刘雅迪

计26

学号：2021010521

step7: 条件语句

思考题

1. 我们的实验框架里是如何处理悬吊 else 问题的？请简要描述。

   答：`statement`语法中有`statement_matched`和`statement_unmatched`两种情况，只有"if"对应`statement_unmatched`语法。所以if else匹配的中间一定是`statement_matched`，这样设置产生式的优先级，使得else只能和同层的if匹配。

2. 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。

   答：将`frontend/tacgen/tacgen.py`中`visitCondExpr`的`expr.then.accept`和`expr.otherwise.accept`移动到`expr.cond.accept`的前面，这样条件表达式一定会访问then和otherwise的部分。

实验内容

1.修正了stage-3关于基础块分配寄存器的bug

`backend/dataflow/cfg.py`函数中修改了查找可到达的基础块函数的bug：

原来把"for v in"手滑写成了"if v in"，怪不得编辑器提醒我v没定义，而我还加了个"v = 0"...

这个bug是因为tac码和riscv码都能正常生成，但是无法编译，报错原因是不能识别“_L1”这个符号发现的

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

2.`frontend/typecheck/namer.py`

模仿`visitBinary`函数完成了`visitCondExpr`函数：

```
def visitCondExpr(self, expr: ConditionExpression, ctx: ScopeStack) -> None:
    """
    1. Refer to the implementation of visitBinary.
    """
    expr.cond.accept(self, ctx)
    expr.then.accept(self, ctx)
    expr.otherwise.accept(self, ctx)
    # raise NotImplementedError
```

3. `frontend/tacgen/tacgen.py`

模仿`visitIf`函数的"else"部分，即if语句没有else分支的情况，同时完成了寄存器的分配。

```
def visitCondExpr(self, expr: ConditionExpression, mv: TACFuncEmitter) -> None:
    """
    1. Refer to the implementation of visitIf and visitBinary.
    """
    expr.cond.accept(self, mv)
    temp = mv.freshTemp()
    skipLabel = mv.freshLabel()
    exitLabel = mv.freshLabel()
    mv.visitCondBranch(
        tacop.CondBranchOp.BEQ, expr.cond.getattr("val"), skipLabel
    )
    expr.then.accept(self, mv)
    mv.visitAssignment(temp, expr.then.getattr("val"))
    mv.visitBranch(exitLabel) #jump
    mv.visitLabel(skipLabel)
    expr.otherwise.accept(self, mv)
    mv.visitAssignment(temp, expr.otherwise.getattr("val"))
    mv.visitLabel(exitLabel)
    expr.setattr("val", temp)
    # raise NotImplementedError
```

step 8: 循环语句

思考题

1. 将循环语句翻译成 IR 有许多可行的翻译方法，例如 while 循环可以有以下两种翻译方式：

   第一种（即实验指导中的翻译方式）：

   - `label BEGINLOOP_LABEL`：开始下一轮迭代
   - `cond 的 IR`
   - `beqz BREAK_LABEL`：条件不满足就终止循环
   - `body 的 IR`
   - `label CONTINUE_LABEL`：continue 跳到这

- `br BEGINLOOP_LABEL`：本轮迭代完成
- `label BREAK_LABEL`：条件不满足，或者 break 语句都会跳到这儿

  第二种：
- `cond 的 IR`
- `beqz BREAK_LABEL`：条件不满足就终止循环
- `label BEGINLOOP_LABEL`：开始下一轮迭代
- `body 的 IR`
- `label CONTINUE_LABEL`：continue 跳到这
- `cond 的 IR`
- `bnez BEGINLOOP_LABEL`：本轮迭代完成，条件满足时进行下一次迭代
- `label BREAK_LABEL`：条件不满足，或者 break 语句都会跳到这儿

  从执行的指令的条数这个角度（`label` 不算做指令，假设循环体至少执行了一次），请评价这两种翻译方式哪一种更好?

