# 归结算法实验报告

## Honor Code

同王浩算法的课题，参考了互联网上对表达式进行解析的思路和数据结构。

## 1.表达式解析

与王浩算法的实现中的实现一模一样，麻烦老师批改的时候翻一下王浩算法的实验报告。

## 2.表达式化简

分三步将表达式化为一个合取范式。

第一步：消除蕴含连接词，按照蕴含等值式对表达式二叉树进行处理

如果是蕴含连接词，那么将节点换为`v`，然后拼上否定的左子树，右子树不变

```
if(treenode->val=='>') {
    treenode->val='v';
    tree* temp=new tree;
    temp->val='!';
    temp->right=treenode->left;
    treenode->left=temp;
}
```

如果是双蕴含，将节点换为`^`，左侧是左子树推出右子树，右侧是右子树推出左子树

```
else if(treenode->val=='<') {
        treenode->val='^';
        tree* temp1=new tree;
        tree* temp2=new tree;
        temp1->val='>';
        temp1->left=treenode->left;
        temp1->right=treenode->right;
        temp2->val='>';
        temp2->left=treenode->right;
        temp2->right=treenode->left;
        treenode->left=temp1;
        treenode->right=temp2;
    }
```

如果这个节点的左右子树非空，递归处理，结束之后返回根节点

```
if(treenode->left!=nullptr) {
    treenode->left=elimImp(treenode->left);
}
if(treenode->right!=nullptr) {
    treenode->right=elimImp(treenode->right);
}
return treenode;
```

第二步：否定词内移

根据DeMorgan律，将否定词内移，其中要注意双重否定的消去

```
if(treenode->val=='!') {
    if(treenode->right->val=='!') {
        treenode->val=treenode->right->right->val;
        treenode->left=treenode->right->right->left;
        treenode->right=treenode->right->right->right;
    } else if(treenode->right->val=='v') {
        treenode->val='^';
        treenode->right->val='!';
        tree* temp=new tree;
        temp->val='!';
        temp->right=treenode->right->left;
        treenode->left=temp;
        treenode->right->left=nullptr;
    } else if(treenode->right->val=='^') {
        treenode->val='v';
        treenode->right->val='!';
        tree* temp=new tree;
        temp->val='!';
        temp->left=nullptr;
        temp->right=treenode->right->left;
        treenode->left=temp;
        treenode->right->left=nullptr;
    }
}
```

递归处理左右子树

```
if(treenode->left!=nullptr) {
    treenode->left=pushNeg(treenode->left);
}
if(treenode->right!=nullptr) {
    treenode->right=pushNeg(treenode->right);
}
return treenode;
```

第三步：利用分配律将表达式化简为范式。寻找所有或，考虑左右，使用分配律，然后递归处理。

```
tree* distriLaw(tree* treenode) {
    if(treenode->val=='v') {
        if(treenode->left!=nullptr) {
            if(treenode->left->val=='^') {
                treenode->val='^';
                tree* temp1=new tree;
                tree* temp2=new tree;
                temp1->val='v';
                temp1->left=treenode->left->left;
                temp1->right=treenode->right;
                temp2->val='v';
                temp2->left=treenode->left->right;
                temp2->right=treenode->right;
                treenode->left=temp1;
                treenode->right=temp2;
            }
        }
        if(treenode->right!=nullptr) {
            if(treenode->right->val=='^') {
                treenode->val='^';
                tree* temp1=new tree;
                tree* temp2=new tree;
                temp1->val='v';
                temp1->left=treenode->left;
                temp1->right=treenode->right->left;
                temp2->val='v';
                temp2->left=treenode->left;
                temp2->right=treenode->right->right;
                treenode->left=temp1;
                treenode->right=temp2;
            }
        }
    }
    if(treenode->left!=nullptr) {
        treenode->left=distriLaw(treenode->left);
    }
    if(treenode->right!=nullptr) {
        treenode->right=distriLaw(treenode->right);
    }
    return treenode;
}
```

## 3.提取子句集

引入`vector<set<atom>> clauses`表示子句集合

其中`atom`定义为，表示一个原子命题

```
struct atom {
    char val;
    bool neg=false;

    //重载set使用到的的比较运算符，根据val的字典序排序
    bool operator<(const atom& other) const {
        if (val != other.val) {
            return val<other.val;
        }
        return neg>other.neg;
    }
};
```

