**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());

}

}