Decaf PA2 实验报告

计64 翁家翌 2016011446

scopy

1. 修改 `decaf/typecheck/Typecheck.java`，添加函数 `public void visitScopy(Tree.Scopy scopy)` 如下（增加了 `BadScopySrcError`、`BadScopyArgError` 的判断）

```
public void visitScopy(Tree.Scopy scopy) {
    Symbol v = table.lookupBeforeLocation(scopy.ident, scopy.getLocation());
    if (v == null) {
        issueError(new UndeclVarError(scopy.getLocation(), scopy.ident));
        scopy.type = BaseType.ERROR;
        return;
    }
    scopy.expr.accept(this);
    if (v.getType().isClassType()) {
        if (!scopy.expr.type.equal(v.getType())) {
            issueError(new BadScopySrcError(scopy.getLocation(),
v.getType().toString(), scopy.expr.type.toString()));
            scopy.type = BaseType.ERROR;
        }
    }
    else {
        issueError(new BadScopyArgError(scopy.getLocation(), "dst",
v.getType().toString()));
        if (!scopy.expr.type.isClassType()) {
            issueError(new BadScopyArgError(scopy.getLocation(), "src",
scopy.expr.type.toString()));
            scopy.type = BaseType.ERROR;
        }
    }
}
```

sealed

1. 修改 `decaf/symbol/Class.java`，在该类别中添加 `boolean isSealed;`
2. 修改 `decaf/typecheck/TypeCheck.java`，修改 `visitTopLevel` 函数一开始的第1个for的 `Class c` 初始化为 `Class c = new Class(cd.name, cd.parent, cd.getLocation(), cd.sealed);`（加上 `sealed` 参数）
3. 修改 `decaf/typecheck/TypeCheck.java`，修改 `visitTopLevel` 函数一开始的第2个for的条件判断（如果一个class的父亲是sealed属性则报 `BadSealedInherError` ）如下：

```
if (cd.parent != null && c.getParent() != null && c.getParent().getSealed()) {
    issueError(new BadSealedInherError(cd.getLocation()));
    c.detachParent();
}
```

GuardStmt

1. 修改 `decaf/typecheck/BuildSym.java`，添加函数 `public void visitGuardedES(Tree.GuardedES guardedES)` 和 `public void visitGuardStmt(Tree.GuardStmt guardStmt)` 使其遍历AST

```
@Override
public void visitGuardedES(Tree.GuardedES guardedES) {
    if (guardedES.stmt != null)
        guardedES.stmt.accept(this);
}

@Override
public void visitGuardStmt(Tree.GuardStmt guardStmt) {
    for (Tree.GuardedES i: guardStmt.guardedES) {
        i.accept(this);
    }
}
```

2. 修改 `decaf/typecheck/TypeCheck.java`，添加函数 `public void visitGuardedES(Tree.GuardedES guardedES)` 和 `public void visitGuardStmt(Tree.GuardStmt guardStmt)`

```
@Override
public void visitGuardedES(Tree.GuardedES guardedES) {
    guardedES.expr.accept(this);
    guardedES.stmt.accept(this);
    checkTestExpr(guardedES.expr);
}

@Override
public void visitGuardStmt(Tree.GuardStmt guardStmt) {
    for (Tree.GuardedES i: guardStmt.guardedES) {
        i.accept(this);
    }
}
```

VarIdent

1. 修改 `decaf/type/BaseType.java`，在该类别中添加 `public static final BaseType UNKNOWN = new BaseType("unknown");` 的类别
2. 修改 `decaf/symbol/Variable.java`，在该类别中添加 `public void setType(Type type)` 函数用来修改Variable中的UNKNOWN类别
3. 修改 `decaf/tree/Tree.java` 中的 `LValue` 类，添加变量 `public Variable symbol;` 用来存储符号为了后续修改类别

4. 修改 `visitAssign` 函数

    1. 向 `decaf/typecheck/BuildSym.java` 文件添加 `public void visitAssign(Tree.Assign assign)` 函数，为了把var类型声明添加到 `LocalScope` 中，具体如下：

    ```
    public void visitAssign(Tree.Assign assign) {
        assign.left.accept(this);
        assign.expr.accept(this);
    }
    ```

    2. 修改 `decaf/typecheck/TypeCheck.java` 中的 `visitAssign` 函数，添加如下代码以修改var类型的type：

    ```
    if (assign.left.type.equal(BaseType.UNKNOWN)) { // var
        assign.left.type = assign.expr.type;
        assign.left.symbol.setType(assign.left.type);
    }
    ```

