Stage-2 实验报告

范逍宇 2019013273

一、工作概述。

Step 5

1.前端。

- 在 `scanner.l` 文件中添加等号对应的词法规则：

```
"="          { return yy::parser::make_ASSIGN  (loc); }
```

- 在 `parser.y` 文件中添加 `VarDecl`，`VarRef`，`AssignExpr`，`IntType`，`Stmtlist` 对应的语法规则：

```
VarDecl     : IntType IDENTIFIER SEMICOLON
                { $$ = new ast::VarDecl($2,$1,POS(@1)); }
            | IntType IDENTIFIER ASSIGN Expr SEMICOLON
                { $$ = new ast::VarDecl($2,$1,$4,POS(@1)); }
            ;
VarRef      : IDENTIFIER
                { $$ = new ast::VarRef($1,POS(@1)); }
            ;
AssignExpr  : VarRef ASSIGN Expr
                { $$ = new ast::AssignExpr($1, $3, POS(@1)); }
            ;
IntType     : INT
                { $$ = new ast::IntType(POS(@1)); }
```

同时在 `Expr` 对应的语法规则中加入 `AssignExpr`，`VarRef`，在 `StmtList` 中加入 `VarDecl` ，并将这几个非终结符加入 `Token`。

2.中端。

- 符号表构建和类型检查：在遍历到 `VarDecl` 节点时检查该变量是否已被声明，如果已被声明，则报错；否则，将该变量加入符号表中，如果在声明时有初始值，则递归遍历其初始值进行检查。如果在遍历到 `VarRef` 节点时发现该变量还未声明，则报错。

```
// in build_sym.cpp
void SemPass1::visit(ast::VarDecl *vdecl) {
    Type *t = NULL;
    vdecl->type->accept(this);
    t = vdecl->type->ATTR(type);
    if (scopes->lookup(vdecl->name, vdecl->getLocation(), false) != NULL) {
        issue(vdecl->getLocation(),
              new DeclConflictError(
                  vdecl->name,
                  scopes->lookup(vdecl->name, vdecl->getLocation(), false)));
    } else {
        Variable* symbol = new Variable(vdecl->name, t, vdecl->getLocation());
        scopes->declare(symbol);
        vdecl->ATTR(sym) = symbol;
        if (vdecl->init != NULL) {
            vdecl->init->accept(this);
        }
}
// in type_check.cpp
void SemPass2::visit(ast::VarRef *ref) {
    // CASE I: owner is NULL ==> referencing a local var or a member var?
    Symbol *v = scopes->lookup(ref->var, ref->getLocation());
    if (NULL == v) {
        issue(ref->getLocation(), new SymbolNotFoundError(ref->var));
        goto issue_error_type;

    } else if (!v->isVariable()) {
        issue(ref->getLocation(), new NotVariableError(v));
        goto issue_error_type;

    } else {
        ref->ATTR(type) = v->getType();
        ref->ATTR(sym) = (Variable *)v;

        if (((Variable *)v)->isLocalVar()) {
            ref->ATTR(lv_kind) = ast::Lvalue::SIMPLE_VAR;
        }
    }
    return;
    // sometimes "GOTO" will make things simpler. this is one of such cases:
issue_error_type:
    ref->ATTR(type) = BaseType::Error;
    ref->ATTR(sym) = NULL;
```

```
    return;
}
```

另外，为 `VarDecl`，`AssignExpr` 添加相应的类型检查函数，递归地对其成员变量进行类型检查：

```
void SemPass2::visit(ast::VarDecl *decl) {
    if (decl->init)
        decl->init->accept(this);
}
void SemPass2::visit(ast::AssignExpr *e) {
    e->left->accept(this);
    e->e->accept(this);
    if (!isErrorType(e->left->ATTR(type)) &&
        !e->e->ATTR(type)->compatible(e->left->ATTR(type))) {
        issue(e->getLocation(),
              new IncompatibleError(e->left->ATTR(type), e->e->ATTR(type)));
    }
    e->ATTR(type) = e->left->ATTR(type);
}
```

- 翻译成 `TAC`：
  - 对于 `VarDecl`，先生成该变量名对应的临时变量，并将这个临时变量与这个变量名关联起来，如果在声明时赋予了初始值，则需要递归遍历初始值部分，并将初始值对应的临时变量赋给这个变量名对应的临时变量。

    ```
    void Translation::visit(ast::VarDecl *decl) {
        Temp temp = tr->getNewTempI4();
        decl->ATTR(sym)->attachTemp(temp);
        if (decl->init != NULL) {
            decl->init->accept(this);
            tr->genAssign(decl->ATTR(sym)->getTemp(), decl->init->ATTR(val));
        }
    }
    ```

  - 对于 `LvalueExpr`，需要根据变量名找到其对应的临时变量，保存该临时变量供上级节点调用。

    ```
    void Translation::visit(ast::LvalueExpr *e) {
        e->lvalue->accept(this);
        const auto &sym = ((ast::VarRef *)e->lvalue)->ATTR(sym);
        e->ATTR(val) = sym->getTemp();
    }
    ```

  - 对于 `AssignExpr`，需要根据左边的变量名找到其对应的临时变量，递归遍历右边的表达式部分，并将表达式对应的临时变量赋给左边的临时变量。

    ```
    void Translation::visit(ast::AssignExpr *e) {
        e->left->accept(this);
        e->e->accept(this);
        const auto &symbol = ((ast::VarRef *)e->left)->ATTR(sym);
        tr->genAssign(symbol->getTemp(), e->e->ATTR(val));
        e->ATTR(val) = e->e->ATTR(val);
    }
    ```

