# 等值式证明：

对于这个题目我的结题思路如下：

大前提：将所有->和<-->消除再进行比较

1. 先遍历两个带证明的等值式，判断是否含有->或者<-->有的进行标记
2. 将两公式中缀表达式转为后缀表达式（处理优先级的问题）
3. 然后对带有标记的公式将->和<-->消除，由于公式被转化成了后缀表达式，等值式模式的转换只需要在原后缀表达式的运算符位置进行字符的替换即可。例如p->q,转后缀成了pq->,然后变换成p!q|(p|q代表pvq),因为后缀表达式可以通过对运算符的目数来向前确定运算数。

例如：

r^p->q的后缀表达式rp^q->

因为运算数可以被运算符绑架在一起，则对于->而言（rp^）与q是它的运算数

运用公式得(rp^)!q|。

这里附带转后缀核心代码：

int Degree(char c) {//定义优先级

if (c == '(') return 6;

if (c == '!') return 5;

if (c == '^') return 4; // 相当于q^p

if (c == '|') return 3; // 相当于qVp，优先级最低

if (c == '>') return 2;

if (c == '=') return 1;

else

return 0;

}

string MidToPost(string c) { //求表达式对应的后缀表达式

stack<char> s; //字符串中去手动去空格

string q="";

int n = c.size() ;

for (int i = 0; i<n; i++) {

if (c[i] != ' ') { // 除去空格

// 如果遇到运算数，直接加入到队列中，用队列来放后缀表达式

if (c[i] == '0' || c[i] == '1') {

q += c[i];

}

else if (c[i] == '!'&&!s.empty() && s.top() == '!') {

s.pop(); // 如果遇到！而且栈顶也是!那么直接抵消出栈

}

else if (!s.size()) {

s.push(c[i]);// 如果栈为空，遇到运算符直接入栈

}

else if (c[i] == ')') {

// 如果是右括号，则弹出对应左括号前的所有的运算符 ，加入到队列中

while (s.top() != '(') {

q += s.top();

s.pop();

}

s.pop(); // 弹出左括号

continue;

}

else if (Degree(s.top()) == 6 || (Degree(c[i]) > Degree(s.top()))) {

s.push(c[i]); // 如果栈顶是左括号，或者当前优先级高，都入栈

}

else if (Degree(s.top()) != 6 && Degree(c[i]) <= Degree(s.top())) {

q += s.top();

s.pop(); // 如果遇到运算符没有栈顶运算符级别高，出栈

while (!s.empty() && Degree(s.top()) != 6 && Degree(c[i]) <= Degree(s.top())) {

q+=s.top(); // 从栈中弹出比当前优先级高的运算符

s.pop();

}

s.push(c[i]); //将当前运算符加入到队列

}

}

}

while (!s.empty()) {

q += s.top(); // 最后将栈里面所有元素弹出加入到队列

s.pop();

}

return q;

}

1. 将->和<->消除后我们需要把合取永真式和析取永假式消除如,(!pp|^)、（!pp^|）。
2. 遇到问题：

如何准确消除永假永真。

消除永真永假之后后缀表达式不同该如何变换。

因为直接推导很多问题解决不了，所以我完成了利用真值表比较法，可以处理亚元问题。

代码如下：

Head.h文件

#ifndef HEAD

#define HEAD

#include<iostream>

#include<cstdio>

#include<stack>

#include<queue>

#include<cstring>

#include<string>

#include<cstdlib>

#include<fstream>

#define N 200

using namespace std;

int Degree(char c); // 求表达式的优先级

bool Calc(char c); // 逻辑表达式数的转换

string MidToPost(string c); // 求表达式对应的后缀表达式

char GetValuePost(string c);

bool conditionalS(bool a,bool b);

bool Equal(bool a,bool b);

int Count(string,int m[]);

int turnNumber(int,int);

#endif

Subfunction.cpp文件

#include"head.h"

//处理后缀表达式

int Degree(char c) {

if (c == '(') return 6;

if (c == '!') return 5;

if (c == '^') return 4; // 相当于q^p

if (c == '|') return 3; // 相当于qVp，优先级最低

if (c == '>') return 2;

if (c == '=') return 1;

else

return 0;

}

string MidToPost(string c) { //求表达式对应的后缀表达式

stack<char> s; //字符串中去手动去空格

string q="";

int n = c.size() ;

for (int i = 0; i<n; i++) {

if (c[i] != ' ') { // 除去空格

// 如果遇到运算数，直接加入到队列中，用队列来放后缀表达式

if (c[i] == '0' || c[i] == '1') {

q += c[i];

