

# 编译原理研讨课实验PR002实验报告

---

## 任务说明

### 任务要求

1. 扩展AST的表示已支持element-wise的操作
2. 操作匹配：类型匹配（静态数组，类型相同），大小匹配（大小相等）
3. 生成合法的AST
4. 不破坏原有C语言代码的语义

### 任务实现

通过修改`llvm-3-3/tool/clang/lib/Sema/SemaExpr.cpp`中的部分函数，完成对操作数的检查和匹配，并限定仅支持C语言标准的int类型的array.

## 成员组成

段江飞，蔡昕，资威

## 实验设计

### 设计思路

- 编译器生成token流、进行语法分析之后，进行语义分析阶段。语义分析阶段，会调用 `Sema::ActOnBinOp` 函数对二元操作符进行分析，而 `Sema::ActOnBinOp` 函数会调用 `Sema::BuildBinOp` 函数进行二元操作符的创建，在 `Sema::BuildBinOp` 函数中，对于固定(built-in)的非重载的操作类型,需要调用 `Sema::CreateBuiltinBinOp` 函数来创建二元操作符。为了支持 `elementWise` 操作，我们需要在二元操作符是 '='、'+'、'\*' 且函数有 `elementWise` 标记的时候进行操作数的检查和匹配，所以本实验需要修改函数 `Sema::CheckForModifiableLvalue`，`Sema::CheckAssignmentOperands`，`Sema::CheckAdditionOperands`，`Sema::CheckMultiplyDivideOperands` 以达到实验目的。
- 语法树的构造  
通过正常的两个int类型的加法可以看到，'+'的两个操作数都会有左值转化为右值的过程，所以对数组的 `elementWise` 的操作的设计可以类似这种操作。

```

-FunctionDecl 0x6cb7460 <test/sim.c:2:1, line:15:1> f 'void ()'
-CompoundStmt 0x6ce4518 <line:3:1, line:15:1>
-DeclStmt 0x6cb7568 <line:4:2, col:7>
-VarDecl 0x6cb7510 <col:2, col:6> a 'int'
-DeclStmt 0x6cb75e8 <line:5:2, col:7>
-VarDecl 0x6cb7590 <col:2, col:6> b 'int'
-DeclStmt 0x6cb7668 <line:6:2, col:7>
-VarDecl 0x6cb7610 <col:2, col:6> c 'int'
-BinaryOperator 0x6cb7750 <line:7:2, col:10> 'int' '='
-DeclRefExpr 0x6cb7680 <col:2> 'int' lvalue Var 0x6cb7610 'c' 'int'
-BinaryOperator 0x6cb7728 <col:6, col:10> 'int' '+'
-ImplicitCastExpr 0x6cb76f8 <col:6> 'int' <LValueToRValue>
-DeclRefExpr 0x6cb76a8 <col:6> 'int' lvalue Var 0x6cb7510 'a' 'int'
-ImplicitCastExpr 0x6cb7710 <col:10> 'int' <LValueToRValue>
-DeclRefExpr 0x6cb76d0 <col:10> 'int' lvalue Var 0x6cb7590 'b' 'int'

```

在合法性检查之后，对数组进行左值到右值的转化，然后利用clang已有的代码构造相应的AST树。

```

-BinaryOperator 0x6ce44b0 <col:6, col:10> 'int [100]' '+'
-ImplicitCastExpr 0x6ce4480 <col:6> 'int [100]' <LValueToRValue>
-DeclRefExpr 0x6ce4430 <col:6> 'int [100]' lvalue Var 0x6ce4230 'D' 'int [100]'
-ImplicitCastExpr 0x6ce4498 <col:10> 'int [100]' <LValueToRValue>
-DeclRefExpr 0x6ce4458 <col:10> 'int [100]' lvalue Var 0x6ce42d0 'E' 'int [100]'

-BinaryOperator 0x6ce40e0 <line:10:2, col:6> 'int *' '='
-DeclRefExpr 0x6ce4078 <col:2> 'int *' lvalue Var 0x6cb77c0 'A' 'int *'
-ImplicitCastExpr 0x6ce40c8 <col:6> 'int *' <LValueToRValue>
-DeclRefExpr 0x6ce40a0 <col:6> 'int *' lvalue Var 0x6ce3ff0 'B' 'int *'

```

- '='操作
  1. 检查左操作数是否assignable，在Sema::CheckForModifiableLvalue中修改使得带有elementWise标记的函数里的数组为assignable
  2. Sema::CheckAssignmentOperands中，检查函数是否有elementWise标记和右操作数是否是静态数组类型(左操作数在assignable的修改中已经检查)
  3. 检查数组的大小和元素类型是否匹配以及是否是int类型的数组
  4. '='的左操作数需要是lvalue，进行lvalue的检查，然后对右操作数根据需要进行lvalue到rvalue的转化
- '+'操作
- '\*'操作

## 具体实现

Sema::CheckForModifiableLvalue 的添加部分

```

if(IsLV == Expr::MLV_ArrayType && S.IsElementWise &&
    ConstantArrayType::classof(E->getType().getTypePtr()))
    return false;

