Stage-2 实验报告

计03 王文琦 2020010915

实验内容

Step 5

`Step 5` 整体比较清晰，主要是引入了局部变量的声明和赋值。

- 首先既然是局部变量的声明和赋值，我们就需要在类型检查扫描的时候，对无效的声明（例如重复声明变量）和无效的赋值（例如对未声明的变量赋值）进行检查并且报错。这一部分主要需要修改 `namer.py` 中对于声明、赋值和标识符的 `visit` 函数。本次我们需要使用 `varsymbol` 来给标识符、赋值表达式的左值挂载一个临时变量，这样在访问这个节点的时候，就可以直接返回该临时变量的值。
- 然后我们要在 `tacgen.py` 文件中执行类似的操作，但是目的不是为了类型检查，而是为了生成三地址码。同时在这里我们需要将上一步操作中节点上的 `varsymbol` 的值挂载上 `temp` 临时变量。

Step 6

`Step 6` 主要是引入三目运算符，包括 `If` 语句和条件表达式。其中条件表达式 `A ? B : C` 在一定程度上可以类比是 `If` 语句中的 `if A then B else C` 。

- 在 `namer.py` 中调用 `visitCondExpr` ，然后顺次访问 `A B C` 三个节点就可以了，子节点的类型检查函数已经写好了。
- 紧接着是在 `tacgen.py` 中为三目运算符生成三地址码。注意这里需要用到条件跳转，所以需要引入 `Label` 变量，条件跳转到指令对应的行号。这一部分可以仿照 `visitIf` 函数执行。注意三目运算也是有返回值的，所以同样需要分配临时变量。

思考题

1. **Step 5:** 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。）

   ```
   addi    rsp, rsp, -16
   ```

2. **Step 5:** 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);
   }
   ```

我觉得可以这样修改：首先在符号表中查找有没有已经声明的符号，如果没有的话，那么和之前的规则一致，进行一次合法的声明，并且如果有右值的话，将右值赋值给符号变量；**如果有的话，**那么先判断其他情况是否是合法的（比如右值表达式是否是合法的），如果是的话，不更改符号表的符号，但是将该符号中的临时变量的值更改为最新的赋值。

3. **Step 6:** 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。

   我使用的是 Python 语言框架， python 框架的语块之间的逻辑从属关系是使用缩进来区分的。所以 `else` 会和同一个缩进等级下的 `if` 匹配；在同一缩进登记下，和最相邻的上文中未匹配的 `if` 进行匹配。

4. 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。

   整体思路就是只将设置三元表达式返回值的部分放到条件跳转语句块中，换言之，就是先计算出两个分支的返回值(执行 `visit` 函数)，并且用两个临时变量保存，然后最后再通过条件判断将两个临时变量值中的一个赋给表达式。