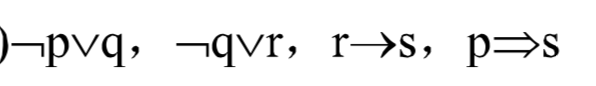
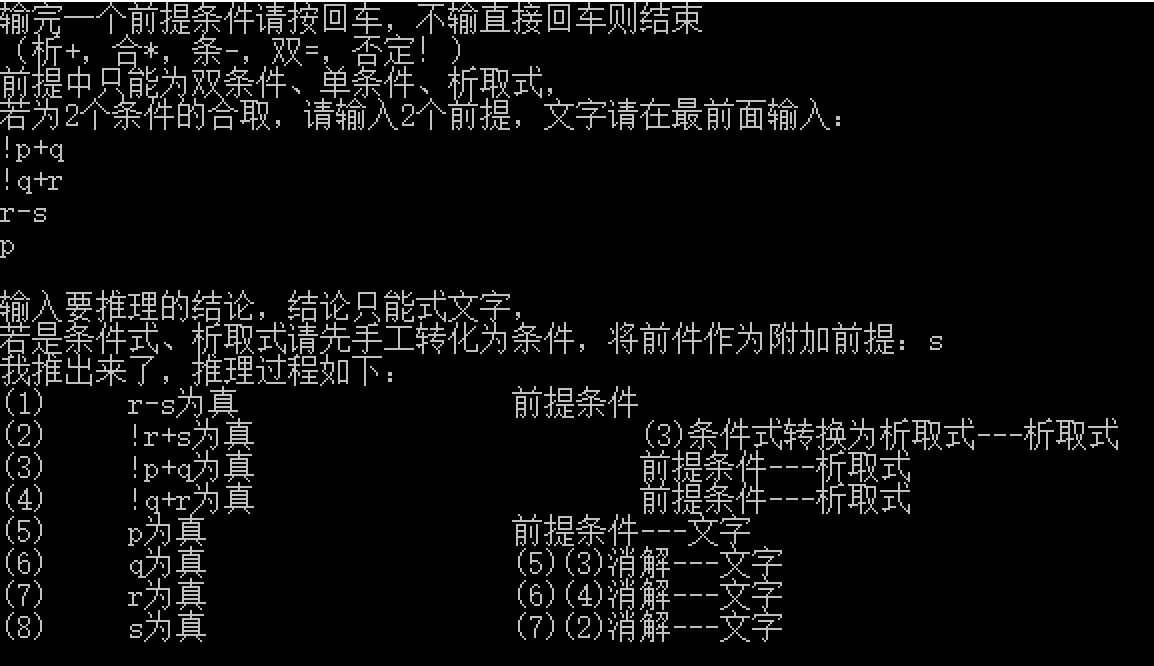
w8消解过程

1. 运行结果：





1. 代码改进与注释（红字为此段伪代码解释，红字加下划线为修改过的代码解释，//黑字为单句解释）

**前面的内容都和之前的第三次实验假言推理一样**

#include<string.h>

#include<stdio.h>

#include<stdlib.h>

struct tm

{

char gs[120],gsLast[120],niyou[120];//前件与后件及理由

int nLitter,nUsed,isLitter,isCond;//条件的长度，是否被使用，是否是字母，是否是条件式

};

void nonoop2(char aa[])

{

//!!p=p

int i=0,j=0;

while(i<strlen(aa)-2)

{

//至少还有2个字符

if(((i+1)<strlen(aa))&&(aa[i]=='!')&&(aa[i+1]=='!'))

{

j=i;

while(j<strlen(aa)-2)

{

aa[j]=aa[j+2];

j++;

}

aa[j]='\0';

break;

}

else

i++;

}

}

void printYsh(struct tm tmrec[],int np)

{

int i=0;

for(i=0;i<np;i++)

{

if(tmrec[i].isLitter==1)

printf("(%d)\t%s为真\t\t\t%s---文字\n",i+1,tmrec[i].gs,tmrec[i].niyou);

else if(tmrec[i].isCond==1)

printf("(%d)\t%s+%s为真\t\t\t%s---析取式\n",i+1,tmrec[i].gs,tmrec[i].gsLast,tmrec[i].niyou);

else

printf("(%d)\t%s为真\t\t\t%s\n",i+1,tmrec[i].gs,tmrec[i].niyou);

}

}

int setNiyou(struct tm tmrec[],int np,char ny0[],int j0,int j1,int nUsed0,int isCond0,int isLitter0)

