Lab4.2 实验报告

PB20111630 张艺耀

实验要求

GVN(Global Value Numbering) 全局值编号，是一个基于 SSA 格式的优化，通过建立变量，表达式到值编号的映射关系，从而检测到冗余计算代码并删除。本次实验采用的算法参考论文为：[Detection of Redundant Expressions: A Complete and Polynomial-Time Algorithm in SSA] 在该论文中，提出了一种适合 SSA IR，多项式算法复杂度的数据流分析方式的算法，能够实现对冗余代码完备的检测。本次实验中，我们将在`Light IR`上实现该数据流分析算法，并根据数据流分析结果删掉冗余代码，达到优化目的。

根据要求补全`src/optimization/GVN.cpp`，`include/optimization/GVN.h`中关于 GVN pass 数据流分析部分 使用GVN算法进行数据流优化。

实验难点

phi左右两边都是相同的值的情况::

如果对于 phi 指令 x2 = phi(x1,x1), x1 是其所在等价类的代表元的话，则这个phi指令无法删除，解决方法是在`ValuephiExpr`函数中增加一个if语句特判语句。

对于值编码在每次迭代循环的处理:

对于同样的`pin`，在不同轮次的迭代中通过`transferFunction`生成的等价类相同，但有不同的`index`，例如:

在`loop3.cminus`第一次迭代循环的时候，可以推得在 label24 中有等价类， 而在第二次循环迭代的时候，可以推得此时在 label11 中有等价类:`index: 8 index: 11 value phi: nullptr value phi: nullptr value expr: (add v6 v7) value expr: (add v9 v10)`但由于`label24`和`label11`都是`label17`的前驱，需要在`label17`中进行`join()`:

```
index: 23
value phi: (phi (add v9 v10) (add v6 v7))
value expr: (phi (add v9 v10) (add v6 v7))
members: {%op14 = add i32 %op12, %op13; %op18 = phi i32 [ %op14,
%label11 ], [ %op27, %label24 ]; }
```

可以看到 intersect() 为它们产生了 phi表达式，但此时的 op14 的值表达式应该是 (add v21 v22)，其中 v21 和 v22 分别是 op12 和 op13 的值编号，因此出现错误。解决方法是在每次迭代时，重置`next_value_number`为相同的起始值，代码如下:

```
...
std::uint64_t value_number_start = next_value_number_;
    // iterate until converge
    // TODO:
    do {
        next_value_number_ = value_number_start;
        bb_reached[entry] = 1;
        for(auto &bb : func_->get_basic_blocks()) {
            if(&bb == entry) {
                continue;
            } else {
                bb_reached[&bb] = 0;
            }
        }
        // see the pseudo code in documentation
        for (auto &bb : func_->get_basic_blocks()) { // you might need to visit the blocks in depth-first order
            // get PIN of bb by predecessor(s)
            // iterate through all instructions in the block
            // and the phi instruction in all the successors
            if(&bb == entry) continue;
            auto pre_bb_list = bb.get_pre_basic_blocks();
            bb_reached[&bb] = 1;
            if(pre_bb_list.empty()) continue;
            if(pre_bb_list.size() == 2) {
                temp = join(pout_[pre_bb_list.front()], pout_[pre_bb_list.back()], pre_bb_list.front(), pre_bb_list.back());
            } else {
                auto it = bb.get_pre_basic_blocks().begin();
                partitions p3 = pout_[*it];
                temp = clone(p3);
            }

            for(auto &instr : bb.get_instructions()) {
                if(instr.is_phi()) continue;
                temp = transferFunction(&instr, &instr, temp, bb);
            }

            // copy statement
            auto succ_bb_list = bb.get_succ_basic_blocks();
            for(auto succ_bb : succ_bb_list) {
                for (auto &instr : succ_bb->get_instructions()) {
```

```
                    if ((&instr)->is_phi()) {
                        //copy stmt
                        temp = cpStmt(&instr, &instr, temp, bb);
                    }
                }
            }
            // check changes in pout
            if(!(pout_[&bb] == temp)) {
                changed = true;
            } else {
                changed = false;
            }
            pout_[&bb] = std::move(temp);

        }
    } while (changed);
...
```

