Stage3 实验报告

姓名：丁俊辉

学号：2021011145

step6

实验过程

在 `scopestack.py` 中，我完成了 `ScopeStack` 类型的定义：

```
class ScopeStack:
    def __init__(self, globalscope: Scope) -> None:
        self.globalscope = globalscope
        self.stack: list[Scope] = [globalscope]

    def push(self, scope: Scope) -> None:
        self.stack.append(scope)

    def pop(self) -> Scope:
        return self.stack.pop()

    def top(self) -> Scope:
        return self.stack[-1]

    def lookup(self, name: str) -> Optional[Symbol]:
        for scope in reversed(self.stack):
            symbol = scope.lookup(name)
            if symbol is not None:
                return symbol
        return None
```

在 `namer.py` 和 `typer.py` 中，我将类型定义中的 `Scpoe` 更改成了 `ScopeStack`，在类的方法中将 `ctx` 变更成 `ScopeStack` 类型，并且加入了相应的控制逻辑（入栈、出栈、符号查找等）：

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

    # Entry of this phase
    def transform(self, program: Program) -> Program:
        # Global scope. You don't have to consider it until Step 6.
        program.globalScope = GlobalScope
        ctx: ScopeStack = ScopeStack(program.globalScope)

        program.accept(self, ctx)
        return program

    ...

    def visitBlock(self, block: Block, ctx: ScopeStack) -> None:
        ctx.push(Scope(ScopeKind.LOCAL))
```

```python
        for child in block:
            child.accept(self, ctx)
        ctx.pop()

    ...

    def visitDeclaration(self, decl: Declaration, ctx: ScopeStack) -> None:
        if ctx.top().lookup(decl.ident.value) is not None:
            raise DecafDeclConflictError(decl.ident.name)
        symbol = VarSymbol(decl.ident.value, decl.var_t.type)
        ctx.top().declare(symbol)
        decl.setattr("symbol", symbol)
        if decl.init_expr is not None:
            decl.init_expr.accept(self, ctx)

    ...



class Typer(Visitor[ScopeStack, None]):
    def __init__(self) -> None:
        pass

    # Entry of this phase
    def transform(self, program: Program) -> Program:
        return program
```

在 `cfg.py` 中，我通过 DFS 完成了找不可达块的逻辑：

```python
class CFG:
    def __init__(self, nodes: list[BasicBlock], edges: list[(int, int)]) -> None:
        self.nodes = nodes
        self.edges = edges

        self.links = []

        for i in range(len(nodes)):
            self.links.append((set(), set()))

        for (u, v) in edges:
            self.links[u][1].add(v)
            self.links[v][0].add(u)

        """
        You can start from basic block 0 and do a DFS traversal of the CFG
        to find all the reachable basic blocks.
        """
        self.reachable = self.computeReachability()

    def computeReachability(self):
        reachable = [False] * len(self.nodes)

        def dfs(node_id):
            if reachable[node_id]:
                return
```

```
            reachable[node_id] = True
            for succ in self.getSucc(node_id):
                dfs(succ)

        dfs(0)
        return reachable

    def isReachable(self, id):
        return self.reachable[id]
```

将此应用到 `BruteRegAlloc` 中，见 `bruteregalloc.py`：

```
class BruteRegAlloc(RegAlloc):
    def __init__(self, emitter: RiscvAsmEmitter) -> None:
        super().__init__(emitter)
        self.bindings = {}
        for reg in emitter.allocatableRegs:
            reg.used = False

    def accept(self, graph: CFG, info: SubroutineInfo) -> None:
        subEmitter = RiscvSubroutineEmitter(self.emitter, info)
        for bb in graph.iterator():
            # you need to think more here
            # maybe we don't need to alloc regs for all the basic blocks
            if not graph.isReachable(bb.id):
                continue
            if bb.label is not None:
                subEmitter.emitLabel(bb.label)
            self.localAlloc(bb, subEmitter)
        subEmitter.emitFunc()

        ...
```

思考题

1. 请画出下面 MiniDecaf 代码的控制流图。

```
int main(){
 int a = 2;
 if (a < 3) {
     {
         int a = 3;
         return a;
     }
     return a;
 }
}
```

A：

在 `TacBinaryOp` 中加入 `AND` 可以编译出如下的 tac 码：

```
(minidecaf) 2021011145@compile:~/minidecaf-2021011145$ python3 main.py --input
example.c --tac
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
```

由 tac 码，可以分出以下四个基本块：

```
<B0>
FUNCTION<main>:
    _T1 = 2
    _T0 = _T1
    _T2 = 3
    _T3 = (_T0 < _T2)
    if (_T3 == 0) branch _L1
</B0>
<B1>
    _T5 = 3
    _T4 = _T5
    return _T4
</B1>
<B2>
    return _T0
</B2>
<B3>
_L1:
    return
</B3>
```

根据基本块的逻辑关系，有如下的控制流图：