对根节点建立字句集，函数原型为`void constructClauses(tree* root,vector<set<atom>> clauses)`

如果是`^`，则对两个子树分别建立子句集，放入vector中

```
if(root->val=='^') {
    constructClauses(root->left,clauses);
    constructClauses(root->right,clauses);
}
```

如果是`v`，对两个子树分别找子句集，然后放入同一个set中，再放入vector

```
else if(root->val=='v') {
    set<atom> clause;
    vector<set<atom>> leftclauses;
    vector<set<atom>> rightclauses;
    constructClauses(root->left,leftclauses);
    constructClauses(root->right,rightclauses);
    if(!leftclauses.empty()) clause.insert(leftclauses[0].begin(), leftclauses[0].end());
    if(!rightclauses.empty()) clause.insert(rightclauses[0].begin(),
rightclauses[0].end());
    clauses.push_back(clause);
}
```

如果是根节点的`val`是原子命题，新建一个子句，然后放入子句集

```
else {
    set<atom> clause;
    atom tempatom;
    if(root->val=='!') {
        tempatom.neg=true;
        tempatom.val=root->right->val;
    } else {
        tempatom.val=root->val;
    }
    clause.insert(tempatom);
    clauses.push_back(clause);
}
```

到达`nullptr`的递归边界后返回

```
if(root==nullptr) {
    return;
}
```

## 4.归结推理

遍历子句中的所有原子命题，判断子句是否可以归结，记录可以归结的原子命题`find`。

```
bool ifCanResolve(set<atom>& clause1,set<atom>& clause2,atom& find) {
    for(auto it=clause1.begin();it!=clause1.end();++it) {
        atom negIt;
        negIt.neg=(find.neg ? false : true);
        negIt.val=it->val;
        if(clause2.find(negIt)!=clause2.end()) {
            find=*it;
            return true;
        }
    }
    return false;
}
```

根据找到的可以归结的原子命题，归结出新子句。首先对子句1擦除可以归结的原子命题，然后插入子句2中的原子命题

```
set<atom> resolve(set<atom>& clause1,set<atom>& clause2,atom& find) {
    set<atom> result=clause1;
    result.erase(find);
    for(auto it=clause2.begin();it!=clause2.end();++it) {
        if(find.neg) {
            if(!(!it->neg&&it->val==find.val)) {
                result.insert(*it);
            }
        } else {
            if(!(it->neg&&it->val==find.val)) {
                result.insert(*it);
            }
        }
    }
    return result;
}
```

引入`set<set<atom>> beenInVector`，表示已经放在子句集中的子句

通过两重循环遍历所有子句，进行归结推理

如果能够找到，进行一次归结，判断归结出的子句是否为空，如果为空，`return true`，找到空子句，命题正确。

反之如果不为空，判断是否已经出现过，如果没有出现过，就加入子句集，同时把`newClauseFound`改为真

```
for(int i=0;i<clauses.size();++i) {
    for(int j=0;j<clauses.size();++j) {
        if(j==i) continue;
        atom find;
        if(ifCanResolve(clauses[i],clauses[j],find)) {
            set<atom> newClause=resolve(clauses[i],clauses[j],find);
            if (newClause.empty()) {
                return true;
            }
            if (beenInVector.find(newClause)==beenInVector.end()) {
                clauses.push_back(newClause);
                beenInVector.insert(newClause);
                newClauseFound=true;
            }
        }
    }
}
```

如果最后不能找到新子句，归结无法进行，`return false`，命题无法证明

```
if (clauses.size()==iniSize) {
    return false;
}
```

## 5.输入处理逻辑

与王浩算法不同，归结推理是直接处理推理的，我们不能要求用户输入一个完整的命题，因此我们采用如下输入方式。

我们首先让用户输入条件，在真实情景中，条件可以有很多个。同时用户可能想判断一个命题是不是重言式，因此可以不输入条件，直接在之后输入结论。用户输入结束之后程序`cout<<"Continue or END:";`用户可以继续输入，或者输入`END`来结束输入。

我们引入变量`emptyReturn`，对每一个条件输入，如果确实是一个空的输入，那我们不认为这是一个错误输入，直接放过。如果不是一个空的条件输入，但是却返回了一个空子句，那么表明这是一个错误输入，程序终止。

我们引入变量`notEmptyReturn`，只要有不是一个空的条件输入，那么我们就把这变量的值改为真。

第一次输入时，也就是说`preconditionInput`为空，那么直接把输入赋给`preconditionInput`。反之，如果不为空，那么我们需要把已经有的条件和新输入的条件拼起来。

