编译原理 MiniDecaf 编译器实验
Step 11 实验报告

李祥泽
2018011331
lixiangz18@mails.tsinghua.edu.cn

实验内容

实现指针

参考助教的实现, 完成了比较简单的版本. 即, 记录指针类型对原变量取地址的级数. 对于 MiniDecaf 来说, "原变量" 只可能是 `int`, 因而指针也只有一种, 并且也无需保存原变量的类型.

实现左值和类型检查

参考助教的实现, 在 `MainVisitor` 中加入 `checkType` 等工具方法, 在 `Type` 中加入左右值的记录信息和互相转换的方法.

这些方法可以判断一个通过分析语法树得到的类型是否是我们所需的类型, 以及其左右值是否满足要求. 在当前为左值而需要右值时, 可以再对左值取其右值.

具体地说, 一个左值在运算栈中体现为一个地址. 而右值为 (类似立即数的) 值. 那么, 对左值求右值就是将地址所存储的值取出, 即 IR 的一条 `load` 指令.

在前面的步骤中, 当访问变量时, 都是直接将变量的值 (先通过 IR 指令 `xxxaddr` 取地址, 然后 `load` 从地址中取值) 读入运算栈. 现在, 由于直接的变量是左值, 我们需要的是其地址, 因而去掉后面的 `load`. 在使用其值时, 变量会变为右值, 这时再附加上所需的 `load`.

对于这一部分, 助教的参考实现是不够严谨的.

见于参考实现 md-xxy 分支 (Java-ANTLR) 的 commit 5384b949 (step 11), `MainVisitor.java` 的 365 到 379 行.

对于赋值表达式, 助教将被赋值变量的**左值**, 即变量的地址, 作为表达式的值; 而按语法规范, 这里应该得到**右值**, 即变量的新值.

如果按助教的实现方式, 形如 `(a = 1) = 2` 的语句 *在左右值意义上* 是合法的 (但这种语句仍然会因为不能通过语法分析而失败, 所以不会引发实际的问题, 只能说不够严谨), 并且将对变量 `a` 两次赋值 . 这不符合 C 的语法.

思考题

为什么类型检查要放到名称解析之后?

因为: 只有在完成了名称解析之后, 才能完全确定一个符号被如何定义, 如何使用. 这才有可能判断定义的方式与使用的方式是否冲突. 否则, 在不完整的信息下, 无法确保在当前未解析的部分没有错误 (反之, 直接指出已经分析的部分是有错误的**不需要**完整的名称解析).

MiniDecaf 中规定一个值只能有一种类型, 但在很多语言中并非如此, 举出反例

如 Python, 其对象的类型是可变的. 如果在 Python 中定义函数时含有参数, 而参数没有限定类型, 那么就可以传入任意类型的对象.

如果要实现指针大小比较需要注意什么问题? 可以和原来整数比较的方法一样吗?

指针比较大小实际上就是地址比较大小. 因此, 从语义上说, 只应该允许指向同一数组或类似数据结构的两个指针进行大小比较. 在这种情况下, 相当于比较两者谁的下标较小. 如果是两个不相干的指针比较大小一般来说是没有意义的.

从 (通过了语义检查之后的) 实现来说, 指针比较和整数比较没有区别.

Honor Code

主要参考实验指导书实现. 参考了助教的示例代码.