}

else if (c[i] == '!'&&!s.empty() && s.top() == '!') {

s.pop(); // 如果遇到！而且栈顶也是!那么直接抵消出栈

}

else if (!s.size()) {

s.push(c[i]);// 如果栈为空，遇到运算符直接入栈

}

else if (c[i] == ')') {

// 如果是右括号，则弹出对应左括号前的所有的运算符 ，加入到队列中

while (s.top() != '(') {

q += s.top();

s.pop();

}

s.pop(); // 弹出左括号

continue;

}

else if (Degree(s.top()) == 6 || (Degree(c[i]) > Degree(s.top()))) {

s.push(c[i]); // 如果栈顶是左括号，或者当前优先级高，都入栈

}

else if (Degree(s.top()) != 6 && Degree(c[i]) <= Degree(s.top())) {

q += s.top();

s.pop(); // 如果遇到运算符没有栈顶运算符级别高，出栈

while (!s.empty() && Degree(s.top()) != 6 && Degree(c[i]) <= Degree(s.top())) {

q+=s.top(); // 从栈中弹出比当前优先级高的运算符

s.pop();

}

s.push(c[i]); //将当前运算符加入到队列

}

}

}

while (!s.empty()) {

q += s.top(); // 最后将栈里面所有元素弹出加入到队列

s.pop();

}

return q;

}

Count.cpp文件

#include"head.h"

//计算运算数的个数并打印表格

int Count(string s,int m[]){

int ch[26],value[30],Count=0;//value用于赋值0 or 1

memset(ch,0,sizeof(ch));

memset(value,5,sizeof(value));//初始化 赋值数组为5用于debug

for(int i=0;i<s.size() ;i++)

switch(s[i]){

case '|':case'^':case'=':case'>':case'!':break;

default:ch[s[i]-'a']++;

}

//打印 运算数部分

for(int i=0;i<26;i++){

if(ch[i]!=0){

Count++;

// printf("%c\t",i+'a');

}

}

cout<<s<<endl; //输出运算式

//给运算数赋值部分

int n=1<<Count;

string stemp;//由于代替s做运算

int k=0;

while(k<n){

stemp=s;//stemp还原

int ktemp=k;

for(int i=0;i<Count;i++){

value[i]=ktemp%2;

ktemp/=2;

}

int j=0;//初始化下标

for(int i=0;i<26;i++){

if(ch[i]!=0){

if(value[j++]==0)

ch[i]='0';

else ch[i]='1'; //ch数组中储存了相应字母的真值

// printf("%c\t",ch[i]);//打印真值表初始值

}

}

for(int i=0;i<stemp.size() ;i++){

if(stemp[i]>='a'&&stemp[i]<='z')//运用桶排序桶的号即为字符，不为零既存在的思想

stemp[i]=ch[stemp[i]-'a'];//用1和0替换命题真值

}

string post;

post = MidToPost(stemp);

char ans=GetValuePost(post);

if(ans=='1')m[turnNumber(k,Count)]++;

//printf("%c\n",ans);

k++;

}

return Count;

}

int turnNumber(int k,int Count){//将数字颠倒转回

int ans=0,power=0;

while(Count--){

ans+=(k%2)\*(1<<Count);

k/=2;

}

return ans;

}

Main.cpp文件

#include"head.h"

int main() {

FILE \*fp;

fp=fopen("logic\_run.txt","r");

char ch[100];

string str;

int i = 0,count=0;

int count1=0,count2=0;//运算数的个数

int m1[100],m2[100];//用于储存极小项//

cout<<"离散课程逻辑表达式求值，可在文件中输入任何逻辑表达式均可打印出其真值表"<<endl;//print the introduce

cout<<"q|p相当于qvp"<<endl;

memset(m1,0,sizeof(m1));

memset(m2,0,sizeof(m2));

while (~fscanf(fp,"%s",ch)){//read the file put on ch

for(i=0;i<strlen(ch);i++){

str+=ch[i];

}

if((count&1)==0)

count1=Count(str,m1);

else count2=Count(str,m2);//cout<<(count1)<<endl;;cout<<(count)<<endl;}

str.clear();//将str清空预备下次储存

count++;//用于奇偶判别

}

int j;

bool flag=true;

for(j=0;j<(1<<count1);j++){

if(m1[j]!=0){

for(int i=j<<(count2-count1);i<=(j<<(count2-count1))+(count2-count1);i++){

if(m2[i]==0)flag=false;//两个不等值

}

}

else {

for(int i=j<<(count2-count1);i<=(j<<(count2-count1))+(count2-count1);i++){

if(m2[i]!=0)flag=false;//两个不等值

}

}

}

if(!flag){

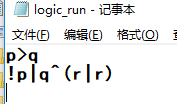
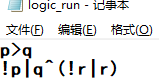
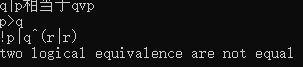
cout<<"two logical equivalence are not equal"<<endl;

}

else cout<<"two logical equivalence are equal"<<endl;

return 0;

}

 运行图： 