然后让用户输入结论，与条件不同，结论有且只有一个。这里我们采取与王浩算法完全相同的错误判定方法，直接根据返回的树是否是空指针来判断输入的正确性，如果输入错误，程序直接`cout<<"Empty Input or Not Well-Formed Formula";`，然后返回。

接下来程序需要将条件和结论拼起来，即`条件 ^ !结论`，考虑到用户可能想直接证明是不是重言式，如果`notEmptyReturn`为假，表示没有输入任何一个条件，归结中要用到的命题就直接是`! 结论`。

最后程序就调用`simplify`函数进行化简，其中依次调用`elimImp`，`pushNeg`，`distriLaw`进行化简。并用`constructClauses`从化简的归结使用的命题中提取子句集。

最后的最后，进行归结推理。

## 6.实验结果

- 输入测试
  - 错误的命题输入，与王浩算法一样，我们就不再这里具体测试了，麻烦老师批改的时候翻一下我的王浩算法的实验报告
  - 空条件输入的测试

    ```
    Please enter precondition:END
    Please enter conclusion:(a->B)->((c->D)->(a^c->B^D))
    Conclusion Constructed:
    (a->B)->((c->D)->((a^c)->(B^D)))
    ```

- 重言式（正确的命题）

  这里采用与王浩算法相同的测例，由于不用输出化简过程，所以显得非常简练

  ```
  Please enter precondition:END
  Please enter conclusion:(R^(!(P->Q)->!(RvS))^((Q->P)v!R))->(P<->Q)
  Conclusion Constructed:
  ((R^((!(P->Q))->(!(RvS))))^((Q->P)v(!R)))->(P<->Q)
  Proposition Used in Revolution:
  ((R^((((!P)vQ)v(!R))^(((!P)vQ)v(!S))))^(((!Q)vP)v(!R)))^((((!P)vQ)vQ)^(((!P)vQ)v(!P))
  )
  Simplified Version:
  ((R^((((!P)vQ)v(!R))^(((!P)vQ)v(!S))))^(((!Q)vP)v(!R)))^((((!P)vQ)vQ)^(((!P)vQ)v(!P))
  )
  Clauses Constructed:
  {R}
  {!P,Q,!R}
  {!P,Q,!S}
  {P,!Q,!R}
  {!P,Q}
  {!P,Q}
  Empty clause found. It's true!
  ```

```
Please enter precondition:END
Please enter conclusion:((P->(Q->R))^(Q->(R->A)))->(P->(Q->A))
Conclusion Constructed:
((P->(Q->R))^(Q->(R->A)))->(P->(Q->A))
Proposition Used in Revolution:
(((!P)v((!Q)vR))^((!Q)v((!R)vA)))^(P^(Q^(!A)))
Simplified Version:
(((!P)v((!Q)vR))^((!Q)v((!R)vA)))^(P^(Q^(!A)))
Clauses Constructed:
{!P,!Q,R}
{A,!Q,!R}
{P}
{Q}
{!A}
Empty clause found. It's true!
```

```
Please enter precondition:END
Please enter conclusion:((S->!Q)^(P->Q)^(RvS)^(R->!Q))->!P
Conclusion Constructed:
((((S->(!Q))^(P->Q))^(RvS))^(R->(!Q)))->(!P)
Proposition Used in Revolution:
(((((!S)v(!Q))^((!P)vQ))^(RvS))^((!R)v(!Q)))^P
Simplified Version:
(((((!S)v(!Q))^((!P)vQ))^(RvS))^((!R)v(!Q)))^P
Clauses Constructed:
{!Q,!S}
{!P,Q}
{R,S}
{!Q,!R}
{P}
Empty clause found. It's true!
```

```
Please enter precondition:END
Please enter conclusion:(a->B)->((c->D)->(a^c->B^D))
Conclusion Constructed:
(a->B)->((c->D)->((a^c)->(B^D)))
Proposition Used in Revolution:
((!a)vB)^(((!c)vD)^((a^c)^(Bv(!D))))
Simplified Version:
((!a)vB)^(((!c)vD)^((a^c)^(Bv(!D))))
Clauses Constructed:
{B,!a}
{D,!c}
{a}
{c}
{B,!D}
Empty clause found. It's true!
```

```
Please enter precondition:END
Please enter conclusion:((!Q->R)^(R->P)^(P^p)^(!P^!p)^(!P->Q))->Q
Conclusion Constructed:
((((((!Q)->R)^(R->P))^(P^p))^((!P)^(!p)))^((!P)->Q))->Q
Proposition Used in Revolution:
(((((QvR)^((!R)vP))^(P^p))^((!P)^(!p)))^(PvQ))^(!Q)
Simplified Version:
(((((QvR)^((!R)vP))^(P^p))^((!P)^(!p)))^(PvQ))^(!Q)
Clauses Constructed:
{Q,R}
{P,!R}
{P}
{p}
{!P}
{!p}
{P,Q}
{!Q}
Empty clause found. It's true!
```

