MiniDecaf Stage 5 Report

梁业升 2019010547（计 03）

2022 年 12 月 27 日

1 实验内容

1.1 词法语法分析

增加对于数组下标和数组维度的解析，对应 `VarRef` 和 `DeclStmt` 的修改：

```
DeclStmt :
    ...
    | Type IDENTIFIER ArrayDims SEMICOLON
        { $$ = new ast::VarDecl($2, $1, $3, POS(@1)); }
    | Type IDENTIFIER ArrayDims ASSIGN LBRACE ExprList RBRACE
    SEMICOLON
        { $$ = new ast::VarDecl($2, $1, $3, $6, POS(@1)); }
    ;

ArrayDims :
    LBRACK ICONST RBRACK
        { $$ = new ast::DimList(); $$->append($2); }
    | LBRACK RBRACK
        { $$ = new ast::DimList(); $$->append(-1); }
    | ArrayDims LBRACK ICONST RBRACK
        { $$ = $1; $$->append($3); }
    ;

VarRef :
    ...
    | IDENTIFIER ArrayIndex
        { $$ = new ast::VarRef($1, $2, POS(@1)); }
    ;

ArrayIndex :
```

```
    LBRACK Expr RBRACK
        { $$ = new ast::ExprList(); $$->append($2); }
    | ArrayIndex LBRACK Expr RBRACK
        { $$ = $1; $$->append($3); }
```

另外，修改参数列表 CommaSepParamList 的定义以支持数组传参：

```
CommaSepParamList :
    ...
    | Type IDENTIFIER ArrayDims
        { $$ = new ast::VarList(); $$->append(new ast::VarDecl(
    $2, $1, $3, nullptr, POS(@1), true)); }
    ...
    | CommaSepParamList COMMA Type IDENTIFIER ArrayDims
        { $1->append(new ast::VarDecl($4, $3, $5, nullptr, POS(
    @3), true)); $$ = $1; }
    ;
```

1.2 符号表构建

在第一个 Pass，对于 VarDecl 节点，当声明为数组时，使用 ArrayType：

```
void SemPass1::visit(ast::VarDecl *vdecl) {
    ...
    if (vdecl->isArray())
        t = new ArrayType(t, vdecl->dims);
    ...
}
```

1.3 类型检查

对于 FuncCallExpr，需要检查所传参数与形参类型是否一致：

```
void SemPass2::visit(ast::FuncCallExpr *e) {
    ...
    funcArgTypeIter = func->getType()->getArgList()->begin();
    for (auto arg = e->args->begin(); arg != e->args->end();
         ++arg, ++funcArgTypeIter) {
        (*arg)->accept(this);
        expect(*arg, *funcArgTypeIter);
        ++numArgs;
    }
    ...
}
```

对于 VarRef，需要检查下标运算是否合法：

```
void SemPass2::visit(ast::VarRef *ref) {
    ...
    if (ref->isArrayRef() && !v->getType()->isArrayType()) {
        issue(ref->getLocation(), new NotVariableError(v));
        goto issue_error_type;
    }

    ref->ATTR(sym) = (Variable *)v;

    if (ref->isArrayRef()) {
        ArrayType *at = dynamic_cast<ArrayType *>(v->getType())
    ;

        if (size_t(at->getDimList()->length()) !=
            ref->indexList->length()) {
            issue(ref->getLocation(), new BadIndexError());
            goto issue_error_type;
        }

        for (auto index = ref->indexList->begin();
             index != ref->indexList->end(); ++index) {
            (*index)->accept(this);
            expect(*index, BaseType::Int);
        }
        ref->ATTR(type) = at->getElementType();
        ref->ATTR(lv_kind) = ast::Lvalue::ARRAY_ELE;
    } else {
        ref->ATTR(type) = v->getType();
        ref->ATTR(lv_kind) = ast::Lvalue::SIMPLE_VAR;
    }
}
```