5. 添加 `visitVarIdent` 函数

    1. 向 `decaf/typecheck/BuildSym.java` 文件添加 `public void visitVarIdent(Tree.VarIdent varIdent)` 函数（就抄了一下 `visitVarDef` 的函数，让它能把x加入 `LocalScope` 中）：

    ```
    public void visitVarIdent(Tree.VarIdent varIdent) {
        Variable v = new Variable(varIdent.name, BaseType.UNKNOWN,
    varIdent.getLocation());
        Symbol sym = table.lookup(varIdent.name, true);
        if (sym != null) {
            if (table.getCurrentScope().equals(sym.getScope())) {
                issueError(new DeclConflictError(v.getLocation(), v.getName(),
                        sym.getLocation()));
            } else if ((sym.getScope().isFormalScope() &&
    table.getCurrentScope().isLocalScope() &&
    ((LocalScope)table.getCurrentScope()).isCombinedtoFormal() )) {
                issueError(new DeclConflictError(v.getLocation(), v.getName(),
                        sym.getLocation()));
            } else {
                table.declare(v);
            }
        }
        else {
            table.declare(v);
        }
        varIdent.symbol = v;
    }
    ```

    2. 向 `decaf/typecheck/BuildSym.java` 文件添加 `public void visitVarIdent(Tree.VarIdent varIdent)` 函数获取正确的type：

```
public void visitVarIdent(Tree.VarIdent varIdent) {
    varIdent.type = BaseType.UNKNOWN;
}
```

ArrayConst

1. 向 `decaf/typecheck/TypeCheck.java` 中添加函数 `public void visitArrayConst(Tree.ArrayConst arrayConst)`，用来检查数组常量的类别，具体如下：

```
public void visitArrayConst(Tree.ArrayConst arrayConst) {
    for (Tree.Expr i: arrayConst.arrayconst)
        i.accept(this);
    arrayConst.type = arrayConst.arrayconst.get(0).type;
    for (Tree.Expr i: arrayConst.arrayconst) {
        if (!arrayConst.type.equal(i.type)) {
            arrayConst.type = BaseType.ERROR;
            return;
        }
    }
    arrayConst.type = new ArrayType(arrayConst.type);
}
```

BinOP: %% and ++

1. 向 `decaf/typecheck/TypeCheck.java` 中添加 `checkBeArr` 函数，用来判断一个类别是否能成为数组的 element type：

```
private boolean checkBeArr(Type type) {
    return (type.equal(BaseType.INT) || type.equal(BaseType.STRING)
            || type.equal(BaseType.BOOL) || type.isArrayType());
}
```

2. 修改 `decaf/typecheck/TypeCheck.java` 中的 `private Type checkBinaryOp(Tree.Expr left, Tree.Expr right, int op, Location location)` 函数：
    1. 一开始把 `var %% xx` 这种判掉：

```
boolean isVar = left.type.equal(BaseType.UNKNOWN) ||
right.type.equal(BaseType.UNKNOWN);
if (op == Tree.MOMO && left.type.equal(BaseType.UNKNOWN)) {
    issueError(new BadArrElementError(location));
    return returnType;
}
```

    2. 添加对 `++` 和 `%%` 的判断：

```
case Tree.MOMO:
    compatible = checkBeArr(left.type) && right.type.equal(BaseType.INT);
    returnType = new ArrayType(left.type);
    break;
case Tree.PLPL:
    compatible = left.type.isArrayType() && right.type.isArrayType() &&
left.type.equal(right.type);
    returnType = left.type;
    break;
```

3. 修改报错条件如下：

```
if (isVar)
    returnType = BaseType.ERROR;
if (!compatible || isVar) {
    issueError(new IncompatBinOpError(location, left.type.toString(),
            Parser.opStr(op), right.type.toString()));
}
```

Default

1. 向 `decaf/typecheck/TypeCheck.java` 中添加 `visitDefault`，添加 `BadArrIndexError`、`BadDefError` 和 `BadArrOperArgError`：

```
public void visitDefault(Tree.Default deft) {
    deft.array.accept(this);
    deft.index.accept(this);
    deft.deft.accept(this);
    deft.type = BaseType.ERROR;
    if (!deft.index.type.equal(BaseType.INT))
        issueError(new BadArrIndexError(deft.index.getLocation()));
    if (deft.array.type.isArrayType()) {
        Type ele = ((ArrayType)deft.array.type).getElementType();
        deft.type = ele;
        if (!ele.equal(deft.deft.type))
            issueError(new BadDefError(deft.index.getLocation(), ele.toString(),
deft.deft.type.toString()));
    }
    else {
        issueError(new BadArrOperArgError(deft.array.getLocation()));
        if (checkBeArr(deft.deft.type))
            deft.type = deft.deft.type;
    }
}
```

