# decaf_PA3 实验报告

电 53 魏宇轩 2015011942

## 1. 类的浅复制

之前的 PA1 和 PA2 中，错误把 Tree.Scopy 类的第一个属性 left 设置为了 String 类型，导致 PA3 中无法通过该属性访问属性的 symbol. 解决方法：

（1）修改 parse.y 中对象浅复制语句的语义动作

```
OCStmt          :   SCOPY '(' IDENTIFIER ',' Expr ')'
                    {
                        $$.stmt = new Tree.Scopy(new Tree.Ident(null, $3.ident, $3.loc, false), $5.expr, $1.loc);
                    }
                ;
```

（2）修改 Tree.java 中 Scopy 类，将第一个属性由 int 改为 Ident, 并相应修改构造函数和 printTo 方法

（3）修改 TypeCheck.java 中的 visitScopy 方法，将 Tree.Scopy.left 的属性获取由根据 String 从符号表 table 中查找改为直接从 Tree.Ident 的属性中获得

浅复制的含义是：如果类属性为基本类型，则复制类属性的值；如果类属性为类对象（类嵌套），则复制类对象的引用。与之相对应的，深复制会递归复制类对象中的内容。

TAC 中用一个指针指向对象内容，因此对象浅复制的实现为遍历整个对象空间，将 src 的每个单元的值赋值给 dst 对应的单元。具体实现：修改 TransPass2.java，增加 visitScopy 方法，得到 src 对象的大小 size。设置指针 i，当 i 小于 size 时，从(src+i)处读取一个单位大小的内容，写入(dst+i).

为了增加可扩展性（支持类对象的”E %% n”表达式），在 Translater.java 中将上述过程封装为 genScopy 方法。

## 2. sealed 的支持

PA2 已完全支持，PA3 不需要修改或增添。

## 3. 串行条件卫士语句

（1）在 TransPass2.java 中增加 visitGuarded 方法，遍历 Tree.Guarded 的属性 GuardList，对于 GuardList 中的每一个 Guard 对象，调用 accept 方法。

（2）在 TransPass2.java 中增加 visitGuard 方法，在 TAC 程序段的尾端设置 Label pass，根据布尔表达式的值跳转到 Label pass 或顺序执行 Guard stmt 中的 TAC 程序段。TAC 程序：

```
    _T0 = expr.val
    if (_T0 == 0) branch _L0
    stmt.TAC
_L0:
```

（3）发现并修正了 PA2 中的一个错误，在 TypeCheck.java 的 visitGuard 方法的末尾补上了之前 PA2 中遗漏的”stmt.accept(this)”语句.

## 4. 支持简单类型推导

（1）参考 TransPass1.java 中 visitVarDef 方法，增加 visitIdent 方法，在 Tree.Ident 形如”var

identifier”时，仿照 visitVarDef 将 identifier 加入变量列表并计算偏移量。为了能否访问到 AST 中靠近叶节点的 Tree.Ident，需要在 visitMethodDef 中增加”funcDef.body.accept(this)”语句并增加 visitBlock, visitAssign 方法，递归地访问 Tree.Ident. **特别注意访问 block 中的 stmt 时，只有 instanceof Assign 的 stmt 才访问，其余 stmt 不能访问，否则会引发错误**。

（2）修改 TransPass2.java 中 visitIdent 方法，在 Tree.Ident 形如”var identifier”时，调用 createTempI4 在 TAC 程序中定义新的临时变量。

**5. 数组初始化常量**

（1）修改 TransPass2.java 的 visitBinary 方法，当操作符为”%%” 时调用 Translater.genCreatedArray 方法。

（2）修改 Translater.java，参照 genNewArray 方法增加 genCreatedArray 方法，将 genNewArray 方法中数组元素初始化为 0 改成初始化为 E.

（3）genCreatedArray 中，开辟（数组长度+1 个单位）的空间，在首地址存入数组长度值，然后遍历数组空间：

当 E 不是类对象时，直接把 E 的值存入相应的指针位置（见 TAC 程序）；

当 E 是类对象时，调用 genScopy 方法生成 E 的对象浅复制 E’，将 E’的引用存入相应的指针位置。

（4）修改 RuntimeError.java，为 RuntimeError 类增加静态属性字符串，表示数组初始化常量负数长度错误。

（5）修改 Translater.java，增加 genCheckCreatedArray 方法，检查”E %% n”中 n 是否为负数。

TAC 程序：

```
    _T0 = n.val
    _T1 = 0
    _T2 = (_T0 < T_1)
    if (_T1 == 0) branch _L0
    _T3 = “Decaf runtime error: The length of the created array should not be less than 0.\n”
    parm _T3
    call _PrintString
    call _Halt
_L0:
    _T4 = 4
    _T5 = (_T0 * _T4)
    _T6 = (_T4 + _T5)
    parm _T6
    _T7 = call_Alloc
    *(_T7 + 0) = _T0
    _T8 = E.val
    _T7 = (_T7 + _T6)
_L1:
    _T6 = (_T6 - _T4)
    if (_T6 == 0) branch _L2
    _T7 = (_T7 - _T4)
    *(_T7 + 0) = _T8
    branch _L1
```

```
_L2:
```

