STAGE 3

计13 占海川 2021050009

实验内容

作用域栈

`frontend/scope/scopestack.py`

- 新建`ScopeStack`类

  ```
  class ScopeStack:
      def __init__(self) -> None:
          // 作用域栈
          self.scopes = []

      // 将扫描到的作用域入栈
      def push(self, scope: Scope) -> None:
          self.scopes.append(scope)

      // 栈顶作用域出栈
      def pop(self) -> None:
          self.scopes.pop()

      // 返回栈顶作用域
      def top(self) -> Scope:
          return self.scopes[-1]

      // 遍历作用域栈，检查符号是否先前声明过
      def lookup(self, name: str) -> Optional[Symbol]:
          for scope in self.scopes[::-1]:
              if scope.containsKey(name):
                  return scope.get(name)
          return None
  ```

符号表构建

`frontend/typecheck/typer.py`

- 将上下文信息`ctx`改为作用域栈类

`frontend/typecheck/namer.py`

- 将上下文信息 `ctx` 改为作用域栈类
- `Block`

  ```
  def visitBlock(self, block: Block, ctx: ScopeStack) -> None:
      # 新建一个局部作用域并入栈
      ctx.push(Scope(ScopeKind.LOCAL))
      for child in block:
          child.accept(self, ctx)
      # 出栈
      ctx.pop()
  ```

- `Declaration`

  ```
  def visitDeclaration(self, decl: Declaration, ctx: ScopeStack) -> None:
      // 检查当前作用域是否声明过该符号
      if ctx.top().lookup(decl.ident.value) == None:
          var = VarSymbol(decl.ident.value, decl.var_t.type)
          ctx.top().declare(var)
          decl.setattr("symbol", var)
          if decl.init_expr != NULL:
              decl.init_expr.accept(self, ctx)
      else:
          raise DecafDeclConflictError(str(decl.ident.value))
  ```

- `Identifier`

```
 def visitIdentifier(self, ident: Identifier, ctx: ScopeStack) -> None:
     // 检查全作用域内是否声明过符号
     if ctx.lookup(ident.value) == None:
         raise DecafUndefinedVarError(str(ident.value))
     ident.setattr("symbol", ctx.lookup(ident.value))
```

寄存器分配

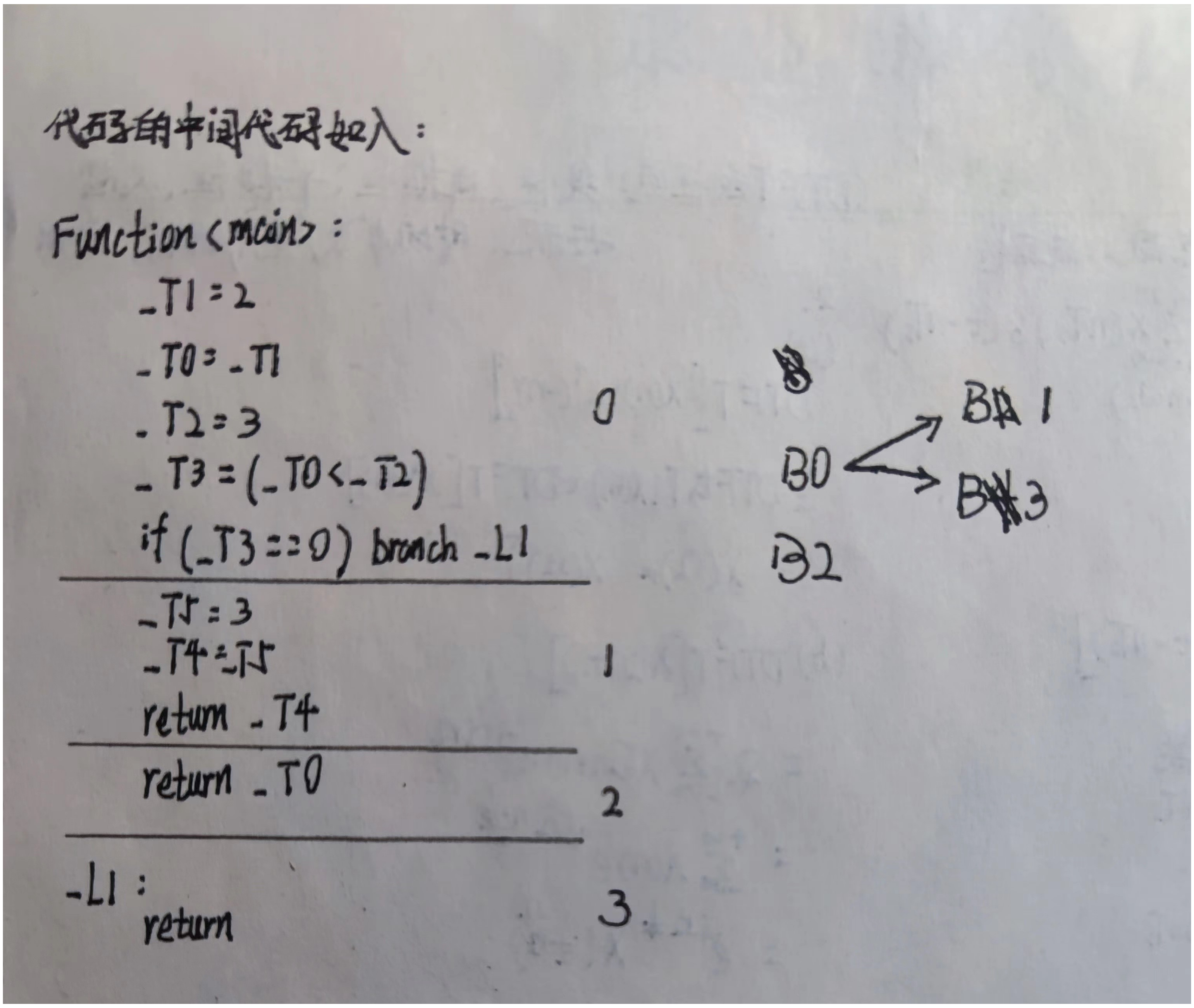
`backend/dataflow/cfg.py`

- 深度遍历邻接表, 找到可达节点

```
stack = []
self.reachable = []
stack.append(0)
while stack:
    top = stack.pop()
    self.reachable.append(top)
    for node in self.links[top][1]:
        if node not in self.reachable:
            stack.append(node)
```

```
def iterator(self):
    reachableNodes = []
    for n in self.reachable:
        reachableNodes.append(self.nodes[n])
    return iter(reachableNodes)
```

思考题



代码的中间代码如下：

```
Function <main> :
        _T1 = 2
        _T0 = _T1
        _T2 = 3                               0
        _T3 = (_T0 < _T2)
        if (_T3 == 0) branch _L1
-----------------------------------------
        _T5 = 3
        _T4 = _T5                             1
        return _T4
-----------------------------------------
        return _T0
                                              2
-----------------------------------------
_L1 :
        return                                3
```

$$\begin{tikzcd}
 & B1 \\
B0 \arrow[ur] \arrow[dr] & \\
 & B3 \\
B2 &
\end{tikzcd}$$