- 非重言式（不能证明的命题）

```
Please enter precondition:END
Please enter conclusion:((!Q->R)^(R->P)^(P^p)^(!P^!p)^(!P->Q))<->Q
Conclusion Constructed:
((((((!Q)->R)^(R->P))^(P^p))^((!P)^(!p)))^((!P)->Q))<->Q
Proposition Used in Revolution:
((((Pvp)vQ)^((Pvp)v((((((((!Q)vR)v((!P)v(!p)))v(Pvp))v(!P))^(((((!Q)v(!P))v((!P)v(!p)
))v(Pvp))v(!P)))^((((((!R)vR)v((!P)v(!p)))v(Pvp))v(!P))^(((((!R)v(!P))v((!P)v(!p)))v(
Pvp))v(!P))))^(((((((!Q)vR)v((!P)v(!p)))v(Pvp))v(!Q))^(((((!Q)v(!P))v((!P)v(!p)))v(Pv
p))v(!Q)))^((((((!R)vR)v((!P)v(!p)))v(Pvp))v(!Q))^(((((!R)v(!P))v((!P)v(!p)))v(Pvp))v
(!Q)))))))^(((!Q)vQ)^(((!Q)v(((((((!Q)vR)v((!P)v(!p)))v(Pvp))v(!P))^(((((!Q)v(!P))v((
!P)v(!p)))v(Pvp))v(!P)))^((((((!R)vR)v((!P)v(!p)))v(Pvp))v(!P))^(((((!R)v(!P))v((!P)v
(!p)))v(Pvp))v(!P)))))^((!Q)v(((((((!Q)vR)v((!P)v(!p)))v(Pvp))v(!Q))^(((((!Q)v(!P))v(
(!P)v(!p)))v(Pvp))v(!Q)))^((((((!R)vR)v((!P)v(!p)))v(Pvp))v(!Q))^(((((!R)v(!P))v((!P)
v(!p)))v(Pvp))v(!Q))))))))^(((!Q)vQ)^(((!Q)v(((((((!Q)vR)v((!P)v(!p)))v(Pvp))v(!P))^(
((((!Q)v(!P))v((!P)v(!p)))v(Pvp))v(!P)))^((((((!R)vR)v((!P)v(!p)))v(Pvp))v(!P))^(((((
!R)v(!P))v((!P)v(!p)))v(Pvp))v(!P)))))^((((!Q)v(((((!Q)vR)v((!P)v(!p)))v(Pvp))v(!Q)))
^((!Q)v(((((!Q)v(!P))v((!P)v(!p)))v(Pvp))v(!Q))))^(((!Q)v(((((!R)vR)v((!P)v(!p)))v(Pv
p))v(!Q)))^((!Q)v(((((!R)v(!P))v((!P)v(!p)))v(Pvp))v(!Q)))))))
Simplified Version:
```

```
((((Pvp)vQ)^((Pvp)v(((((((((!Q)vR)v((!P)v(!p)))v(Pvp))v(!P))^(((((!Q)v(!P))v((!P)v(!p)
))v(Pvp))v(!P)))^((((((!R)vR)v((!P)v(!p)))v(Pvp))v(!P))^(((((!R)v(!P))v((!P)v(!p)))v(
Pvp))v(!P))))^((((((!Q)vR)v((!P)v(!p)))v(Pvp))v(!Q))^(((((!Q)v(!P))v((!P)v(!p)))v(Pv
p))v(!Q)))^((((((!R)vR)v((!P)v(!p)))v(Pvp))v(!Q))^(((((!R)v(!P))v((!P)v(!p)))v(Pvp))v
(!Q)))))))^(((!Q)vQ)^(((!Q)v((((((!Q)vR)v((!P)v(!p)))v(Pvp))v(!P))^(((((!Q)v(!P))v((
!P)v(!p)))v(Pvp))v(!P)))^((((((!R)vR)v((!P)v(!p)))v(Pvp))v(!P))^(((((!R)v(!P))v((!P)v
(!p)))v(Pvp))v(!P)))))^((!Q)v((((((!Q)vR)v((!P)v(!p)))v(Pvp))v(!Q))^(((((!Q)v(!P))v(
(!P)v(!p)))v(Pvp))v(!Q)))^((((((!R)vR)v((!P)v(!p)))v(Pvp))v(!Q))^(((((!R)v(!P))v((!P)
