类translate/TransPass2.java

新增变量currentFunc，新增函数visitSuperExpr，修改函数visitCallExpr和visitMethodDef，支持super表达式。

```
public class TransPass2 extends Tree.Visitor {

    private Function currentFunc;

    @Override
    public void visitMethodDef(Tree.MethodDef funcDefn) {
        currentFunc = funcDefn.symbol;
        ...
        currentFunc = null;
    }

    @Override
    public void visitSuperExpr(Tree.SuperExpr superExpr) {
        superExpr.val = currentThis;
    }

    @Override
    public void visitCallExpr(Tree.CallExpr callExpr) {
        if (callExpr.isArrayLength) {
            ...
        } else {
            ...
            else {
                Temp vt = tr.genLoad(callExpr.receiver.val, 0);
                Temp func = tr.genLoad(vt, callExpr.symbol.getOffset());
                callExpr.val = tr.genIndirectCall(func, callExpr.symbol
                        .getReturnType());
                Temp vt = tr.genLoad(callExpr.receiver.val, 0);
                if (callExpr.receiver instanceof Tree.ThisExpr) {
                    ClassType thisType =
currentFunc.getScope().getOwner().getType();
                    VTable thisVtable =
thisType.getSymbol().getVtable();
                    vt = tr.genLoadVTable(thisVtable);
```

```
                }
                if (callExpr.receiver instanceof Tree.SuperExpr) {
                    ClassType superType =
currentFunc.getScope().getOwner().getType().getParentType();
                    while (superType != null &&
superType.getClassScope().lookupVisible(callExpr.method) == null) {
                        superType = superType.getParentType();
                    }
                    VTable superVtable =
superType.getSymbol().getVtable();
                    vt = tr.genLoadVTable(superVtable);
                }
                Temp func = tr.genLoad(vt, callExpr.symbol.getOffset());
                callExpr.val = tr.genIndirectCall(func,
callExpr.symbol.getReturnType());
            }
        }
    }

}
```

阶段四：支持对象复制

- 修改类translate/TransPass2.java

新增函数visitScopyExpr、visitDcopyExpr和dfsCopy，支持scopy和dcopy表达式。

```
public class TransPass2 extends Tree.Visitor {

    @Override
    public void visitScopyExpr(Tree.ScopyExpr scopyExpr) {
        scopyExpr.expr.accept(this);
        Temp src = scopyExpr.expr.val;

        Class cls = ((ClassType) scopyExpr.expr.type).getSymbol();
        Temp dst = tr.genDirectCall(cls.getNewFuncLabel(),
BaseType.INT);
        scopyExpr.val = dst;

        Temp vtable = tr.genLoad(src, 0);
        tr.genStore(vtable, dst, 0);

        for (int offset = OffsetCounter.WORD_SIZE; offset <
cls.getSize(); offset += OffsetCounter.WORD_SIZE) {
            Temp var = tr.genLoad(src, offset);

            if (cls.getTypeMap().get(offset).equal(BaseType.COMPLEX)) {
                Temp real = tr.genLoadReal(var);
                Temp img = tr.genLoadImg(var);
```

```
                Temp newcomp = tr.genNewComplex();
                tr.genStoreReal(real, newcomp);
                tr.genStoreImg(img, newcomp);
                tr.genStore(newcomp, dst, offset);
            } else {
                tr.genStore(var, dst, offset);
            }
        }
    }

    @Override
    public void visitDcopyExpr(Tree.DcopyExpr dcopyExpr) {
        dcopyExpr.expr.accept(this);
        Temp src = dcopyExpr.expr.val;
        Class cls = ((ClassType) dcopyExpr.expr.type).getSymbol();
        dcopyExpr.val = dfsCopy(cls, src);
    }

    private Temp dfsCopy(Class cls, Temp src) {
        Temp dst = tr.genDirectCall(cls.getNewFuncLabel(),
BaseType.INT);

        Temp vtable = tr.genLoad(src, 0);
        tr.genStore(vtable, dst, 0);

        for (int offset = OffsetCounter.WORD_SIZE; offset <
cls.getSize(); offset += OffsetCounter.WORD_SIZE) {
            Temp var = tr.genLoad(src, offset);

            if (cls.getTypeMap().get(offset).isClassType()) {
                Class classtype = ((ClassType)
cls.getTypeMap().get(offset)).getSymbol();
                Temp newclass = dfsCopy(classtype, var);
                tr.genStore(newclass, dst, offset);
            } else if
(cls.getTypeMap().get(offset).equal(BaseType.COMPLEX)) {
                Temp real = tr.genLoadReal(var);
                Temp img = tr.genLoadImg(var);
                Temp newcomp = tr.genNewComplex();
                tr.genStoreReal(real, newcomp);
                tr.genStoreImg(img, newcomp);
                tr.genStore(newcomp, dst, offset);
            } else {
                tr.genStore(var, dst, offset);
            }
        }
        return dst;
    }

