MiniDecaf Stage 2 Report

梁业升 2019010547（计 03）

2022 年 10 月 24 日

1 实验内容

1.1 词法语法分析

局部变量声明：

```
+ DeclStmt     : Type IDENTIFIER SEMICOLON
+                  { $$ = new ast::VarDecl($2, $1, POS(@1)); }
+              | Type IDENTIFIER ASSIGN Expr SEMICOLON
+                  { $$ = new ast::VarDecl($2, $1, $4, POS(@1));
    }
+              ;
+ VarRef       : IDENTIFIER
+                  { $$ = new ast::VarRef($1, POS(@1)); }
+              ;
```

变量引用：

```
+ VarRef       : IDENTIFIER
+                  { $$ = new ast::VarRef($1, POS(@1)); }
+              ;
```

赋值表达式

```
  Expr         : ICONST
                   { $$ = new ast::IntConst($1, POS(@1)); }
+              | VarRef
+                  { $$ = new ast::LvalueExpr($1, POS(@1)); }
+              | VarRef ASSIGN Expr
+                  { $$ = new ast::AssignExpr($1, $3, POS(@2));
   }
               | LPAREN Expr RPAREN
...
```

需要注意的是变量声明语句（DeclStmt）不属于语句，所以不能放在 Stmt 中；但其可以和其他语句组成复合语句，因此需要在 StmtList 中加上：

```
  StmtList    : /* empty */
                  { $$ = new ast::StmtList(); }
              | StmtList Stmt
                  { $1->append($2); $$ = $1; }
+             | StmtList DeclStmt
+                 { $1->append($2); $$ = $1; }
```

另外，IfStmt 和 IfExpr 在框架中已给出，在此不再赘述。

1.2 符号表构建

在第一个 Pass，对于 VarDecl 节点，在当前作用域中添加符号表项，并给节点添加对应的符号：

```
void SemPass1::visit(ast::VarDecl *vdecl) {
    Type *t = NULL;
    vdecl->type->accept(this);
    t = vdecl->type->ATTR(type);

    Variable *var = new Variable(vdecl->name, t, vdecl->
    getLocation());
    if (scopes->lookup(vdecl->name, vdecl->getLocation(), false
    ) != nullptr) {
        issue(vdecl->getLocation(), new DeclConflictError(vdecl
    ->name, var));
    } else {
        scopes->declare(var);
        vdecl->ATTR(sym) = var;
    }
}
```

1.3 类型检查

针对新增的表达式 IfExpr 增加类型检查（其余新增的表达式在框架中已给出）：

```
void SemPass2::visit(ast::IfExpr *e) {
    e->condition->accept(this);
    expect(e->condition, BaseType::Int);
```

```
    e->true_brch->accept(this);
    expect(e->true_brch, BaseType::Int);

    e->false_brch->accept(this);
    expect(e->false_brch, BaseType::Int);

    e->ATTR(type) = BaseType::Int;
}
```

1.4 翻译为中间代码

对于 IfExpr，我们需要支持短路求值，因此需要用到条件跳转。例如，对于 a = cond ? 1 : 0，对应的三地址码如下（设 a 和 cond 的寄存器分别为 T1 和 T2）：

```
    JZERO  T2, L1
    ASSIGN T1, 1
    JUMP   L2
L1:
    ASSIGN T1, 0
L2:
```

对应的翻译代码如下：

```
void Translation::visit(ast::IfExpr *e) {
    Label falseLabel = tr->getNewLabel();
    Label trueLabel = tr->getNewLabel();

    e->condition->accept(this);

    Temp temp = tr->getNewTempI4();

    tr->genJumpOnZero(falseLabel, e->condition->ATTR(val));
    e->true_brch->accept(this);
    tr->genAssign(temp, e->true_brch->ATTR(val));
    tr->genJump(trueLabel);

    tr->genMarkLabel(falseLabel);
    e->false_brch->accept(this);
    tr->genAssign(temp, e->false_brch->ATTR(val));

    tr->genMarkLabel(trueLabel);
```

```
    e->ATTR(val) = temp;
}
```

对于 VarDecl，我们需要为变量（符号）分配一个临时寄存器（本阶段只考虑局部变量）；如有初始化，则进行赋值：

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

对于 LvalueExpr，我们将对应符号的寄存器附在左值节点上：

```
void Translation::visit(ast::LvalueExpr *e) {
    ast::VarRef *ref;
    switch (e->lvalue->getKind()) {
    case ast::ASTNode::VAR_REF:
        ref = dynamic_cast<ast::VarRef *>(e->lvalue);
        mind_assert(ref != NULL);
        ref->accept(this);

        if (ref->ATTR(sym)->isGlobalVar()) {
            mind_assert(false);
        } else {
            e->ATTR(val) = ref->ATTR(sym)->getTemp();
        }
        break;
    default:
        mind_assert(false);
    }
}
```

1.5 生成机器代码

本阶段需要新增的三地址码翻译只有 `ASSIGN`，我们使用 `add rd, x0, rs1` 即可完成赋值：

```
void RiscvDesc::emitAssignTac(Tac *t) {
    // eliminates useless assignments
    if (!t->LiveOut->contains(t->op0.var))
        return;

    int r0 = getRegForWrite(t->op0.var, 0, 0, t->LiveOut);
    int r1 = getRegForRead(t->op1.var, r0, t->LiveOut);
    addInstr(RiscvInstr::ADD, _reg[r0], _reg[RiscvReg::ZERO],
    _reg[r1], 0, EMPTY_STR, NULL);
}
```

2 思考题

1. **Step 5**：
   (a) **1**：`addi sp, sp, -26`
   (b) **2**：定义：省略判断命名重复的步骤，直接覆盖即可；查找：无影响。
2. **Step 6**：
   (a) **1**：Bison 默认使用 Shift。
   (b) **2**：将两个 branch 分别执行完，再根据条件进行赋值。