v(!p)))v(Pvp))v(!Q))))))))^(((!Q)vQ)^(((!Q)v((((((!Q)vR)v((!P)v(!p)))v(Pvp))v(!P))^(
((((!Q)v(!P))v((!P)v(!p)))v(Pvp))v(!P)))^((((((!R)vR)v((!P)v(!p)))v(Pvp))v(!P))^(((((
!R)v(!P))v((!P)v(!p)))v(Pvp))v(!P)))))^((((!Q)v(((((!Q)vR)v((!P)v(!p)))v(Pvp))v(!Q)))
^((!Q)v(((((!Q)v(!P))v((!P)v(!p)))v(Pvp))v(!Q))))^(((!Q)v(((((!R)vR)v((!P)v(!p)))v(Pv
p))v(!Q)))^((!Q)v(((((!R)v(!P))v((!P)v(!p)))v(Pvp))v(!Q)))))))
Clauses Constructed:
{P,Q,p}
{!P,P,!Q,R,!p,p}
{!Q,Q}
{!P,P,!Q,R,!p,p}
{!P,P,!Q,R,!p,p}
{!Q,Q}
{!P,P,!Q,R,!p,p}
{!P,P,!Q,R,!p,p}
{!P,P,!Q,!p,p}
{!P,P,!Q,!R,R,!p,p}
{!P,P,!Q,!R,!p,p}
Resolution failed. Can't be proved.
```

```
Please enter precondition:END
Please enter conclusion: ((a->b)^(c->d)^(!bv!d)<->(!av!c))v((A<->B)<->((C<->D)->((A<-
>C)->(B<->D))))
Conclusion Constructed:
((((a->b)^(c->d))^((!b)v(!d)))<->((!a)v(!c)))v((A<->B)<->((C<->D)->((A<->C)->(B<-
>D))))
Proposition Used in Revolution:
```

```
(((((cvc)^(cv((((avc)vb)^((av(!d))vb))^((((!b)vc)vb)^(((!b)v(!d))vb)))^((((avc)vd)^(
(av(!d))vd))^((((!b)vc)vd)^(((!b)v(!d))vd))))))^(((!d)vc)^(((!d)v((((avc)vb)^((av(!d)
)vb))^((((!b)vc)vb)^(((!b)v(!d))vb))))^((((!d)v((avc)vd))^((!d)v((av(!d))vd)))^(((!d)
v(((!b)vc)vd))^((!d)v(((!b)v(!d))vd)))))))^((dvc)^((((dv((avc)vb))^(dv((av(!d))vb)))^
((dv(((!b)vc)vb))^(dv(((!b)v(!d))vb))))^(((dv((avc)vd))^(dv((av(!d))vd)))^((dv(((!b)v
c)vd))^(dv(((!b)v(!d))vd)))))))^(((avc)^((((av((avc)vb))^(av((av(!d))vb)))^((av(((!b)
vc)vb))^(av(((!b)v(!d))vb))))^(((av((avc)vd))^(av((av(!d))vd)))^((av(((!b)vc)vd))^(av
(((!b)v(!d))vd))))))^((cvc)^((((cv((avc)vb))^(cv((av(!d))vb)))^((cv(((!b)vc)vb))^(cv(
((!b)v(!d))vb))))^(((cv((avc)vd))^(cv((av(!d))vd)))^((cv(((!b)vc)vd))^(cv(((!b)v(!d))
vd))))))))^(((((((!B)v((!A)vC))^((!B)v((!C)vA)))^((!B)v(((BvD)^(Bv(!B)))^(((!D)vD)^((
!D)v(!B))))))^((!B)v(((AvB)^(Av(!A)))^(((!B)vB)^((!B)v(!A))))))^(((((!A)v((!A)vC))^((
!A)v((!C)vA)))^((((!A)v(BvD))^((!A)v(Bv(!B))))^(((!A)v((!D)vD))^((!A)v((!D)v(!B))))))
^((((!A)v(AvB))^((!A)v(Av(!A))))^(((!A)v((!B)vB))^((!A)v((!B)v(!A)))))))^((((((((!C)v
