stage-3

学号：2021012806

姓名：尤梓锐

step6

实验内容

作用域

首先自定义了作用域类型栈的数据类型ScopeStack：

```
 class ScopeStack:
-    pass
\ No newline at end of file
+    def __init__(self) -> None:
+        self.stack = []
+
+    def is_empty(self) -> bool:
+        if len(self.stack) == 0:
+            return True
+        else:
+            return False
+
+    def size(self) -> int:
+        return len(self.stack)
+
+    def push(self, scope: Scope) -> None:
+        self.stack.append(scope)
+
+    def pop(self) -> Optional[Scope]:
+        if self.is_empty():
+            return None
+        else:
+            return self.stack.pop()
+
+    # def top(self) -> Optional[Scope]:
+    #     if self.is_empty():
+    #         return None
+    #     else:
+    #         return self.stack[-1]
+
+    def __iter__(self):
+        self.count = 0
+        return self
+
+    def __next__(self):
+        if self.count + self.size() > 0:
+            self.count = self.count - 1
+            return self.stack[self.count]
+        else:
+            raise StopIteration
```

在Namer初始化时中加入scopestack：

```
+class Namer(Visitor[ScopeStack, None]):
     def __init__(self) -> None:
+        self.scopestack = ScopeStack()
         pass
```

在transform阶段加入应该要位于栈底的全局变量层：

```
ctx = Scope(program.globalScope)

self.scopestack.push(ctx)
program.accept(self, ctx)
self.scopestack.pop()
return program
```

每一个block对应一个新的作用域层次，因此在每次访问一个block时新建一个ctx并将其压入栈中，访问block结束后scopestack弹栈：

```
def visitBlock(self, block: Block, ctx: Scope) -> None:
    ctx = Scope(ScopeKind.LOCAL)
    self.scopestack.push(ctx)
    for child in block:
        child.accept(self, ctx)
        # print(type(child))
        # print(self.scopestack.size())
        # print(ctx)
    self.scopestack.pop()
```

有了作用域之后，每次访问identifier就需要考虑其可能不定义在当前局部的作用域，而在更外层的作用域，即检查该符号是否被定义过需要对栈进行遍历，找到最近的含有该符号的ctx。由于本step中没有全局变量相关的内容，检查到全局作用域的时候报错：

```
if ctx.lookup(ident.value) == None:
    raise DecafUndefinedVarError(ident.value)

ident.setattr("symbol", ctx.get(ident.value))
for context in self.scopestack:
    if bool(context.lookup(ident.value)):
        ident.setattr("symbol", context.get(ident.value))
        break
    elif context.kind == ScopeKind.GLOBAL:
        raise DecafUndefinedVarError
```

数据流

在后端，对于每个tac的block（根据分支、返回等指令对tac进行分割，每一段完全顺序执行无跳转的代码块是一个block），数据流分析部分的任务是检查是否存在不会运行到的block，如果有，那么在生成riscv代码的时候就可以将这部分省略。

具体的方法就是对控制流图（每个block指向自己可能跳转到的下一个block）进行DFS，统计遍历到的block节点，将没统计到的节点忽略：

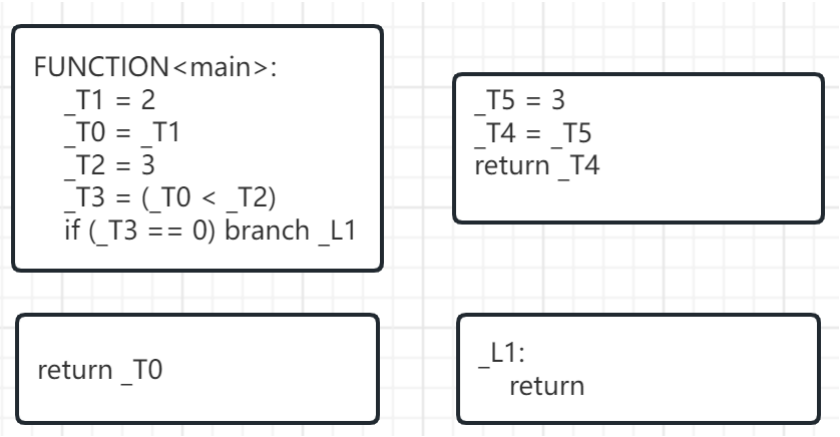
```
  stack = [0]
  touch = [0]
  while len(stack) > 0:
      node_index = stack.pop()
      succ_indices = self.links[node_index][1]
      for succ_index in succ_indices:
          if succ_index not in touch:
              touch.append(succ_index)
              stack.append(succ_index)

  self.nodes = []
  for node_index in touch:
      self.nodes.append(nodes[node_index])
```

思考题

该程序的tac代码如下：

```
FUNCTION<main>:
    _T1 = 2
    _T0 = _T1
    _T2 = 3
    _T3 = (_T0 < _T2)
    if (_T3 == 0) branch _L1
    _T5 = 3
    _T4 = _T5
    return _T4
    return _T0
_L1:
    return
```

对其进行block切分如下：



构建控制流图如下：