3.后端。

只需要实现赋值对应的三地址码到 `riscv` 汇编代码的转换，编写 `emitAssignTac` 函数并在 `emitTac` 函数中增加相应的 `case` 即可。

```
void RiscvDesc::emitAssignTac(tac::Tac *t) {
    // eliminates useless assignments
    if (!t->LiveOut->contains(t->op0.var))
        return;
    int r1 = getRegForRead(t->op1.var, 0, t->LiveOut);
    int r0 = getRegForWrite(t->op0.var, r1, 0, t->LiveOut);
    addInstr(RiscvInstr::MOVE, _reg[r0], _reg[r1], NULL, 0, EMPTY_STR, NULL);
}
```

Step 6

1.前端。

- 在 `scanner.l` 文件中添加问号和冒号对应的词法规则：

  ```
  "?"           { return yy::parser::make_QUESTION  (loc); }
  ":"           { return yy::parser::make_COLON  (loc); }
  ```

- 经检查， `parser.y` 文件中已有的条件语句和条件表达式对应的语法规则没有错误。

2.中端。

- 在类型检查时，需要添加条件语句和条件表达式对应的递归检查函数：

```
void SemPass2::visit(ast::IfStmt *s) {
    s->condition->accept(this);
    if (!s->condition->ATTR(type)->equal(BaseType::Int)) {
        issue(s->condition->getLocation(), new BadTestExprError());
        ;
    }
    s->true_brch->accept(this);
    s->false_brch->accept(this);
}
void SemPass2::visit(ast::IfExpr *e) {
    e->condition->accept(this);
    if (!e->condition->ATTR(type)->equal(BaseType::Int)) {
        issue(e->condition->getLocation(), new BadTestExprError());
    }
    e->true_brch->accept(this);
    e->false_brch->accept(this);
    e->ATTR(type) = e->true_brch->ATTR(type);
}
```

- 在生成三地址码时，根据语义生成相应的分支跳转指令，需要注意的是，要为条件表达式生成临时变量以供上级节点调用。

```
void Translation::visit(ast::IfExpr *e) {
    Label L1 = tr->getNewLabel(); // entry of the false branch
    Label L2 = tr->getNewLabel(); // exit
    e->condition->accept(this);
    Temp temp = tr->getNewTempI4();
    e->ATTR(val)=temp;
    tr->genJumpOnZero(L1, e->condition->ATTR(val));

    e->true_brch->accept(this);
    tr->genAssign(e->ATTR(val), e->true_brch->ATTR(val));
    tr->genJump(L2); // done

    tr->genMarkLabel(L1);
    e->false_brch->accept(this);
    tr->genAssign(e->ATTR(val), e->false_brch->ATTR(val));
    tr->genMarkLabel(L2);
}
```

3.后端。

- 经检查，实验框架提供的 `prepareSingleChain` 函数准确无误，能够根据三地址码生成相应的 `riscv` 条件跳转指令。

二、思考题。

Step 5

1.我们假定当前栈帧的栈顶地址存储在 `sp` 寄存器中，请写出一段 **risc-v 汇编代码**，将栈帧空间扩大 16 字节。（提示1：栈帧由高地址向低地址延伸；提示2：risc-v 汇编中 addi reg0, reg1, <立即数> 表示将 reg1 的值加上立即数存储到 reg0 中。）

My answer：

```
addi sp, sp, -16
```

2.有些语言允许在同一个作用域中多次定义同名的变量，例如这是一段合法的 Rust 代码（你不需要精确了解它的含义，大致理解即可）：

```
fn main() {
  let a = 0;
  let a = f(a);
  let a = g(a);
}
```

其中 `f(a)` 中的 `a` 是上一行的 `let a = 0;` 定义的，`g(a)` 中的 `a` 是上一行的 `let a = f(a);`。

如果 MiniDecaf 也允许多次定义同名变量，并规定新的定义会覆盖之前的同名定义，请问在你的实现中，需要对定义变量和查找变量的逻辑做怎样的修改？（提示：如何区分一个作用域中**不同位置**的变量定义？）

My answer：

可以对一个作用域中的语句从上到下依次解析，按解析顺序对同名变量进行依次标号，再添加到作用域的符号表中，同时规定在引用变量时，引用的实际上是当前符号表中标号最大（即最新定义的）变量，这样就实现了新的定义覆盖同名定义的效果。

Step 6

1.你使用语言的框架里是如何处理悬吊 else 问题的？请简要描述。

My answer：

人为规定：`else` 和最近的 `if` 结合，为了实现这样的语法，我们要设置产生式的优先级，优先选择没有 else 的 if 。 这样，在匹配 `if(a) if(b) c=0; else d=0;` 时，第一个 if 优先选择没有 else 的 if 语法，这样第二个 if 就只能选择有 else 的 if 语法，也就实现了 else 与最近的 if 匹配。事实上 bison 默认在移进归约冲突的时候选择移进，从而实现了对悬挂 else 进行就近匹配。

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。

My answer：

在生成 `TAC` 时，先遍历 `true_branch` 和 `false_branch` 生成相应代码，再根据 `condition` 的值做相应跳转，代码实现如下：

```
void Translation::visit(ast::IfExpr *e) {
    Label L1 = tr->getNewLabel(); // entry of the false branch
    Label L2 = tr->getNewLabel(); // exit
    e->condition->accept(this);

    // changed the position of this two lines
    e->true_brch->accept(this);
    e->false_brch->accept(this);

    Temp temp = tr->getNewTempI4();
    e->ATTR(val)=temp;
    tr->genJumpOnZero(L1, e->condition->ATTR(val));

    tr->genAssign(e->ATTR(val), e->true_brch->ATTR(val));
    tr->genJump(L2); // done

    tr->genMarkLabel(L1);

    tr->genAssign(e->ATTR(val), e->false_brch->ATTR(val));
    tr->genMarkLabel(L2);
}
```