实验设计

实现思路，相应代码，优化前后的IR对比（举一个例子）并辅以简单说明

GVN.h

CongruenceClass

```
struct CongruenceClass {
    size_t index_;
    // representative of the congruence class, used to replace all the members (except itself) when analysis is done
    Value *leader_;
    // value expression in congruence class
    std::shared_ptr<GVNExpression::Expression> value_expr_;
    // value φ-function is an annotation of the congruence class
    std::shared_ptr<GVNExpression::PhiExpression> value_phi_;
    // equivalent variables in one congruence class
    std::set<Value *> members_;
    //constant
    Constant *c_;

    bool phi_;

```

```
    CongruenceClass(size_t index) : index_(index), c_{}, leader_{}, value_expr_{}, value_phi_{}, members_{}, phi_(false){}

    bool operator<(const CongruenceClass &other) const { return this->index_ < other.index_; }
    bool operator==(const CongruenceClass &other) const;

};
```

等价类里增加了常量用于常量判断和折叠

EXpression

GVNExpression增加了数个类用于处理不同的Expression，其中具有代表性的是VNExpression，它代表的是值编码，具有一个`size_t`类型的成员；还有运算符所对应的各个类用于实现对冗余指令的检测与消除(`add, sub, mul, sdiv, fadd, fsub, fmul, fdiv, getelementptr, cmp, fcmp, zext, fptosi, sitofp`):

`enum gvn_expr_t { e_constant, e_bin, e_phi, e_vn, e_cmp, e_gep, e_cast, e_call};`

以`CallExpression`为例:

```
class CallExpression : public Expression {
  public:
    static std::shared_ptr<CallExpression> create(Function *func, std::vector<std::shared_ptr<Expression>> args) {
      return std::make_shared<CallExpression>(func, args);
    }
    virtual std::string print() {return " Call ";}
    bool equiv(const CallExpression *other) const {
      return (func_ == other->func_ && args_ == other->args_);
    }
    CallExpression(Function *func, std::vector<std::shared_ptr<Expression>> args) : Expression(e_call), func_(func), args_(args) {}
    Function *retFun() {return func_;}

  private:
    Function *func_;
    std::vector<std::shared_ptr<Expression>> args_;
};
```

它的私有成员分别是`Function *func_`和`std::vector<std::shared_ptr<Expression>> args_`其中前者用于表示函数的类型，后者存储了函数的参数对应的表达式。

值得注意的是`CmpExpression`不仅仅需要`Op`成员，还需要有一个`CmpOp`的成员用来辨别`CmpOp`的类型。同时因为`icmp`和`fcmp`是不同的类型，在`ValueExpr`中要使用`unsigned int`类型进行转换。

GVN.cpp

在原有框架的基础上添加了如下函数：

```
    // 得到ve对应值编码
    std::shared_ptr<GVNExpression::Expression> getVN(const partitions &pout,

std::shared_ptr<GVNExpression::Expression> ve);
    //  对于Val 创建并返回ve 记录常量
    std::shared_ptr<GVNExpression::Expression> getVE(Constant **con, Value *val, partitions pin);

    // 得到值编码vn对应的phi表达式
    std::shared_ptr<GVNExpression::PhiExpression> getVP(std::shared_ptr<GVNExpression::VNExpression> vn, partitions pin);

    // 拷贝函数用于detectEquiv函数
    partitions cpStmt(Instruction *x, Value *e, partitions pin, BasicBlock &bb);

```

Join()

对应伪代码对Ci Cj取交集即可。

```
GVN::partitions GVN::join(const partitions &P1, const partitions &P2,
BasicBlock *lbb, BasicBlock *rbb) {
    // TODO: do intersection pair-wise
    partitions P;
    for(auto ci : P1) {
        for(auto cj : P2) {
            auto Ck = intersect(ci, cj, lbb, rbb);
            if(Ck == nullptr) continue;
            P.insert(Ck);
        }
    }
    return P;
}
```

Intersect()

对应伪代码编写，其中需要注意当Ci和Cj的`leader_`相等时无需建立Phi表达式，直接返回即可。

```
std::shared_ptr<CongruenceClass>
GVN::intersect(std::shared_ptr<CongruenceClass> Ci,

 std::shared_ptr<CongruenceClass> Cj,
                                                BasicBlock *lbb,
BasicBlock *rbb) {

    std::shared_ptr<CongruenceClass> Ck = createCongruenceClass();

    if(Ci->index_ == 0) return Cj;
    if(Cj->index_ == 0) return Ci;

    for (auto &i : Ci->members_) {
        for (auto &j : Cj->members_) {
            if (i == j) Ck->members_.insert(i);
        }
    }

    if(Ck->members_.size() == 0) return nullptr;

    if(Ci->leader_ == Cj->leader_) {
        Ck->leader_ = Ci->leader_;

