刘雅迪

计26

学号：2021010521

step6: 作用域和块语句

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;
 }
}
```

```
    [Start]
       |
       v
  [Basic Block 1](int a = 2;)
       |
       v
  [Basic Block 2](if (a < 3)进行条件判断)
       |
       +---(True)---> [Basic Block 3] (return a = 3)
       |
       +---(False)---> [Basic Block 4] (return a = 2)
```

实验内容

frontend

1.

`frontend/scope/scopestack.py`：

设计了一个Scope的作用域栈，栈底是全局作用域，其他的是局部作用域。

将原来`Scope`类的`declare`函数、`isGlobalScope`函数以及`lookup`函数的范围扩大到整个作用域栈。

同时增加`currentScope`函数获取当前的Scope，即栈顶的Scope；

`addScope`函数：像栈内加入一个Scope；

`popScope`函数：弹出栈顶的Scope；

`isConflict`函数：查看当前的Scope（即栈顶的Scope）中是否有重复的名称。

```
class ScopeStack:
    defaultstackdepth = 256
    def __init__(self, globalscope: Scope, stackdepth: int=defaultstackdepth):
        self.globalscope = globalscope
        self.stack = [globalscope]
        self.stackdepth = stackdepth
        self.loopdepth = 0

    #得到当前的Scope，即栈顶的Scope
    def currentScope(self):
        if not self.stack: return self.globalscope
        return self.stack[-1]

    #向栈内加入一个Scope
    def addScope(self, scope: Scope) -> None:
        if len(self.stack) < self.stackdepth:
            self.stack.append(scope)
        else:
            raise stackOverflow

    #弹出栈顶的Scope
    def popScope(self):
        self.stack.pop()

    #看当前的Scope中是否有重复的名称
    def isConflict(self, name: str) -> Optional[Symbol]:
        if self.currentScope().containsKey(name):
            return self.currentScope().get(name)
        return None

    # To declare a symbol.
    def declare(self, symbol: Symbol) -> None:
        self.currentScope().declare(symbol)

    # To check if this is a global scope.
    def isGlobalScope(self) -> bool:
        return self.currentScope().isGlobalScope()

    # To get a symbol if declared in the scope
    def lookup(self, name: str) -> Optional[Symbol]:
        s = len(self.stack)
        for d in range(s-1, -1, -1):
            if self.stack[d].containsKey(name):
                return self.stack[d].get(name)
        return None
```

2.

`frontend/typecheck/namer.py`：

将所有的`Scope`改为`ScopeStack`，同时修改`visitFunction`函数和`visitBlock`函数：

`visitFunction`函数：

首先查看名称是否重复，若不重复，则将新的变量压入当前的栈中，即`ctx.currentScope()`

```
def visitFunction(self, func: Function, ctx: ScopeStack) -> None:
    # func.body.accept(self, ctx)
    if ctx.isConflict(func.ident.value):
        raise DecafDeclConflictError
    else:
        newSymbol = FuncSymbol(func.ident.value, func.ret_t.type, ctx.currentScope())
        ctx.declare(newSymbol)
        func.body.accept(self, ctx)
```

`visitBlock`函数：

首先定义一个新的block，然后在作用域栈中将其压入栈中，再accept这个block的child，最后将该block弹出栈。

```
def visitBlock(self, block: Block, ctx: ScopeStack) -> None:
    # for child in block:
    #     child.accept(self, ctx)
    block_scope = Scope(ScopeKind.LOCAL)
    ctx.addScope(block_scope)
    for child in block:
        child.accept(self, ctx)
    ctx.popScope()
```

以及`visitDeclaration`函数开头的使用`ctx.lookup`查看是否存在相同的名称改为使用`ctx.isConflict`查看。

3.

`frontend/typecheck/typer.py`：

将`class Typer(Visitor[Scope, None]):`改为`class Typer(Visitor[ScopeStack, None]):`

backend

1.

`backend/dataflow/cfg.py`：

在CFG类中添加`findAvailableBlock`函数，利用dfs的方式，返回所有可到达的基本块的id的list。

```
def findAvailableBlock(self):
    available_block_id_list = [0]
    v = 0
    for block in self.iterator():
        if v in self.links[block.id][1]:
            available_block_id_list.append(v)
    return available_block_id_list
```

2.

`backend/reg/bruteregalloc.py`：

按照注释修改了`BruteRegAlloc`类的`accept`函数，利用CFG类的`findAvailableBlock`函数，只有可到达的基本块才会分配寄存器。

```
def accept(self, graph: CFG, info: SubroutineInfo) -> None:
    subEmitter = RiscvSubroutineEmitter(self.emitter, info)
    available_block = graph.findAvailableBlock()
    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 bb.id not in available_block: continue
        if bb.label is not None:
            subEmitter.emitLabel(bb.label)
        self.localAlloc(bb, subEmitter)
    subEmitter.emitFunc()
```

Honor Code

没有参考代码和其他资源，没有借鉴同学的代码。

将自己的代码`frontend/typecheck/namer.py`和`frontend/tacgen/tacgen.py`中的`visitDeclaration`函数给毛心怡同学参考过。