```

Sema::CheckAssignmentOperands 的添加部分:

```

//handle the case: unqualified array '='
//check if this function support elementwise and RHS'type is ConstantArray
if(this->IsElementWise && RHSType.getTypePtr()->isConstantArrayType())
{
    const ConstantArrayType *lhs = dyn_cast<ConstantArrayType>
(LHSType.getTypePtr());
    const ConstantArrayType *rhs = dyn_cast<ConstantArrayType>
(RHSType.getTypePtr());
    QualType lhs_dt = lhs->getElementType().getUnqualifiedType();
    QualType rhs_dt = rhs->getElementType().getUnqualifiedType();
    //check whether LHS and RHS have same size and type and make sure type of
array is int
    if(lhs->getSize() == rhs->getSize() && lhs_dt == rhs_dt && \
        lhs_dt.getTypePtr()->isIntegerType())
    {
        // check whether LHSExpr is lvalue
        if(LHSExpr->isLValue())
        {
            //if RHSExpr is not of rvalue kind ,transfer RHSCheck to rvalue kind for
assignment and building AST

            if(!(RHSCheck->isRValue()))
            {
                Qualifiers tmp;
                ImplicitCastExpr *rhs_r2l = ImplicitCastExpr::Create(Context,
                    Context.getUnqualifiedArrayType(RHSType.getUnqualifiedType(), tmp),
                    CK_LValueToRValue, RHSCheck, 0, VK_RValue);
                RHS = rhs_r2l;
            }
        }

        return RHS.get()->getType();
    }
}

```

Sema::CheckAdditionOperands的添加部分:

```

//handle the case: unqualified array '+'
//check if this function support elementwise and LHS's and RHS'type is
ConstantArray
if(this->IsElementWise && LHS.get()->getType().getTypePtr()-
>isConstantArrayType() &&
    RHS.get()->getType().getTypePtr()->isConstantArrayType())
{
    const ConstantArrayType *lhs = dyn_cast<ConstantArrayType>(LHS.get()-
>getType().getTypePtr());
    const ConstantArrayType *rhs = dyn_cast<ConstantArrayType>(RHS.get()-
>getType().getTypePtr());
    QualType lhs_dt = lhs->getElementType().getUnqualifiedType();
    QualType rhs_dt = rhs->getElementType().getUnqualifiedType();

```

```

    //check whether LHS and RHS have same size and type and make sure type of
    array is int
    if(lhs->getSize() == rhs->getSize() && lhs_dt == rhs_dt &&
        lhs_dt.getTypePtr()->isIntegerType())
    {
        //if LHS or RHS is not of rvalue kind ,transfer it to rvalue kind for further
        addition computation
        if(!(LHS.get()->isRValue()))
        {
            Qualifiers tmp;
            ImplicitCastExpr *lhs_r2l = ImplicitCastExpr::Create(Context,
                Context.getUnqualifiedArrayType(LHS.get()-
>getType().getUnqualifiedType(), tmp),
                CK_LValueToRValue, LHS.get(), 0, VK_RValue);
            LHS = lhs_r2l;
        }
        if(!(RHS.get()->isRValue()))
        {
            Qualifiers tmp;
            ImplicitCastExpr *rhs_r2l = ImplicitCastExpr::Create(Context,
                Context.getUnqualifiedArrayType(RHS.get()-
>getType().getUnqualifiedType(), tmp),
                CK_LValueToRValue, RHS.get(), 0, VK_RValue);
            RHS = rhs_r2l;
        }
        return LHS.get()->getType();
    }
}

```

Sema:CheckMultiplyDivideOperands的添加部分:

```

    //handle the case: unqualified array '*'
    //check whether this function support elementwise and LHS's and RHS'type is
    ConstantArray (only for multiply)
    if(!IsDiv && !IsCompAssign && this->IsElementWise &&
        LHS.get()->getType().getTypePtr()->isConstantArrayType() &&
        RHS.get()->getType().getTypePtr()->isConstantArrayType() )
    {
        const ConstantArrayType *lhs = dyn_cast<ConstantArrayType>(LHS.get()-
>getType().getTypePtr());
        const ConstantArrayType *rhs = dyn_cast<ConstantArrayType>(RHS.get()-
>getType().getTypePtr());
        QualType lhs_dt = lhs->getElementType().getUnqualifiedType();
        QualType rhs_dt = rhs->getElementType().getUnqualifiedType();
        //check whether LHS and RHS have same size and type and make sure the type of
        array is int
        if(lhs->getSize() == rhs->getSize() && lhs_dt == rhs_dt &&
            lhs_dt.getTypePtr()->isIntegerType())
        {
            //if LHS or RHS is not of rvalue kind ,transfer it to rvalue kind for
            further computation
            if(!(LHS.get()->isRValue()))

```

```
{
    Qualifiers tmp;
    ImplicitCastExpr *lhs_r2l = ImplicitCastExpr::Create(Context,
        Context.getUnqualifiedArrayType(LHS.get()-
>getType().getUnqualifiedType(), tmp),
        CK_LValueToRValue, LHS.get(), 0, VK_RValue);
    LHS = lhs_r2l;
}
if(!(RHS.get()->isRValue()))
{
    Qualifiers tmp;
    ImplicitCastExpr *rhs_r2l = ImplicitCastExpr::Create(Context,
        Context.getUnqualifiedArrayType(RHS.get()-
>getType().getUnqualifiedType(), tmp),
        CK_LValueToRValue, RHS.get(), 0, VK_RValue);
    RHS = rhs_r2l;
}
return LHS.get()->getType();
}
```

## 测试过程

## 总结

### 实验结果总结

经过测试，实验结果能够满足任务书中支持C语言标准的int类型的elementWize操作，检查操作匹配以及生成合法的AST的要求，并且不会破坏原有C语言代码的语义。

### 分成员总结

段江飞： '+'、 '='、 '\*'代码实现

资 威： '\*'代码实现，测试

蔡 昕： 实验报告编写