D)v((!A)vC))^(((!C)vD)v((!C)vA)))^(((((!C)vD)v(BvD))^(((!C)vD)v(Bv(!B))))^((((!C)vD)v
((!D)vD))^(((!C)vD)v((!D)v(!B))))))^(((((!C)vD)v(AvB))^(((!C)vD)v(Av(!A))))^((((!C)vD
)v((!B)vB))^(((!C)vD)v((!B)v(!A))))))^((((((!D)vC)v((!A)vC))^(((!D)vC)v((!C)vA)))^(((
((!D)vC)v(BvD))^(((!D)vC)v(Bv(!B))))^((((!D)vC)v((!D)vD))^(((!D)vC)v((!D)v(!B))))))^(
((((!D)vC)v(AvB))^(((!D)vC)v(Av(!A))))^((((!D)vC)v((!B)vB))^(((!D)vC)v((!B)v(!A))))))
)^((((((((!A)vC)v((!A)vC))^(((!A)vC)v((!C)vA)))^(((((!A)vC)v(BvD))^(((!A)vC)v(Bv(!B))
))^((((!A)vC)v((!D)vD))^(((!A)vC)v((!D)v(!B))))))^(((((!A)vC)v(AvB))^(((!A)vC)v(Av(!A
))))^((((!A)vC)v((!B)vB))^(((!A)vC)v((!B)v(!A))))))^((((((!C)vA)v((!A)vC))^(((!C)vA)v
((!C)vA)))^(((((!C)vA)v(BvD))^(((!C)vA)v(Bv(!B))))^((((!C)vA)v((!D)vD))^(((!C)vA)v((!
D)v(!B))))))^(((((!C)vA)v(AvB))^(((!C)vA)v(Av(!A))))^((((!C)vA)v((!B)vB))^(((!C)vA)v(
(!B)v(!A)))))))^((((((BvD)v((!A)vC))^((BvD)v((!C)vA)))^((((BvD)v(BvD))^((BvD)v(Bv(!B
))))^(((BvD)v((!D)vD))^((BvD)v((!D)v(!B))))))^((((BvD)v(AvB))^((BvD)v(Av(!A))))^(((Bv
D)v((!B)vB))^((BvD)v((!B)v(!A))))))^(((((Bv(!B))v((!A)vC))^((Bv(!B))v((!C)vA)))^((((B
v(!B))v(BvD))^((Bv(!B))v(Bv(!B))))^(((Bv(!B))v((!D)vD))^((Bv(!B))v((!D)v(!B))))))^(((
(Bv(!B))v(AvB))^((Bv(!B))v(Av(!A))))^(((Bv(!B))v((!B)vB))^((Bv(!B))v((!B)v(!A))))))^
(((((((!D)vD)v((!A)vC))^(((!D)vD)v((!C)vA)))^(((((!D)vD)v(BvD))^(((!D)vD)v(Bv(!B))))^
((((!D)vD)v((!D)vD))^(((!D)vD)v((!D)v(!B))))))^(((((!D)vD)v(AvB))^(((!D)vD)v(Av(!A)))
)^((((!D)vD)v((!B)vB))^(((!D)vD)v((!B)v(!A))))))^((((((!D)v(!B))v((!A)vC))^(((!D)v(!B
))v((!C)vA)))^(((((!D)v(!B))v(BvD))^(((!D)v(!B))v(Bv(!B))))^((((!D)v(!B))v((!D)vD))^(
((!D)v(!B))v((!D)v(!B))))))^(((((!D)v(!B))v(AvB))^(((!D)v(!B))v(Av(!A))))^((((!D)v(!B
))v((!B)vB))^(((!D)v(!B))v((!B)v(!A))))))))))
Simplified Version:
```

```
(((((cvc)^(cv(((((avc)vb)^((av(!d))vb))^((((!b)vc)vb)^(((!b)v(!d))vb)))^((((avc)vd)^(
(av(!d))vd))^((((!b)vc)vd)^(((!b)v(!d))vd))))))^(((!d)vc)^(((!d)v((((avc)vb)^((av(!d)
)vb))^((((!b)vc)vb)^(((!b)v(!d))vb))))^((((!d)v((avc)vd))^((!d)v((av(!d))vd)))^(((!d)
