PR002 实验报告

第 11 组 谭弘泽 张传奇 冯凌璇

【实验目标】

对于实验 1 实现的 asCheck 制导语句，检查其限制范围内的函数，实现以下功能：

1. 检查出其中声明的未定义边界的数组
2. 检查出其中没有 else 的 if 语句
3. 检查出其中没有 break 的 case 语句
4. 检查出其中使用的多级指针

输出格式为：Error : “错误原因”. line: “行号” col: “列号”

示例：

```
Error : Missing boundary. line:1 col:1
Error : If stmt miss else stmt. line:1 col:1
Error : Missing break. line:1 col:1
Error : MultiPointer. line:1 col:1
```

【实验流程】

1. asCheck 的总体流程

   找到函数解析完成时的入口，插入语义动作，作为自定义检查的入口 Sema::ActOnFinishFunctionBody 是函数结束后的动作，作为我们的入口 Sema::ActOnDoingAsCheck 是我们自己定义的检查主函数 定义一个 Sema::asCheck_entry 作为每次递归调用的入口。

2. 实现入口函数 asCheck_entry

   根据 Stmt 的不同类型调用不同的检查函数，并实现 AST 树的遍历。当 Stmt 为 SwitchStmt 类则检查 case 语句是否有 break；当 Stmt 为 IfStmt 类，则检查 if 语句是否为空；当 Stmt 为 DeclStmt 类，则检查未定义边界数组和多级指针。

```
void Sema::asCheck_entry(const Stmt* S)
{
    if(!S)return ;
    switch(S->getStmtClass())
    {
```

```
        case Stmt::SwitchStmtClass:
            checkCaseAndBreak(S);
            break;
        case Stmt::IfStmtClass:
            checkNoElse(S);
            break;
        case Stmt::DeclStmtClass:
            checkDeclStmt(S);
            break;
        default:
            break;
    }
    for(Stmt::const_child_range CI = S->children();CI;++CI){
        if(*CI != NULL)asCheck_entry(*CI);
    }
    return;
}
```

3. 实现 case-break 匹配的查找

遍历 Stmt 的子结点。初始时，将 needBreak 置为 false，遇到 CaseStmtClass 时，needBreak 会被值为 true，遇到 DefaultStmt 时，needBreak 被置为 false。每次遍历遇到 CaseStmtClass 或者 DefaultStmt 检查 needBreak 的值，如果为真说明前面的 case 没有 break。为了解决 switch 语句的嵌套问题，我们组选择维护一个栈来实现 case 和 break 的匹配，对于每一个 switch 语句，将所有的 case 和 default 都压入栈中，一个 switch 语句结束时将栈清空，检查 needBreak 的值。用这种办法就可以解决 switch 的嵌套问题。

```
void Sema::checkCaseAndBreak(const Stmt *S)
{
    const bool debug = false;

    bool needBreak = false;
    const Stmt* lastCaseStmt = NULL;
```

```
    Stmt::const_child_range CI = S->children();
    std::stack<Stmt::const_child_range> vstack;

    if(debug)llvm::errs() << "start while\n";
    while (CI) {
        if(debug)llvm::errs() << "still while ";
        const Stmt* subStmt = *CI;
        if(*CI == NULL)
        {
            ++CI;
            continue;
        }
        if(debug)llvm::errs() << subStmt->getStmtClassName() << "\n";
        switch(subStmt->getStmtClass())
        {
            case Stmt::DefaultStmtClass:
                if(needBreak){
                    printStmtLoc(lastCaseStmt," Error : Missing
break.");
                }
                needBreak = false;
                lastCaseStmt = subStmt;
                vstack.push(CI);
                CI = subStmt->children();
                continue;
            case Stmt::CaseStmtClass:
                if(needBreak){
                    printStmtLoc(lastCaseStmt," Error : Missing
break.");
                }
                needBreak = true;
                lastCaseStmt = subStmt;
                vstack.push(CI);
```

```
                CI = subStmt->children();
                if(debug)llvm::errs() << "push\n";
                continue;
            case Stmt::BreakStmtClass://find break stmt at same level.
                if(needBreak)needBreak = false;
                break;
            case Stmt::CompoundStmtClass:
                if(subStmt->children())
                {
                    if(debug)llvm::errs() << "push\n";
                    vstack.push(CI);
                    CI = subStmt->children();
                    continue;
                }
                else break;
            default:
                break;
        }
        ++CI;
        while(!CI&&!vstack.empty())
        {
            CI = vstack.top();
            vstack.pop();
            if(debug)llvm::errs() << "pop\n";
            ++CI;
        }
    }// while
    //The needBreak must be false
    if(needBreak){
        //lost break, print Error info
        printStmtLoc(lastCaseStmt," Error : Missing break.");
    }
    return;
```

```
}
```

