编译原理 PA3

计 71 张程远 2017011429

前期工作

我一直选择的是分阶段框架，因此首先需要将 PA2 的修改工作应用到 PA3 框架中，具体而言就是按照指导书中的指示做，先直接复制粘贴覆盖掉一部分 PA3 的文件，再手动比对修改一些文件，最后得到 PA3 的框架。

除 0 报错

仿照数组越界的报错, 在 TacEmitter 里加入函数 emitDivError, 只在 Op 为 div 或者 mod 时被调用，检查传入的 Temp value 是否为 0 即可。

Abstract

在 Tacgen 中加一句判断：如果遍历到的函数是一个抽象函数，那么就不访问函数的 body。

Var 和局部类型推导

由于我采用了一个 bool isvar 判断 var 是否存在初始化，对于两种 var 我用了不同名字的变量来处理，因此同样要在 TacEmitter 的 visitLocalVadrDef 里做讨论。

Lambda 和函数类型

这里我采用了与实验指导书不同的想法来考虑这件事。通过观察我们可以想到, 所谓“以方法名直接当作函数使用”的 Call（也就是 VarSel），实际上可以转化为 Lambda 表达式的形式。我们考虑实验指导书上给出的 b.getf(Params), 它实际上就是一个 function, 做了 b.getf() 的操作，也就是 fun(Params){ b.getf(Params); }。静态 call 也可以看成类似的操作。这样做的好处在于，我们不用再做任何关于扩展 Call 的设计，只需要设计好 Lambda 表达式的操作，并且把这些扩展 Call 包装成 Lambda 就可以了。具体地，如果我们发现一个 VarSel 是函数类型，那么我们将包装其为一个 Lambda，其中 fun 所需要的参数列表为 VarSel 本身所接受的参数列表，返回使用相同参数列表调用这个 VarSel 的结果。这将在为我们构造的 Lambda 捕获变量这一环节带来便利。

接下来我们考虑 Lambda 表达式捕获的变量。这里并没有“精确地”捕获 Lambda 表达式的变量，而是考虑所有 Lambda 可能捕获的变量，将它们全部加入到 Lambda 中定义的

CapturedList 里面。这实际上有点偷懒，最后开辟的内存空间也会变多，但对于通过本次 PA 来说可谓简便而有效。为了实现这一点，我在每个 visitLambda 函数的末尾（在 LambdaScope 关闭前）做这样的操作，将捕获到的所有东西加到 LambdaSymbol 里的列表里。到这里，接下来只需要考虑如何处理 Lambda 就可以了。

对于处理 Lambda，我考虑的方法是类比于成员函数的处理模式。这个“成员函数”的参数列表应该为 Lambda 所需传入的参数与 Lambda 捕获变量的集合之并。这样我们就可以为它构建 Entry，相应虚表里就出现了 Lambda 的相关内容。然后在 TacEmitter 里加入函数 visitLambda，作用为将 Lambda 的 entry 和捕获到的变量存入内存。与此同时，我们还保存了 lambda 的参数个数，以便在调用时使用。最后，在调用 Lambda 的时候，我们首先获取 Entry 和参数个数，然后遍历 lambda 的参数并传入即可。

踩过的坑

这次的代码框架，lowlevel 里加入了比较多的内容，其实是不太好阅读的——因为很多底层代码乍一看上去不知道有什么用，或者不知道为什么要加一个这样的东西。好在 TacEmitter 部分的代码给了不少使用的示例，模仿加完特性 0 以后再回过头看代码会舒服很多。

由于这次没有按照推荐的步骤来，所以修改时还是遇到了很多的错误，比如如何判断 this 是否是一个参数，如何安排参数传入的顺序以保证我能正确访问，Capture 函数遍历的作用域是否正确，等等。调这些 Bug 的时候我打了很多很多 Log.info 来获得信息，花费了不少时间。不过由于总体思路是清晰的，对于思路的每一个环节，如果发生错误就从头捋一遍，再配合调试信息，往往就能找到修改的方法。

有一个令我印象特别深的点，是在测试时遇到了在 Simulator.java 里报的 NullPointer 错误。Simulator 是十分底层的代码，即便使用 Log.info 输出调试信息也很难定位到真实的错误位置。关于这一点，我询问了其他同学以后发现没有人遇到过类似的问题，于是我相信这应该是个愚蠢的错误，便回头仔细检查导致这个错误产生的代码块，发现了很明显的 bug。这件事说明 debug 的时候不一定要在报错处查看错误附近的信息，也可以回头检查自己新加的代码，都能较为容易地检查出 Bug。