对于 VarDecl，检查数组长度是否合法，注意对于参数列表中的声明可以省略第一维向量的长度。另外，检查初始化列表中的值是否为常数（只支持这种情况）。

```
void SemPass2::visit(ast::VarDecl *decl) {
    if (decl->isArray()) {
        for (auto dim = decl->dims->begin(); dim != decl->dims
    ->end(); ++dim) {
            if (*dim <= 0) {
```

```
            if (!decl->isParam() || dim != decl->dims->
    begin())
                issue(decl->getLocation(), new
    ZeroLengthedArrayError());
        }
    }
    }

    if (decl->init) {
        if (decl->isArray())
            issue(decl->getLocation(), new NotArrayError());
        decl->init->accept(this);
        if (decl->ATTR(sym)->isGlobalVar() &&
            decl->init->getKind() != ast::ASTNode::INT_CONST) {
            issue(decl->getLocation(), new NotConstInitError())
    ;
        }
        if (!decl->init->ATTR(type)->compatible(decl->ATTR(sym)
    ->getType()))
            issue(decl->getLocation(),
                    new IncompatibleError(decl->ATTR(sym)->
    getType(),
                                          decl->init->ATTR(type))
    );
    } else if (decl->init_list) {
        if (!decl->isArray())
            issue(decl->getLocation(), new NotArrayError());

        for (auto init = decl->init_list->begin();
                init != decl->init_list->end(); ++init) {
            (*init)->accept(this);
            if (decl->ATTR(sym)->isGlobalVar() &&
                (*init)->getKind() != ast::ASTNode::INT_CONST)
    {
                issue(decl->getLocation(), new
    NotConstInitError());
            }
        }
    }
}
```

1.4 翻译为中间代码

增加一个 TAC 类型：

- `ALLOC dest, size`：在栈上分配 `size` 大小的空间，首地址赋值到 `dest`

翻译 `AssignExpr`，对向数组赋值的表达式进行处理：

```
void Translation::visit(ast::AssignExpr *s) {
    ...
    if (ref->ATTR(sym)->isGlobalVar()) {
        mind_assert(ref->ATTR(addr) != NULL);
        tr->genStore(ref->ATTR(addr), 0, s->e->ATTR(val));
    } else {
        if (isArrayRef) {
            tr->genStore(ref->ATTR(addr), 0, s->e->ATTR(val));
        } else {
            tr->genAssign(ref->ATTR(sym)->getTemp(), s->e->ATTR
    (val));
        }
    }
    ...
}
```

对于 `VarRef`，我们计算全局变量或数组元素的地址，并附到节点的 `ATTR(addr)` 临时寄存器上：

```
if (ref->ATTR(sym)->isGlobalVar()) {
    Temp addr = tr->genLoadSym(ref->ATTR(sym)->getLabel());
    if (isArrayRef) {
        ref->ATTR(addr) = tr->genAdd(addr, offset);
    } else {
        ref->ATTR(addr) = addr;
    }
} else {
    if (isArrayRef) {
        Temp base = ref->ATTR(sym)->getTemp();
        ref->ATTR(addr) = tr->genAdd(base, offset);
    } else {
        ref->ATTR(addr) = nullptr;
    }
}
```

其中计算数组 `offset` 时可以对常量表达式作优化，在编译期计算出偏移量：

```
bool isConst = true;
```

```
int constOffset = 0;
int multiplier = 1;
auto dimIter = arrayType->getDimList()->rbegin();
for (auto iter = ref->indexList->rbegin();
        iter != ref->indexList->rend(); iter++, dimIter++) {
    (*iter)->accept(this);
    if ((*iter)->getKind() != ast::ASTNode::INT_CONST || !
    isConst) {
        if (isConst) {
            offset = tr->genLoadImm4(constOffset * 4);
            isConst = false;
        }
        Temp added = tr->genMul((*iter)->ATTR(val),
                                tr->genLoadImm4(multiplier * 4)
    );
        offset = tr->genAdd(offset, added);
    } else {
        int val = dynamic_cast<ast::IntConst *>(*iter)->value;
        constOffset += val * multiplier;
    }
    multiplier *= *dimIter;
}

if (isConst) {
    offset = tr->genLoadImm4(constOffset * 4);
}
```