但是，对于 switch 语句的 case 语句中含有例如 if 语句等有多个控制流的的情况，原本我们考虑可以判断如果每个控制流语句都含有 break，那么可以判断该 case 语句有 break 匹配，否则报错：Missing break。但是，我们后来发现如下的情况也是可能存在的，并且是可以过编译的：

```
switch(a)
{
    ...
    case 1:
        if(true) {case 2: break;}
    ...
}
```

类似的，还可能会有以下两种情况：

```
switch(a)
{
    ...
    case 1:
        if(false) {case 2:;}
        break;
    ...
}

switch(a)
{
    ...
    case 0:
        if(b)
            case 1:
        else
            case 2:
        break;
    ...
```

```
}
```

但是这些情况太麻烦了，需要构造分支树和执行流等，于是我们选择不处理，直接忽略。

4. 实现 if-else 的查找

利用 IfStmt 类下的 getElse 方法，如果有 else 语句，那么这个方法就会返回指向 else 语句的指针，没有 else 的情况下这个方法放回 NULL，以此作为判据检查出没有 else 的 if 语句。

```
void Sema::checkNoElse(const Stmt *S)
{
    const Stmt* else_stmt =  (dyn_cast<IfStmt>(S))->getElse();
    if(else_stmt==NULL)
        printStmtLoc(S," Error : If stmt miss else stmt.");
    return;
}
```

5. 实现多级指针解析时的检查

多级指针的查找主要由函数 Sema::checkDeclStmt 实现。每次在函数 asCheck_entry 遇到变量声明时就调用该函数。在该函数中，通过 getDeclGroup 方法可以获得一个声明的 group，遍历该 group，分不同的类别进行不同的检查：

```
DGrp = DS->getDeclGroup();
        for ( I = DGrp.begin(), E = DGrp.end(); I != E; ++I) {  // check
each decl in DeclStmt
            VD = VarD = dyn_cast_or_null<VarDecl>(*I);
            if (VD) {  // ValueDecl
                QT = VD->getType();
                if (checkdepth(QT))
                    printDeclLoc(VD," Error : MultiPointer.");
            }
            ..........
            RD = dyn_cast_or_null<RecordDecl>(*I);
            if (RD) {   // RecordDecl --> struct
                checkRecordDecl(RD);
```

```
            }
            TdD = dyn_cast_or_null<TypedefDecl>(*I);
            if (TdD) {  // TypedefDecl
                QT = TdD->getUnderlyingType();
                if (checkdepth(QT))
                    printDeclLoc(TdD," Error : MultiPointer.");
            }
        }
```

