Principles of Compilers
Experiment: Stage-2

Chenghua Liu*
liuch18@mails.tsinghua.edu.cn
Department of Computer Science
Tsinghua University

目录

*刘程华，学号 2018011687

1 实验内容及过程

1.1 step5

step5 要增加变量了，包括变量的声明和变量的使用（读取/赋值）。

通过所给例子学习，语法词法分析部分不需要修改，然后是语义分析部分，我们需要构建符号表部分的内容，在/frontend/typecheck/name.py 中，我们根据注释所写补充以下函数：visitDeclaration、visitAssignment、visitIdentifier。类型检查部分，不需要修改。生成三地址码部分，我们在/frontend/tacgen/tacgen.py 中根据注释所写补充以下函数：visitDeclaration、visitAssignment、visitIdentifier。然后是中间代码生成部分，我们在/backend/riscv/riscvasmemitter.py 中的 RiscvInstrSelector 类添加函数 visitAssign。

1.2 step6

step6 我们要支持 if 语句和条件表达式（又称三元/三目表达式，ternary expression）。和上一 step 流程一样，经过分析我们需要修改以下两部分。一是语义分析部分，在/frontend/typecheck/name.py 中根据注释所写补充函数 visitCondExpr；二是生成三地址码部分，我们在/frontend/tacgen/tacgen.py 中根据注释所写补充以下函数 visitCondExpr。

2 思考题

2.1 step5

1. 我们假定当前栈帧的栈顶地址存储在 sp 寄存器中，请写出一段 risc-v 汇编代码，将栈帧空间扩大 16 字节。（提示 1：栈帧由高地址向低地址延伸；提示 2：risc-v 汇编中 addi reg0, reg1, < 立即数 > 表示将 reg1 的值加上立即数存储到 reg0 中。）

   答：

   ```
   addi sp, sp -16
   ```

2. 有些语言允许在同一个作用域中多次定义同名的变量，例如这是一段合法的 Rust 代码（你不需要精确了解它的含义，大致理解即可）：

   ```
   fn main() {
           let a = 0;
           let a = f(a);
           let a = g(a);
   }
   ```

   其中 f(a) 中的 a 是上一行的 let a = 0; 定义的，g(a) 中的 a 是上一行的 let a = f(a);。

如果 MiniDecaf 也允许多次定义同名变量，并规定新的定义会覆盖之前的同名定义，请问在你的实现中，需要对定义变量和查找变量的逻辑做怎样的修改？（提示：如何区分一个作用域中不同位置的变量定义？）

答：

在定义变量时仍然做符号名存在性检查。如果不存在则照常。如果存在，不报错而是进下一步计算初始化表达式 (在初始化表达式中仍可能调用这个变量所以暂时还不能覆盖)，计算完成后将结果存入该变量原先占有的内存空间 (相当于一次赋值)。而查找变量的逻辑无需修改。

2.2 step6

1. 你使用语言的框架里是如何处理悬吊 else 问题的？请简要描述。

   答：

   我使用的是 python 框架，它区分 statement_matched 和 statement unmatched，if 与 else 匹配的时 if 后面的语句只能是 matched，没有匹配 else 的 if 只能由 unmatched 生成。换句话说，python 框架通过禁止在 if 和 else 之间插入落单的 if 来处理悬吊 else 问题。

2. 在实验要求的语义规范中，条件表达式存在短路现象。即：

   ```
   int main() {
           int a = 0;
           int b = 1 ? 1 : (a = 2);
           return a;
   }
   ```

   会返回 0 而不是 2。如果要求条件表达式不短路，在你的实现中该做何种修改？简述你的思路。

   答：

   生成三地址码阶段，我们先对两个子表达式递归深入获取其两者的结果，然后后再根据条件选择将哪一个赋值为左侧。