```

```
        switch (bb_reached[lbb]) {
            case 1 : {
                Ck->index_ = Ci->index_;
                Ck->value_expr_ = Ci->value_expr_;
                Ck->value_phi_ = Ci->value_phi_;
                Ck->c_ = Ci->c_;
                break;
            }

            default: {
                Ck->index_ = Cj->index_;
                Ck->value_expr_ = Cj->value_expr_;
                Ck->value_phi_ = Cj->value_phi_;
                Ck->c_ = Cj->c_;
                break;
            }
        }

        return Ck;
    }

    Ck->index_ = next_value_number_++;
    auto iter = Ck->members_.begin();
    shared_ptr<Expression> lhs;
    shared_ptr<Expression> rhs;

    if(Ci->phi_) lhs = ConstantExpression::create(Ci->c_);
    else lhs = VNExpression::create(Ci->index_);

    if(Cj->phi_) rhs = ConstantExpression::create(Cj->c_);
    else rhs = VNExpression::create(Cj->index_);

    Ck->leader_ = *iter;
    Ck->value_expr_ = nullptr;
    Ck->value_phi_ = PhiExpression::create(lhs, rhs, lbb, rbb);
    Ck->c_ = nullptr;

    return Ck;
}
```

DetectEquivalences()

首先对全局变量和函数参数进行处理，然后初用顶元始化所有`pout`之后初始化第一个基本块之后的处理与伪代码相同，细节上在每轮迭代寻找并设置cpStmt

```
void GVN::detectEquivalences() {

    bool changed = false;

    partitions P;

    //  set global value
    for (auto &gv : m_->get_global_variable()) {
        auto cc = createCongruenceClass(next_value_number_++);
        if(gv.is_const()){
            cc->leader_ = gv.get_init();
            cc->value_expr_ =
ConstantExpression::create(gv.get_init());
            cc->members_.insert(&gv);
            cc->c_ = gv.get_init();
        }
        else{
            cc->leader_ = &gv;
            cc->members_.insert(&gv);
        }
        P.insert(cc);
    }

    // set args
    for (auto arg : func_->get_args()) {
        auto cc = createCongruenceClass(next_value_number_++);
        cc->leader_ = arg;
        cc->members_.insert(arg);
        P.insert(cc);
    }

    // get entry block
    auto entry = func_->get_entry_block();


    // initialize
    for(auto &instr : entry->get_instructions()) {
```

```
        if(! instr.is_phi())
        P = transferFunction(&instr, &instr, P, *entry);
    }
    for(auto successor : entry->get_succ_basic_blocks()) {
        for(auto &instr : successor->get_instructions()) {
            if(instr.is_phi())
                P = cpStmt(&instr, &instr, P, *entry);
        }
    }

    pout_[entry] = std::move(P);

    // set up top partition
    std::shared_ptr<CongruenceClass> top_ = createCongruenceClass();
    partitions top;
    top.insert(top_);

    // initialize pout with top
    for (auto &bb : func_->get_basic_blocks()) {
        if (!(&bb == entry)) {
            pout_[&bb] = clone(top);
        }
    }

    std::uint64_t start = next_value_number_;

    // iterate until converge
    // TODO:
    do {
        next_value_number_ = start;
        bb_reached[entry] = 1;
        for(auto &bb : func_->get_basic_blocks()) {
            if(&bb == entry) {
                continue;
            } else {
                bb_reached[&bb] = 0;
            }
        }
        // see the pseudo code in documentation
        for (auto &bb : func_->get_basic_blocks()) { // you might need
to visit the blocks in depth-first order
            // get PIN of bb by predecessor(s)
            // iterate through all instructions in the block
```

```
            // and the phi instruction in all the successors

            if(&bb == entry) continue;
            auto pre_bb_list = bb.get_pre_basic_blocks();
            bb_reached[&bb] = 1;

            //
            if(pre_bb_list.empty()) continue;
            if(pre_bb_list.size() == 2) {
                P = join(pout_[pre_bb_list.front()], pout_[pre_bb_list.back()], pre_bb_list.front(), pre_bb_list.back());
            } else {
                auto it = bb.get_pre_basic_blocks().begin();
                partitions p3 = pout_[*it];
                P = clone(p3);
            }

            for(auto &instr : bb.get_instructions()) {
                if(instr.is_phi()) continue;
                P = transferFunction(&instr, &instr, P, bb);
            }

            // copy statement
            auto succ_bb_list = bb.get_succ_basic_blocks();
            for(auto succ_bb : succ_bb_list) {
                for (auto &instr : succ_bb->get_instructions()) {
                    if ((&instr)->is_phi()) {
                        //copy stmt
                        P = cpStmt(&instr, &instr, P, bb);
                    }
                }
            }

            // check changes in pout
            if(!(pout_[&bb] == P)) {
                changed = true;
            } else {
                changed = false;
            }
            pout_[&bb] = std::move(P);

        }
    } while (changed);
```

```
121  }
```