如果是简单的 ValueDecl 类（例如 int **a;）或者是 TypedefDecl 类（例如 typedef int a;），则直接调用 Sema::checkdepth 函数来检查指针的级数。如果是 RecordDecl 类，说明该声明是一个结构，需要调用函数 Sema::checkRecordDecl 对结构内部的声明进行遍历检查，如果结构内含有结构的声明，则需要递归检查。

```
void Sema::checkRecordDecl(const RecordDecl *RD){
    //RecordDecl::field_iterator FE,FI;
    const ValueDecl* VD;
    const RecordDecl *RD1;
    QualType QT;
    const FieldDecl *F;

    for (DeclContext::decl_iterator I = RD->decls_begin(), E =
RD->decls_end(); I!=E; ++I){
        RD1 = dyn_cast_or_null<RecordDecl>(*I);
        if (RD1){
            checkRecordDecl(RD1);
        }else{
            F = dyn_cast_or_null<FieldDecl>(*I);
            if(F){
                QT = F->getType();
                if (checkdepth(QT))
                    printDeclLoc(F," Error : MultiPointer.");
            }
```

```
        }
    }
}
```

Sema::checkdepth 函数的实现：输入为一个 QualType 类，初始化指针级数的计数器 pdepth 为 0，每次判断 QT 是不是一个指针类型，如果是，则将计数器加一，QT 变为原 QT 被指向的类别；否则停止计数，得到指针最后指向的类别，如果是结构，仍然需要检查结构内部的声明。最后判断计数器的值，如果 pdepth 大于等于二，说明用了超过多级指针，返回 true 用于报错。

```
bool Sema::checkdepth(QualType QT) {
    int pdepth = 0;
    // check pointer level
    for ( ; QT->isPointerType(); QT = QT->getPointeeType()) {
        ++pdepth;
    }
    if (QT->isRecordType()){
        checkRecordDecl(QT->getAsStructureType()->getDecl());
    }
    if (pdepth >= 2){
        return true;
    }
    return false;
}
```

6. 实现数组边界未定义的查找

观察-Xclang -ast-dump 得到的 AST 树，我们发现形如 int a[]={1,2}这样的语句会变成和 int a[2]={1,2}同样的子树，无法在 AST 处对二者进行区分。

所以原计划在词法层面解决问题，原计划是单独写一个函数，在其中新开一个 Lexer ，扫描函数体中的[]，如果这两者连续出现便报错。但是我们在测试的时候发现的一个问题，即存在着这种很可能发生的情况：

```
#define N  //忘记写具体的值
```

```
{
        ...
        int a[N] = {1,2,3,4,5};
        ...
}
```

为了排除这种情况，我们重新换了一种思路，既然形如 int a[]={1,2}的语句本身并未直接给出其数组宽度 2，因此，必然存在一处语法动作使得数组宽度被计算出来并回填到 AST 树结点。那么，只要在此处插入代码将标签置位，则在检查中遇到声明的时候只需对标签是否被置位进行检查即可。

所以在 Sema::AddInitializerToDecl 中，当检查到不完全数组类型的时候，把我们预先定义好的 VarDecl 类中的 IsRefilledBoundary 成员变量置为 true。

```
if (!VDecl->isInvalidDecl() && (DclT != SavT))
  {
    VDecl->setIsRefilledBoundary(true);//delay message until doing asCheck
    VDecl->setType(DclT);
  }
```

而后，和检查多级指针在并列位置，当检查到某个 VarDecl 的 IsRefilledBoundary 为 true 时会报错。

```
void Sema::checkDeclStmt(const Stmt* subStmt) { // 当发现变量定义时该接口被调用
     ……
    if(subStmt->getStmtClass()==Stmt::DeclStmtClass){
        DS = dyn_cast<DeclStmt>(subStmt);
        DGrp = DS->getDeclGroup();
        for ( I = DGrp.begin(), E = DGrp.end(); I != E; ++I) {  // check each decl in DeclStmt
            VD = VarD = dyn_cast_or_null<VarDecl>(*I);
            ……
            if(VarD){  //VarDecl
                if (VarD->getIsRefilledBoundary())
```

```
                    printDeclLoc(VarD," Error : Missing boundary.");
                }
                ……
            }
        }
    }
```

7. 修改文件汇总

   Decl.h include/clang/AST

   SemaDecl.cpp lib/Sema

   SemaChecking.cpp lib/Sema

【实验结果】

见测试文件