在翻译 LvalueExpr 时，利用在上面计算得到的 VarRef 的 ATTR(addr)，可以很方便地计算 ATTR(val)。这里需要注意区分下标运算（取值）和数组传参（取地址），以及对全局变量和局部变量的区别处理：

```
bool isArrayRef = ref->isArrayRef();
bool isArrayType = ref->ATTR(sym)->getType()->isArrayType();

if (ref->ATTR(sym)->isGlobalVar()) {
    if (isArrayRef || !isArrayType) {
        e->ATTR(val) = tr->genLoad(ref->ATTR(addr), 0);
    } else {
        e->ATTR(val) = tr->genLoadSym(ref->ATTR(sym)->getLabel
    ());
    }
} else {
    if (isArrayRef) {
```

```
        e->ATTR(val) = tr->genLoad(ref->ATTR(addr), 0);
    } else {
        e->ATTR(val) = ref->ATTR(sym)->getTemp();
    }
}
```

翻译 VarDecl 时，增加对初值的处理:

```
if (decl->ATTR(sym)->isGlobalVar()) {
    if (decl->init_list != nullptr) {
        defaultValues = new int[arrLength]();
        int i = 0;
        for (auto iter = decl->init_list->begin();
                iter != decl->init_list->end(); iter++, i++) {
            ast::IntConst *intConst =
                dynamic_cast<ast::IntConst *>(*iter);
            defaultValues[i] = intConst->value;
        }
    }
    tr->genDeclGlobVar(decl->ATTR(sym)->getLabel(), arrLength,
                       defaultValues);
} else {
    if (at != nullptr) {
        Temp temp = tr->genAlloc(arrLength);
        decl->ATTR(sym)->attachTemp(temp);

        if (decl->init_list != nullptr) {
            int i = 0;
            for (auto iter = decl->init_list->begin();
                 iter != decl->init_list->end(); iter++, i++) {
                (*iter)->accept(this);
                tr->genStore(temp, i * 4, (*iter)->ATTR(val));
            }

            int remaining = arrLength - decl->init_list->length
    ();
            if (remaining > 0) {
                // Fill the rest with 0
                Temp len = tr->genLoadImm4(remaining);
                Temp startAddr = tr->genAdd(
                    temp, tr->genLoadImm4(decl->init_list->
    length() * 4));
                tr->genParam(startAddr, 0);
```

```
                tr->genParam(len, 1);
                tr->genParam(tr->genLoadImm4(0), 2);
                Label l = tr->getNewLabel();
                l->target = true;
                l->str_form = "fill_n";
                tr->genCall(l);
            }
        }
    } 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));
        }
    }
}
```

1.5 翻译为汇编代码

对于 ALLOC，首先修改 SP 在栈上分配空间，然后将修改后的 SP 赋值到目的寄存器即可。

```
void RiscvDesc::emitAllocTac(tac::Tac *t) {
    if (!t->LiveOut->contains(t->op0.var))
        return;
    // Allocate memory on stack.
    // Modify SP to get enough space for the array.
    addInstr(RiscvInstr::ADDI, _reg[RiscvReg::SP],
             _reg[RiscvReg::SP], NULL, t->op1.ival * -4,
             EMPTY_STR, "allocate memory for array on stack");

    // Get current SP.
    int regIndex = getRegForWrite(t->op0.var, 0, 0,
                                  t->LiveOut);
    addInstr(RiscvInstr::MOVE, _reg[regIndex],
             _reg[RiscvReg::SP], NULL, 0, EMPTY_STR,
             "get current SP");
}
```

2 思考题

1. **Step 11**：与现有实现类似（并未如文档所说统一分配内存），当遇到 `ALLOC` 指令时修改 `SP` 以分配空间并记录首地址，只不过空间大小是动态计算的而已。
2. **Step 12**：计算地址（偏移量）不需要用到第一维的长度。