ValueExpr()

代码过长故不在此展示，但其中每个指令的处理都很相似。对于纯函数只需加一句if语句特判：

```
1  if(! func_info_->is_pure_function(dynamic_cast<Function *>(instr-
>get_operand(0)))) {
2              return nullptr;
3          }
```

TransferFunction()

```
1  GVN::partitions GVN::transferFunction(Instruction *x, Value *e,
partitions pin, BasicBlock &bb) {
2      // TODO: get different ValueExpr by Instruction::OpID, modify pout
3      partitions pout = clone(pin);
4
5      if(x->is_void()) return pout;
6
7      Constant *constant = nullptr;
8      shared_ptr<Expression> ve = valueExpr(x, pin, &constant);
9
10     if(ve == nullptr or x->is_alloca() or x->is_load()) {
11         for(auto cc : pout){
12             if(cc->members_.find(x) != cc->members_.end())
13                 return pout;
14         }
15         auto cc = createCongruenceClass(next_value_number_++);
16         cc->leader_ = x;
17         cc->members_.insert(x);
18         pout.insert(cc);
19         return pout;
20     }
21
22     shared_ptr<PhiExpression> vpf = valuePhiFunc(ve, pin);
23     for(auto cc : pout ){
24         if(cc->value_expr_ == ve or
25            ( vpf and (std::static_pointer_cast<Expression>((cc)-
>value_phi_) == std::static_pointer_cast<Expression>(vpf)))) {
26                 if(constant){
27                     cc->leader_ = constant;
```

```
                }
                cc->members_.insert(x);
                cc->value_expr_ = ve;
                cc->c_ = constant;
                return pout;
            }
        }

        if(constant){
            e = constant;
        }
        auto cc = createCongruenceClass(next_value_number_++);
        cc->leader_ = e;
        cc->value_expr_ = ve;
        cc->value_phi_= vpf;
        cc->members_.insert(x);
        cc->c_ = constant;
        pout.insert(cc);

        return pout;
    }
```

ValuePhiFunc()