  答：第一种方式更好。第一种方式在每轮迭代中进行条件判断，即先执行cond，条件满足则执行body并跳转回开头，条件不满足则循环终止。第二种方式先执行条件判断，满足后再开始下一轮迭代，即执行body与cond，条件满足跳转回开头，条件不满足循环终止。假设循环执行到不满足cond结束，只进行一轮的话两种方式执行的指令条数相等，而进行轮次大于一轮时，以两轮为例，第一种方式执行的指令为：cond, body, br, cond, beqz，而第二种方式执行的指令为：cond, body, cond, bnez, body, cond，所以第一种方式更好。

2. 我们目前的 TAC IR 中条件分支指令采用了单分支目标（标签）的设计，即该指令的操作数中只有一个是标签；如果相应的分支条件不满足，则执行流会继续向下执行。在其它 IR 中存在双目标分支（标签）的条件分支指令，其形式如下：

   ```
   br cond, false_target, true_target
   ```

   其中`cond`是一个临时变量，`false_target`和`true_target`是标签。其语义为：如果`cond`的值为0（假），则跳转到`false_target`处；若`cond`非0（真），则跳转到`true_target`处。它与我们的条件分支指令的区别在于执行流总是会跳转到两个标签中的一个。

   你认为中间表示的哪种条件分支指令设计（单目标 vs 双目标）更合理？为什么？（言之有理即可）

   答：我认为选择双目标的条件分支指令设计更合理。因为双目标更加灵活，符合编程语言的控制流结构，对于if-elif-else的结构双目标的条件分支会更加方便。

实验内容

1.词法分析

在`frontend/lexer/lex.py`的reserved中添加"for"和"continue"：

```
# Reserved keywords
reserved = {
    "return": "Return",
    "int": "Int",
    "if": "If",
    "else": "Else",
    "while": "While",
    "break": "Break",
    "for": "For",
    "continue": "Continue",
}
```

2.语法分析

在`frontend/parser/ply_parser.py`中添加"for"和"continue"的语法，其中注意"for"语法中`init`为`declaration`的情况：

```
def p_continue(p):
    """
    statement_matched : Continue Semi
    """
    p[0] = Continue()

def p_for_1(p):
    """
    statement_matched : For LParen expression Semi expression Semi expression RParen
statement_matched
        | For LParen declaration Semi expression Semi expression RParen statement_matched
    statement_unmatched : For LParen expression Semi expression Semi expression RParen
statement_unmatched
        | For LParen declaration Semi expression Semi expression RParen statement_unmatched
    """
    p[0] = For(p[3], p[5], p[7], p[9])
```

3.生成ast树

在`frontend/ast/tree.py`中添加`For`类和`Continue`类，其中`For`类参考`While`类写，`Continue`类参考`Break`类写即可：

```
class For(Statement):
    """
    AST node of for statement.
    """

    def __init__(self, init: Expression, cond: Expression, update: Expression, body:
Statement) -> None:
        super().__init__("for")
        self.init = init
        self.cond = cond
        self.update = update
        self.body = body
```

```
    def __getitem__(self, key: int) -> Node:
        return (self.init, self.cond, self.update, self.body)[key]

    def __len__(self) -> int:
        return 4

    def accept(self, v: Visitor[T, U], ctx: T):
        return v.visitFor(self, ctx)
```

```
class Continue(Statement):
    """
    AST node of continue statement.
    """

    def __init__(self) -> None:
        super().__init__("continue")

    def __getitem__(self, key: int) -> Node:
        raise _index_len_err(key, self)

    def __len__(self) -> int:
        return 0

    def accept(self, v: Visitor[T, U], ctx: T):
        return v.visitContinue(self, ctx)

    def is_leaf(self):
        return True
```