**6. 数组下标动态访问**

修改 TransPass2.java，参照 Translater.genCheckIndex 方法，增加 visitDynamicIndex 方法：

（1）修改控制逻辑：将 Translater.genCheckIndex 数组越界报错的代码替换为计算 default 表达式的代码，并增加一个 Label exit，放在 Dynamic Index TAC 代码段最后，在计算完 default 表达式后跳转到 exit.

（2）修改返回值：定义临时变量 ultimate，起分支汇总作用，如果数组越界，则在计算 default 表达式之后调用 genAssign 将 default 表达式的计算结果赋值给 ultimate；否则在数组取址代码段后将数组元素赋值给 ultimate. 最后将 ultimate 的值作为数组下标动态访问表达式的值。

**7. 数组迭代语句**

（1）增加 PA2 遗漏的对于 break 语句的支持：

修改 TypeCheck.java 的 visitForeachStmt 方法，用 breaks.push(foreachstmt)和 breaks.pop()包裹 foreachstmt.block 的处理程序段。

（2）修改 TransPass2.java，增加 visitForeachStmt 方法：

定义临时变量 i 和 len 分别代表迭代器和数组长度。设置两个标签，分别表示数组越界判断入口 check 和数组迭代语句结束出口 exit。当迭代器 i 大于等于数组长度 len 时，跳转到 exit，否则访问数组元素；当 while 条件不满足时，跳转到 exit；否则执行 block 的 TAC 语句，并无条件跳转到 check.

需要特别注意 whileExpr 为空的判断。

支持 break 语句，用 loopExits.push(exit)和 loopExits.pop()包裹 foreachstmt.block 的处理程序段。

TAC 程序：

```
    _T0 = 0                 ; i
    _T1 = *(E.val – 4)      ; length of array
    _T2 = E.val             ; array pointer
    _T3 = 4                 ; length of a unit
_L0:
    _T6 = (_T0 < _T1)
    if (_T6 == 0) branch(_L1)
    _T4 = (_T0 * _T3)       ; offset
    _T5 = *(_T2 + _T4)      ;x
    _T8 = 1
    _T0 = (_T0 + _T8)
    _T7 = B.TAC
    if (_T7 == 0) branch(_L1)
    block.TAC
    branch(_L0)
_L1:
```

**8. 增加除零错误检查**

（1）修改 RuntimeError.java，增加静态属性字符串，表示除零错误。

（2）修改 Translater.java，修改 genDiv 方法和 genMod 方法，增加被除数是否为 0 的判断，

如果被除数为 0 则报错并退出程序，否则跳转到正常执行除法/求模运算。

**9. 新增测试用例**

9.1 weiyx_virtual_function.decaf

【测试目的】

测试虚函数的动态绑定，加深对于面向对象工作机制的理解。

【正确输出】

Son

【decaf 代码】

```
class Father
{
    int height;
    void print()
    {
        Print("Father\n");
    }
}

class Son extends Father
{
    int age;
    void print()
    {
        Print("Son\n");
    }
}

class Main
{
    static void main()
    {
        class Father obj;
        obj = new Son();
        obj.print();
    }
}
```

9.2 q1-scopy-test2.decaf

【测试目标】

测试 scopy 对象浅复制对于基本类型属性的作用

【正确输出】

1122

3144

【decaf 代码】

```
class A {
```

```
    int a;
    void seta(int a){
        this.a = a;
    }
    int getA(){
        return a;
    }
}

class Father {
    int field;
    class A a;
    void setfield(int f) {
        this.field = f;
        this.a = new A();
    }
    void setOnlyfiled(int f) {
        this.field = f;
    }
    int getfield()
    {
        return this.field;
    }
    void seta(int a) {
        this.a.seta(a);
    }
    int getA() {
        this.a.getA();
    }
}

class Main
{
    static void main()
    {
        class Father a;
        class Father b;
        b = new Father();
        b.setfield(1);
        b.seta(2);
        scopy(a,b);
        Print(b.getfield());
        Print(a.getfield());
        Print(b.getA());
```

```
            Print(a.getA());
            Print("\n");
            b.seta(4);
            b.setOnlyfiled(3);
            Print(b.getfield());
            Print(a.getfield());
            Print(b.getA());
            Print(a.getA());
        }
}
```