编译原理 MiniDecaf 编译器实验
Step 4 实验报告

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

说明: 本 Step 是与 Step 3 一并实现的.

实验内容

实现二元逻辑运算符 `&&`, `||` 和二元比较运算符 `==` 等

依照实验指导书中提示的二元运算语法, 编写 g4 文件, 并在对应的 Visitor 中实现 IR 生成和 IR 到汇编的生成.

指导书中给出的上下文无关语法已经很好地考虑了运算符优先级与结合性, 因此直接采用了 (修改了一些名字以符合个人代码习惯).

思考题

在表达式计算时, 对于某一步运算, 是否一定要先计算出所有的操作数的结果才能进行运算?

不是. 例如, 在支持短路求值的逻辑运算中, 如果第一操作数出现了短路情况, 则第二操作数就不必计算.

在 MiniDecaf 中, 我们对于短路求值未做要求, 但在包括 C 语言的大多数流行的语言中, 短路求值都是被支持的. 为何这一特性广受欢迎? 你认为短路求值这一特性会给程序员带来怎样的好处?

考虑以下情况: 一个 C 函数接受一个指针参数, 如果该指针为空或所指变量不符合某些限制则直接返回. 如果支持短路求值, 只需要写成 `if (ptr==NULL || notValid(*ptr)) return;` 就可以了; 如果指针为空, 后面的检查根本不会执行. 反之, 为了确保不访问空指针, 就必须拆开成两个 if 语句 `if (ptr==NULL) return; if (notValid(*ptr)) return;`

对于其他语言, 虽然不一定有指针, 但总存在可能引发异常的类似检查. 通过短路求值, 可以简洁地消除掉这种异常.

Honor Code

主要参考实验指导书实现.