先判断`ve`是否具有`φk(vi1, vj1) ⊕ φk(vi2, vj2)`的形式，调用`getVP`函数找到值编码对应的`Value_phi_`，如果两边都是Phi指令的话，进行伪代码的如下后续操作：

```
// process left edge
        vi = getVN(POUTkl, vi1 ⊕ vi2)
        if vi is NULL
        then vi = valuePhiFunc(vi1 ⊕ vi2, POUTkl)
        // process right edge
        vj = getVN(POUTkr, vj1 ⊕ vj2)
        if vj is NULL
        then vj = valuePhiFunc(vj1 ⊕ vj2, POUTkr)
```

```
shared_ptr<PhiExpression> GVN::valuePhiFunc(shared_ptr<Expression> ve,
const partitions &P) {
    shared_ptr<Expression> vi_, vj_;
    shared_ptr<PhiExpression> phik;
    if(ve->get_expr_type() == Expression::e_bin) {
        auto bin = std::dynamic_pointer_cast<BinaryExpression>(ve);
```

```
6         auto op = bin->retOp();
7         auto lhs = bin->retLhs();
8         auto rhs = bin->retRhs();
9         shared_ptr<VNExpression> lhs_vn =
std::dynamic_pointer_cast<VNExpression>(lhs);
10        shared_ptr<VNExpression> rhs_vn =
std::dynamic_pointer_cast<VNExpression>(rhs);
11        auto lhs_phi = getVP(lhs_vn, P);
12        auto rhs_phi = getVP(rhs_vn, P);
13
14        if(lhs_phi and rhs_phi) {
15
16            if(lhs_phi->retLbb() != rhs_phi->retLbb()) return nullptr;
17            if(lhs_phi->retRbb() != rhs_phi->retRbb()) return nullptr;
18
19            auto poutkl = pout_[lhs_phi->retLbb()];
20            auto poutkr = pout_[lhs_phi->retRbb()];
21
22            auto v1_con = std::dynamic_pointer_cast<ConstantExpression>
(lhs_phi->retLhs());
23            auto v2_con = std::dynamic_pointer_cast<ConstantExpression>
(rhs_phi->retLhs());
24            shared_ptr<Expression> vi12;
25            int is_constant = 514;
26            if(v1_con or v2_con) {
27                is_constant = 114;
28            }
29
30            switch (is_constant) {
31            case 114:
32                return nullptr;
33                break;
34            case 514:
35                vi12 = BinaryExpression::create(bin->retOp(), lhs_phi-
>retLhs(), rhs_phi->retLhs());
36                   break;
37            default:
38                break;
39            }
40            if(!vi12) return nullptr;
41
42            vi_ = getVN(poutkl, vi12, 0);
43            if(vi_ == nullptr){
```

```
                auto vi_phi = valuePhiFunc(vi12, poutkl);
                if(vi_phi){
                    bool flag = false;
                    for(auto cc : poutkl){
                        if(cc->value_phi_ == vi_phi){
                            vi_ = VNExpression::create(cc->index_);
                            flag = true;
                            break;
                        }
                    }
                }
            }

            auto v3_con = std::dynamic_pointer_cast<ConstantExpression>(lhs_phi->retRhs());
            auto v4_con = std::dynamic_pointer_cast<ConstantExpression>(rhs_phi->retRhs());
            shared_ptr<Expression> vj12;
            is_constant = 514;
            if(v3_con or v4_con) {
                is_constant = 114;
            }

            switch (is_constant) {
            case 114:
                return nullptr;
                break;
            case 514:
                vj12 = BinaryExpression::create(bin->retOp(), lhs_phi->retRhs(), rhs_phi->retRhs());
            default:
                break;
            }
            if(!vj12) return nullptr;

            vj_ = getVN(poutkr, vj12, 0);
            if(vj_ == nullptr) {
                auto vj_phi = valuePhiFunc(vj12, poutkr);
                if(vj_phi) {
                    bool flag = false;
                    for(auto cc : poutkr){
                        if(cc->value_phi_ == vj_phi){
                            vi_ = VNExpression::create(cc->index_);
```

```
                            flag = true;
                            break;
                        }
                    }
                }
            }

            if(vi_ && vj_) {
                phik = PhiExpression::create(vi_, vj_, lhs_phi->retLbb(), rhs_phi->retRbb());
            }
        }
        return phik;

}
```

```
if vi is not NULL and vj is not NULL
then return φk(vi, vj)
else return NULL
```

优化前后的IR对比

bin.cminus

```
/* c and d are redundant, and also check for constant propagation */
int main(void) {
    int a;
    int b;
    int c;
    int d;
    if (input() > input()) {
        a = 33 + 33;
        b = 44 + 44;
        c = a + b;
    } else {
        a = 55 + 55;
        b = 66 + 66;
        c = a + b;
    }
    output(c);
    d = a + b;
```

```
    output(d);
}
```

优化前

```
define i32 @main() {
label_entry:
  %op0 = call i32 @input()
  %op1 = call i32 @input()
  %op2 = icmp sgt i32 %op0, %op1
  %op3 = zext i1 %op2 to i32
  %op4 = icmp ne i32 %op3, 0
  br i1 %op4, label %label5, label %label14
label5:                                           ; preds =
%label_entry
  %op6 = add i32 33, 33
  %op7 = add i32 44, 44
  %op8 = add i32 %op6, %op7
  br label %label9
label9:                                           ; preds =
%label5, %label14
  %op10 = phi i32 [ %op8, %label5 ], [ %op17, %label14 ]
  %op11 = phi i32 [ %op7, %label5 ], [ %op16, %label14 ]
  %op12 = phi i32 [ %op6, %label5 ], [ %op15, %label14 ]
  call void @output(i32 %op10)
  %op13 = add i32 %op12, %op11
  call void @output(i32 %op13)
  ret i32 0
label14:                                          ; preds =
%label_entry
  %op15 = add i32 55, 55
  %op16 = add i32 66, 66
  %op17 = add i32 %op15, %op16
  br label %label9
}
```