v(((!b)vc)vd))^((!d)v(((!b)v(!d))vd)))))))^((dvc)^((((dv((avc)vb))^(dv((av(!d))vb)))^
((dv(((!b)vc)vb))^(dv(((!b)v(!d))vb))))^(((dv((avc)vd))^(dv((av(!d))vd)))^((dv(((!b)v
c)vd))^(dv(((!b)v(!d))vd)))))))^(((avc)^((((av((avc)vb))^(av((av(!d))vb)))^((av(((!b)
vc)vb))^(av(((!b)v(!d))vb))))^(((av((avc)vd))^(av((av(!d))vd)))^((av(((!b)vc)vd))^(av
(((!b)v(!d))vd))))))^((cvc)^((((cv((avc)vb))^(cv((av(!d))vb)))^((cv(((!b)vc)vb))^(cv(
((!b)v(!d))vb))))^(((cv((avc)vd))^(cv((av(!d))vd)))^((cv(((!b)vc)vd))^(cv(((!b)v(!d))
vd)))))))^(((((((!B)v((!A)vC))^((!B)v((!C)vA)))^((!B)v(((BvD)^(Bv(!B)))^(((!D)vD)^((
!D)v(!B))))))^((!B)v(((AvB)^(Av(!A)))^(((!B)vB)^((!B)v(!A))))))^(((((!A)v((!A)vC))^((
!A)v((!C)vA)))^((((!A)v(BvD))^((!A)v(Bv(!B))))^(((!A)v((!D)vD))^((!A)v((!D)v(!B)))))
^((((!A)v(AvB))^((!A)v(Av(!A))))^(((!A)v((!B)vB))^((!A)v((!B)v(!A)))))))^((((((((!C)v
D)v((!A)vC))^(((!C)vD)v((!C)vA)))^(((((!C)vD)v(BvD))^(((!C)vD)v(Bv(!B))))^((((!C)vD)v
((!D)vD))^(((!C)vD)v((!D)v(!B))))))^(((((!C)vD)v(AvB))^(((!C)vD)v(Av(!A))))^((((!C)vD
)v((!B)vB))^(((!C)vD)v((!B)v(!A))))))^((((((!D)vC)v((!A)vC))^(((!D)vC)v((!C)vA)))^(((
((!D)vC)v(BvD))^(((!D)vC)v(Bv(!B))))^((((!D)vC)v((!D)vD))^(((!D)vC)v((!D)v(!B))))))^(
((((!D)vC)v(AvB))^(((!D)vC)v(Av(!A))))^((((!D)vC)v((!B)vB))^(((!D)vC)v((!B)v(!A)))))
)^((((((((!A)vC)v((!A)vC))^(((!A)vC)v((!C)vA)))^(((((!A)vC)v(BvD))^(((!A)vC)v(Bv(!B))
))^((((!A)vC)v((!D)vD))^(((!A)vC)v((!D)v(!B))))))^(((((!A)vC)v(AvB))^(((!A)vC)v(Av(!A
))))^((((!A)vC)v((!B)vB))^(((!A)vC)v((!B)v(!A))))))^((((((!C)vA)v((!A)vC))^(((!C)vA)v
((!C)vA)))^(((((!C)vA)v(BvD))^(((!C)vA)v(Bv(!B))))^((((!C)vA)v((!D)vD))^(((!C)vA)v((!
D)v(!B))))))^(((((!C)vA)v(AvB))^(((!C)vA)v(Av(!A))))^((((!C)vA)v((!B)vB))^(((!C)vA)v(
(!B)v(!A)))))))^((((((BvD)v((!A)vC))^((BvD)v((!C)vA)))^((((BvD)v(BvD))^((BvD)v(Bv(!B
))))^(((BvD)v((!D)vD))^((BvD)v((!D)v(!B))))))^((((BvD)v(AvB))^((BvD)v(Av(!A))))^(((Bv
D)v((!B)vB))^((BvD)v((!B)v(!A))))))^(((((Bv(!B))v((!A)vC))^((Bv(!B))v((!C)vA)))^((((B