```

```
66  }
```

- 修改类symbol/Class.java

新增变量typeMap，用于返回虚表偏移量对应变量类型。

```
1   public class Class extends Symbol {
2
3       private Map<Integer, Type> typeMap = new HashMap<>();
4
5       public Map<Integer, Type> getTypeMap() {
6           return typeMap;
7       }
8
9       public void resolveFieldOrder() {
10          ...
11          if (parentName != null) {
12              ...
13              for (Entry entry : parent.getTypeMap().entrySet()) {
14                  this.typeMap.put((Integer) entry.getKey(), (Type)
    entry.getValue());
15              }
16          } else {
17              ...
18          }
19          ...
20          while (iter.hasNext()) {
21              ...
22              if (sym.isVariable()) {
23                  ...
24                  this.typeMap.put(size, sym.getType());
25                  size += OffsetCounter.WORD_SIZE;
26              } else if (!((Function) sym).isStatik()) {
27                  ...
28              }
29          }
30      }
31
32  }
```

阶段五：支持串行循环卫士

- 修改类translate/TransPass2.java

新增函数visitDoStmt，将DoStmt翻译成TAC。

```
1   public class TransPass2 extends Tree.Visitor {
2
3       @Override
```

```
    public void visitDoStmt(Tree.DoStmt doStmt) {
        Label loop = Label.createLabel();
        Label exit = Label.createLabel();

        tr.genMark(loop);
        loopExits.push(exit);

        Label next = Label.createLabel();
        for (Tree.DoSubStmt doSubStmt: doStmt.dslist) {
            tr.genMark(next);
            doSubStmt.expr.accept(this);
            tr.genBeqz(doSubStmt.expr.val, next = Label.createLabel());
            doSubStmt.stmt.accept(this);
            tr.genBranch(loop);
        }
        tr.genMark(next); // Guard for end

        loopExits.pop();
        tr.genMark(exit);
    }

}
```

阶段六：除0运行错误检查

- 修改类translate/TransPass2.java

修改函数visitBinOp，增加运行错误检查。

```
public class TransPass2 extends Tree.Visitor {

    @Override
    public void visitBinary(Tree.Binary expr) {
        ...
        case Tree.DIV:
            tr.genCheckDivisionByZero(expr.right.val);
            ...
            break;
        case Tree.MOD:
            tr.genCheckDivisionByZero(expr.right.val);
            ...
            break;
        ...
    }

}
```

- 修改类translate/Translater.java

新增函数genCheckDivisionByZero，增加除0错误检查。

```
public class Translater {

    public void genCheckDivisionByZero(Temp division) {
        Label exit = Label.createLabel();
        genBnez(division, exit);
        Temp msg = genLoadStrConst(RuntimeError.DIVISION_BY_ZERO);
        genParm(msg);
        genIntrinsicCall(Intrinsic.PRINT_STRING);
        genIntrinsicCall(Intrinsic.HALT);
        genMark(exit);
    }

}
```

- 修改类error/RuntimeError.java

新增除0错误。

```
public final class RuntimeError {

    public static final String DIVISION_BY_ZERO = "Decaf runtime error:
Division by zero error.\n";

}
```

技巧心得

本次作业难度相对PA2再次提升，通过以下方法可以加速编程。

0. 仔细阅读代码框架

如果不理解代码框架，本次作业几乎无从下手。

1. 仔细阅读测试样例及正确输出

当充分理解测试样例和正确输出后，才能理解本次任务和新的语法特性。

2. 仔细阅读中间tac文件

中间tac文件是理解本次任务的关键所在，一定要同时对照样例和tac文件，能够很快的理解任务。

总结

本次实验PA3相对实验PA2难度再次提升，一个明显的特点就是尽管实现代码量并不算很大，但必须非常充分理解整个框架的结构，完全理解新的语法特性，需要改动的地方较多。在实现PA3过程中，通过以上两个技巧，加速了编程，尽管实现的过程有些痛苦，但在实现的过程中，充分的锻炼了笔者的编程能力，对中间代码翻译的相关概念理解有了质的提高，笔者在实践之中真正感受到了编译的神奇之处。