优化后

```
define i32 @main() {
label_entry:
  %op0 = call i32 @input()
  %op1 = call i32 @input()
```

```
  %op2 = icmp sgt i32 %op0, %op1
  %op3 = zext i1 %op2 to i32
  %op4 = icmp ne i32 %op3, 0
  br i1 %op4, label %label5, label %label14
label5:                                           ; preds =
%label_entry
  br label %label9
label9:                                           ; preds =
%label5, %label14
  %op10 = phi i32 [ 154, %label5 ], [ 242, %label14 ]
  call void @output(i32 %op10)
  call void @output(i32 %op10)
  ret i32 0
label14:                                           ; preds =
%label_entry
  br label %label9
}
```

可见冗余的op6 7 8 11 12 13 15 16 17被删去 对d的计算`d = a + b`因之前计算了`c = a + b`也被删去，其余的被删去的指令转化为第三个基本块的phi函数，其中也有常量传播（154和242的计算）。

思考题

1. 请简要分析你的算法复杂度

   论文 Detection of Redundant Expressions: A Complete and Polynomial-Time Algorithm in SSA 详细分析了此算法的时间复杂度：

```
Let there be n number of expressions in a program. By definitions of Join and transferFunction a partition can have O(n) classes with each class of O(v) size, where v is the number of variables and constants in the program. The join operation is class-wise intersection of partitions. With efficient data structure that supports lookup, intersection of each class takes O(v) time. With a total of n2 such intersections, a join takes O(n2.v) time. If there are j join points, the total time taken by all the join operations in an iteration is O(n2.v.j). The transfer function involves construction and lookup of value expression or value φ-function in the input partition. A value expression is computed and searched for in O(n) time. Computation of value φ-function for an expression x+y essentially involves lookup of value expressions, recursively, in partitions at left and right predecessors of a join block. If a lookup table is maintained to map value expressions to value φ-functions (or NULL when a value expression does not have a value φ-function), then computation of a value φ-function can be done in O(n.j) time. Thus transfer function of a statement x = e takes O(n.j) time. In a program with n expressions total time taken by all the transfer functions in an iteration is O(n2.j). Thus the time taken by all the joins and transfer functions in an iteration is O(n2.v.j). As shown in [4], in the worst case the iterative analysis takes n iterations and hence the total time taken by the analysis is O(n3.v.j).

[4]:4. Gulwani, S., Necula, G.C.: A polynomial-time algorithm for global value number- ing. In: Giacobazzi, R. (ed.) SAS 2004. LNCS, vol. 3148, pp. 212–227. Springer, Heidelberg (2004)
```

2. `std::shared_ptr`如果存在环形引用，则无法正确释放内存，你的 Expression 类是否存在 circular reference?

   存在，因为有phi的存在，如下：

   ```
   %op1 = phi %op2,%op3

   %op2 = add %op1, 1
   ```

3. 尽管本次实验已经写了很多代码，但是在算法上和工程上仍然可以对 GVN 进行改进，请简述你的 GVN 实现可以改进的地方

   对于GVN算法，理解原理就已经很不简单了，不知道怎么进行优化。

对于我的代码，可行的优化如下：

常量折叠可以转到`replace_cc_members()`函数中处理，这样会省去`ValueExpr`里很多不必要的逻辑。

`ValuePhiFunction`函数过于复杂，对左边和右边的处理本质上是一样的，可以写在一个函数中。

...

实验总结

了解了GVN算法的运行方式，了解并能熟悉运用重载、智能指针和模版类等C++特性，增强了自己编写C++代码的能力。

代码中的很多想法来自论坛和QQ群，尽管如此，在编写代码的过程中仍然遇到了许多问题，但通过进行GDB调试可以快速找到段错误的地方并进行DEBUG。

实验没有满分，但是最后的一点分数不知道问题出现在什么地方。

实验反馈（可选 不会评分）

框架不具备定式的结果就是在做实验的前提下无法避免要和同学助教花费大量时间讨论。当然也有此次试验是第一次出现在编译课程的缘故，实验文档比较不完善。特别是等价类的几个成员的设置在一开始极其令人迷惑。

希望助教能再完善一下框架并适当补充一两个函数作为例子，如`ValueExpr`可以帮同学先补充一条指令的实现，让同学结合补充其余的指令。

希望助教能把比较难比较偏的测试点和标答放出来，方便找到代码的BUG。