v(!B))v(BvD))^((Bv(!B))v(Bv(!B))))^(((Bv(!B))v((!D)vD))^((Bv(!B))v((!D)v(!B))))))^(((
(Bv(!B))v(AvB))^((Bv(!B))v(Av(!A))))^(((Bv(!B))v((!B)vB))^((Bv(!B))v((!B)v(!A)))))))^
(((((((!D)vD)v((!A)vC))^(((!D)vD)v((!C)vA)))^(((((!D)vD)v(BvD))^(((!D)vD)v(Bv(!B))))^
((((!D)vD)v((!D)vD))^(((!D)vD)v((!D)v(!B))))))^(((((!D)vD)v(AvB))^(((!D)vD)v(Av(!A)))
)^((((!D)vD)v((!B)vB))^(((!D)vD)v((!B)v(!A))))))^((((((!D)v(!B))v((!A)vC))^(((!D)v(!B
))v((!C)vA)))^(((((!D)v(!B))v(BvD))^(((!D)v(!B))v(Bv(!B))))^((((!D)v(!B))v((!D)vD))^(
((!D)v(!B))v((!D)v(!B))))))^(((((!D)v(!B))v(AvB))^(((!D)v(!B))v(Av(!A))))^((((!D)v(!B
))v((!B)vB))^(((!D)v(!B))v((!B)v(!A))))))))))
Clauses Constructed:
{c}
{a,b,c}
{c,!d}
{a,b,c,!d}
{a,c,!d,d}
{a,!d,d}
{!b,c,!d,d}
{!b,!d,d}
{c,d}
{a,b,c,d}
{a,b,!d,d}
{!b,b,c,d}
{!b,b,!d,d}
{a,c,d}
{a,!d,d}
{!b,c,d}
{!b,!d,d}
```

```
{a,c}
{a,b,c}
{a,b,!d}
{a,!b,b,c}
{a,!b,b,!d}
{a,c,d}
{a,!d,d}
{a,!b,c,d}
{a,!b,!d,d}
{c}
{a,b,c}
{a,b,c,!d}
{!b,b,c}
{!b,b,c,!d}
{a,c,d}
{a,c,!d,d}
{!b,c,d}
{!b,c,!d,d}
{!A,!B,C}
{A,!B,!C}
{!B,B,D}
{A,!B,B}
{!A,C}
{!A,A,!C}
{!A,B,D}
{!A,!B,B}
{!A,!D,D}
{!A,!B,!D}
{!A,A,B}
{!A,A}
{!A,!B,B}
{!A,!B}
{!A,!C,C,D}
{A,!C,D}
{B,!C,D}
{!B,B,!C,D}
{!C,!D,D}
{!B,!C,!D,D}
{A,B,!C,D}
{!A,A,!C,D}
{!B,B,!C,D}
{!A,!B,!C,D}
{!A,C,!D}
{A,!C,C,!D}
{B,C,!D,D}
{!B,B,C,!D}
{C,!D,D}
{!B,C,!D}
{A,B,C,!D}
{!A,A,C,!D}
{!B,B,C,!D}
{!A,!B,C,!D}
```

```
{!A,C}
{!A,A,!C,C}
{!A,B,C,D}
{!A,!B,B,C}
{!A,C,!D,D}
{!A,!B,C,!D}
{!A,A,B,C}
{!A,A,C}
{!A,!B,B,C}
{!A,!B,C}
{!A,A,!C,C}
{A,!C}
{A,B,!C,D}
{A,!B,B,!C}
{A,!C,!D,D}
{A,!B,!C,!D}
{A,B,!C}
{!A,A,!C}
{A,!B,B,!C}
{!A,A,!B,!C}
{!A,B,C,D}
{A,B,!C,D}
{B,D}
{!B,B,D}
{B,!D,D}
{!B,B,!D,D}
{A,B,D}
{!A,A,B,D}
{!B,B,D}
{!A,!B,B,D}
{!A,!B,B,C}
{A,!B,B,!C}
{!B,B,D}
{!B,B}
{!B,B,!D,D}
{!B,B,!D}
{A,!B,B}
{!A,A,!B,B}
{!B,B}
{!A,!B,B}
{!A,C,!D,D}
{A,!C,!D,D}
{B,!D,D}
{!B,B,!D,D}
{!D,D}
{!B,!D,D}
{A,B,!D,D}
{!A,A,!D,D}
{!B,B,!D,D}
{!A,!B,!D,D}
{!A,!B,C,!D}
{A,!B,!C,!D}
```

```
{!B,B,!D,D}
{!B,B,!D}
{!B,!D,D}
{!B,!D}
{A,!B,B,!D}
{!A,A,!B,!D}
{!B,B,!D}
{!A,!B,!D}
Resolution failed. Can't be proved.
```