foreach

1. 修改 `decaf/tree/Tree.java` 中的 `Block` 类，添加变量 `boolean isforeach;`

2. 修改 `decaf/tree/Tree.java` 中的 `ForEach` 类，添加变量 `public Block block;` 和 `public LocalScope associateScope;`
3. 修改 `decaf/tree/Tree.java` 中的 `ExDef` 类，添加变量 `public Variable symbol;` 用来存储符号为了后续修改类别
4. 修改 `block` 类：
    1. 修改 `decaf/typecheck/BuildSym.java` 中的 `Block` 类，使得当执行到 `foreachBlock` 时候不新建 `LocalScope`，具体如下：

```
public void visitBlock(Tree.Block block) {
    if (!block.isforeach) {
        block.associatedScope = new LocalScope(block);
        table.open(block.associatedScope);
    }
    for (Tree s : block.block) {
        s.accept(this);
    }
    if (!block.isforeach)
        table.close();
}
```

    2. 修改 `decaf/typecheck/TypeCheck.java` 中的 `Block` 类，使得当执行到 `foreachBlock` 时候不新建 `LocalScope`，具体如下：

```
public void visitBlock(Tree.Block block) {
    if (!block.isforeach)
        table.open(block.associatedScope);
    for (Tree s : block.block) {
        s.accept(this);
    }
    if (!block.isforeach)
        table.close();
}
```

5. 添加函数 `visitExDef` ：
    1. 添加 `decaf/typecheck/BuildSym.java` 中的 `visitExDef` 函数，获取type：

```
public void visitExDef(Tree.ExDef exdef) {
    if (exdef.typel == null) {
        exdef.type = BaseType.UNKNOWN;
    }
    else {
        exdef.typel.accept(this);
        exdef.type = exdef.typel.type;
    }
}
```

    2. 添加 `decaf/typecheck/TypeCheck.java` 中的 `visitExDef` 函数，还是获取type，如果重复声明则报错：

```
public void visitExDef(Tree.ExDef exdef) {
    Symbol v = table.lookupBeforeLocation(exdef.name, exdef.getLocation());
    if (exdef.typel == null) {
        if (v != null && exdef.getLocation().compareTo(v.getLocation()) > 0) {
            issueError(new DeclConflictError(exdef.getLocation(), exdef.name,
v.getLocation()));
        }
        exdef.type = BaseType.UNKNOWN;
    }
    else {
        exdef.typel.accept(this);
        exdef.type = exdef.typel.type;
    }
}
```

6. 添加函数 `visitForEach`：

    1. 添加 `decaf/typecheck/BuildSym.java` 中的 `visitForEach` 函数，向block中添加x的声明：

    ```
    public void visitForEach(Tree.ForEach foreach) {
        foreach.block = (Tree.Block) foreach.stmt;
        foreach.block.isforeach = true;
        foreach.associatedScope = new LocalScope(foreach.block);
        foreach.exdef.accept(this);
        table.open(foreach.associatedScope);
        Variable v = new Variable(foreach.exdef.name, foreach.exdef.type,
    foreach.exdef.getLocation());
        table.declare(v);
        foreach.exdef.symbol = v;
        foreach.expr1.accept(this);
        foreach.expr2.accept(this);
        foreach.stmt.accept(this);
        table.close();
    }
    ```

    2. 添加 `decaf/typecheck/TypeCheck.java` 中的 `visitForEach` 函数，添加 `BadArrOperArgError` 的判断：

    ```
    public void visitForEach(Tree.ForEach foreach) {
        table.open(foreach.associatedScope);
        foreach.exdef.accept(this);
        foreach.expr1.accept(this);
        boolean iserr = false;
        if (foreach.exdef.type.equal(BaseType.UNKNOWN)) { // var
            if (foreach.expr1.type.isArrayType()) {
                foreach.exdef.type =
    ((ArrayType)foreach.expr1.type).getElementType();
                foreach.exdef.symbol.setType(foreach.exdef.type);
            }
            else {
                iserr = true;
                if (foreach.expr1.type != BaseType.ERROR) {
    ```

```
                foreach.exdef.type = BaseType.ERROR;
                issueError(new
BadArrOperArgError(foreach.expr1.getLocation()));
            }
        }
    }
    else {
        if (!foreach.expr1.type.isArrayType() && foreach.exdef.type !=
BaseType.ERROR) {
            iserr = true;
            issueError(new BadArrOperArgError(foreach.expr1.getLocation()));
        }
    }
    checkTestExpr(foreach.expr2);
    foreach.expr2.accept(this);
    if (!iserr) {
        breaks.add(foreach);
        foreach.stmt.accept(this);
        breaks.pop();
    }
    table.close();
}
```