在`frontend/ast/visitor.py`的`Visitor`类中添加`visitFor`函数和`visitContinue`函数:

```
def visitFor(self, that: For, ctx: T) -> Optional[U]:
    return self.visitOther(that, ctx)

def visitContinue(self, that: Continue, ctx: T) -> Optional[U]:
    return self.visitOther(that, ctx)
```

4.在`ScopeStack`类中添加检查break/continue语句是否在一个循环内的函数

`frontend/scope/scopestack.py`的`ScopeStack`类:

```
#检查 break/continue 语句是否在一个循环内
def openLoop(self) -> None:
    self.loopdepth += 1

def closeLoop(self) -> None:
    self.loopdepth -= 1

def inLoop(self) -> bool:
    return self.loopdepth > 0
```

5. `frontend/typecheck/namer.py`

按照注释写`visitFor`函数：

```
"""
    def visitFor(self, stmt: For, ctx: Scope) -> None:

    1. Open a local scope for stmt.init.
    2. Visit stmt.init, stmt.cond, stmt.update.
    3. Open a loop in ctx (for validity checking of break/continue)
    4. Visit body of the loop.
    5. Close the loop and the local scope.
    """
    def visitFor(self, stmt: For, ctx: ScopeStack) -> None:
        for_scope = Scope(ScopeKind.LOCAL)
        ctx.addScope(for_scope)
        if not stmt.init is NULL: stmt.init.accept(self, ctx)
        if not stmt.cond is NULL: stmt.cond.accept(self, ctx)
        if not stmt.update is NULL: stmt.update.accept(self, ctx)
        ctx.openLoop()
        stmt.body.accept(self, ctx)
        ctx.closeLoop()
        ctx.popScope()
```

按照注释修改`visitWhile`函数并完成`visitBreak`函数和`visitContinue`函数：

```
    def visitWhile(self, stmt: While, ctx: ScopeStack) -> None:
        stmt.cond.accept(self, ctx)
        ctx.openLoop()
        stmt.body.accept(self, ctx)
        ctx.closeLoop()

    def visitBreak(self, stmt: Break, ctx: ScopeStack) -> None:
        """
        You need to check if it is currently within the loop.
        To do this, you may need to check 'visitWhile'.

        if not in a loop:
            raise DecafBreakOutsideLoopError()
        """
        if not ctx.inLoop():
            raise DecafBreakOutsideLoopError()
        # raise NotImplementedError

    """
    def visitContinue(self, stmt: Continue, ctx: Scope) -> None:

    1. Refer to the implementation of visitBreak.
    """
    def visitContinue(self, stmt: Continue, ctx: ScopeStack) -> None:
        if not ctx.inLoop():
            raise DecafBreakOutsideLoopError()
```

6. `frontend/tacgen/tacgen.py`

模仿`visitWhile`函数写`visitFor`函数，其中具体的跳转情况按照实验指导书上的循环语句例子来写：

```
    def visitFor(self, stmt: For, mv: TACFuncEmitter) -> None:
        beginLabel = mv.freshLabel()
        loopLabel = mv.freshLabel()
        breakLabel = mv.freshLabel()
        mv.openLoop(breakLabel, loopLabel)

        stmt.init.accept(self, mv)
        mv.visitLabel(beginLabel)
        stmt.cond.accept(self, mv)
        mv.visitCondBranch(tacop.CondBranchOp.BEQ, stmt.cond.getattr("val"), breakLabel)
        stmt.body.accept(self, mv)

        mv.visitLabel(loopLabel)
        stmt.update.accept(self, mv)

        mv.visitBranch(beginLabel)
        mv.visitLabel(breakLabel)
        mv.closeLoop()
```

模仿`visitBreak`函数写`visitContinue`函数：

```
    def visitContinue(self, stmt: Continue, mv: TACFuncEmitter) -> None:
        mv.visitBranch(mv.getContinueLabel())
```

Honor Code

没有参考代码和其他资源，没有借鉴同学的代码也没有将自己的代码给同学参考过。