编译原理PA3实验报告

提前上学2018 李嘉图

2018 年 12 月 14 日

1 主要任务

PA3的主要任务是在完成语义分析后，根据程序生成三地址码，并加入运行时错误检测。

2 具体流程

对于一种新增特性中间代码生成的支持，如果涉及类大小的计算，需要在translate/TransPass1.java 中新增计算；并在translate/TransPass2.java 中增加生成中间代码的Visitor函数，以生成对应的中间代码。

3 实验操作

3.1 类的浅复制的支持

在Decaf中，整数是直接利用一个4 Bytes大小的变量存储的，而所有对象都是利用一个4 Bytes大小的变量存储一个地址。

普通的对象赋值A = B会直接将A的位置设为B 的位置。也就是说，A和B共享同一块内存，一旦一个中的成员变量发生改变，另一个也随之改变。而浅复制scopy(A, B)则是“拉出一层”，将A指向一个新建的对象，并将其所有成员变量赋值为B的成员变量。在底层实现上，可以看作一个内存复制。由于需要访问变量B指向的内存，因此需要在前面的语义分析趟加入一些支持。

在tree/Tree.java中的类LValue/Ident中加入一个成员变量Class classInfo，表示这个标识符指向类的位置。在typecheck/TypeCheck.java中

的函数visitIdent中加入对指向类位置的查询。即当语义分析指出这个标识符是类的时候，将其classInfo设为对应的Variable。

这样我们就可以在代码生成时知道一个对象的信息。在translate/TransPass2.java中新增函数visitScopy，给左边对象使用tr.genDirectCall(classInfo.getNewFuncLabel, INT)分配空间，并将右侧对象的内存复制过去。为了兼顾性能和生成程序大小，当对象空间较小时直接用for循环生成多个指令利用tr.genStore()复制；当对象空间较大时生成一个赋值的循环结构。

3.2 sealed的支持

sealed关键字并没有任何运行时行为，不需要额外支持。

3.3 串行条件卫士语句的支持

依如下算法翻译串行条件卫士语句：

```
1 for each IfSubStmt ifsub in guardedStmt
2 begin
3     translate expr
4     beqz expr exit
5     translate stmt
6     exit :
7 end
```

3.4 支持简单的类型推导

var x只能出现在赋值语句的左边，在translate/TransPass2.java的visitAssign函数中增加对于var x支持，即生成一个从右侧表达式向左侧对应symbol的赋值操作即可。

3.5 支持数组操作

3.5.1 数组初始化表达式

在translate/TransPass2.java的函数visitBinary中增加相应的部分：依如下算法翻译数组初始化表达式:

```
checkArrayInitIndex(expr)
if type of array is not class then
    genNewArray(expr, repeatTime)
else
    genNewArrayClass(expr, repeatTime)
```

其中`genNewArray`已经有实现，只需要实现`genNewArrayClass`即可。这里类的对象和`BaseType`需要分类讨论的原因是对象在`Decaf`中只存储地址，如果直接产生数组会产生一个指向同一内存位置的数组，不能做到生成的每个元素独立。正确的操作是，生成一个循环语句，为每个位置的对象分配新的内存空间，并赋初值。具体实现形如：

```
genNewArrayClass(length, source):
    size = length * WORD + WORD
    obj = NEWARRAY(length)
:loop
    size = size - WORD
    beqz size exit
    obj = obj + WORD
    cur = NEWCLASS(source)
    scopy(cur, source)
    store cur obj
    branch loop
:exit
```

3.5.2 数组下标动态访问表达式

数组是不是合法下标有两种情况——负数下标和超出数组长度，只有两者都不满足才是合法下标，因此在实现上应当利用控制流做逻辑运算符短路处理。在`translate/TransPass2.java`中的函数`visitArrayDefault`中增加代码，实现方法如下：

```
    translate(array)
    translate(index)
    cond = index < length
```

```
    beqz cond def
    cond = index < 0
    bnez cond def
    array = array + WORD
    val = *array
:def
    translate(default)
    val = *default
:exit
```

3.5.3 数组迭代语句

数组迭代语句有两种形式，第一种是给定迭代变量类型的，第二种是自动推导类型的。自动推导类型的工作在语义分析期已经完成，在翻译中间代码时可以统一考虑。

需要特殊考虑的是对于`break`语句的支持，具体方法是用一个栈存储当前位于开循环体内的退出标记，在进入循环体时入栈，在退出循环体时退栈。

在`translate/TransPass2.java`的函数`visitForeachStmt`中增加代码，翻译方法如下：

```
i = 0
length = *(array - WORD)
:loop
    cond = i < length
    beqz cond exit
    var = *array
    translate(condition)
    beqz condition.val  exit
    ExitStack.push exit
    translate(block)
    i = i + 1
    array = array + WORD
    branch loop
```

```
14      ExitStack.pop
15  : exit
```