{

//将字符串ny0与j0赋到推理意见中

char stmpj0[20],stmpj1[20];

int nLen1=0,j=0,nLenj0=0,nLenj1=0;

nLen1=strlen(ny0);

itoa(j0+1,stmpj0,10);//将j0+1转换成字符串保存在stmpj0中

nLenj0=strlen(stmpj0);//前一个依据

itoa(j1+1,stmpj1,10);

nLenj1=strlen(stmpj1);//后一个依据

if(j0==-1)

{

for(j=0;j<nLen1;j++)

tmrec[np].niyou[j]=ny0[j];

tmrec[np].niyou[j]='\0';

}

else if(j1==-1)//由前一步推理所得结论

{

tmrec[np].niyou[0]='(';

for(j=0;j<nLenj0;j++)

tmrec[np].niyou[j+1]=stmpj0[j];

tmrec[np].niyou[j+1]=')';

for(j=0;j<nLen1;j++)

tmrec[np].niyou[j+2+nLenj0]=ny0[j];

tmrec[np].niyou[j+2+nLenj0]='\0';

}

else

{

//由前两步推理所得

tmrec[np].niyou[0]='(';

for(j=0;j<nLenj0;j++)

tmrec[np].niyou[j+1]=stmpj0[j];

tmrec[np].niyou[j+1]=')';

tmrec[np].niyou[nLenj0+2]='(';

for(j=0;j<nLenj1;j++)

tmrec[np].niyou[nLenj0+j+3]=stmpj1[j];

tmrec[np].niyou[nLenj0+j+3]=')';

for(j=0;j<nLen1;j++)

tmrec[np].niyou[nLenj1+nLenj0+j+4]=ny0[j];

tmrec[np].niyou[nLenj1+nLenj0+j+4]='\0';

}

tmrec[np].nUsed=nUsed0;//附加前提从未使用过 nUsed0，int isCond0，int isLitter0）

tmrec[np].isCond=isCond0;//是条件式

tmrec[np].isLitter=isLitter0;//是文字

}

int inputPrimary(struct tm gs0[])

{

struct tm tmp;

char pstate[120];

char ny0[100]="前提条件";

char ny1[100]="条件式转换为析取式";

char ny2[100]="双条件导出的析取式";

//char ny3[100]="合取的性质";

//char ny3[100]="条件式转换为析取式";

//char ny4[100]="合取的定义";

//char ny5[100]="双否定得到";

int i=0,j=0,nLen=0,k=0;

int i0=0;//原始条件

printf("输完一个前提条件请按回车，不输直接回车则结束\n");

printf("（析+，合\*，条-，双=，否定！）\n");

printf("前提中只能为双条件、单条件、析取式，\n");

printf("若为2个条件的合取，请输入2个前提，文字请在最前面输入：\n");

while(1)

{

gets(pstate);

//fflush(stdin);

nLen=strlen(pstate);

if(nLen==0)

break;

//设置nUsed，isLitter，isCond,nLitter

//判断是否为文字

if(nLen==1)//标注单个文字

{

gs0[i].nLitter=strlen(pstate);

gs0[i].gs[0]=pstate[0];

gs0[i].gs[1]='\0';

gs0[i].gsLast[0]='\0';

setNiyou(gs0,i,ny0,-1,-1,0,0,1);//前提类型，无，无，未使用，不是条件式，是文字

}

else if((nLen==2)&&pstate[0]=='!')//标注！p

{

gs0[i].nLitter=strlen(pstate);

gs0[i].gs[0]=pstate[0];

gs0[i].gs[1]=pstate[1];

gs0[i].gs[2]='\0';

gs0[i].gsLast[0]='\0';

setNiyou(gs0,i,ny0,-1,-1,0,0,1);//前提类型，无，无，未使用，不是条件式，是文字

}

else

{

for(j=0;j<nLen;j++)

{

if(pstate[j]=='-')//标注条件式p-q

{

gs0[i].nLitter=strlen(pstate);

for(k=0;k<nLen;k++)

gs0[i].gs[k]=pstate[k];//条件前件进gs

gs0[i].gs[k]='\0';

//for(k=j+1;k<nLen;k++)

//gs0[i].gsLast[k-j-1]=pstate[k];//条件后件进gsLast

gs0[i].gsLast[0]='\0';

setNiyou(gs0,i,ny0,-1,-1,0,0,0);//前提类型，无，无，未使用，是条件式，不是文字

//再转换条件式！p-q为！!p+q

i++;

gs0[i].nLitter=gs0[i-1].nLitter;

gs0[i].gs[0]='!';

for(k=0;k<j;k++)

gs0[i].gs[k+1]=pstate[k];//条件前件进gs

gs0[i].gs[k+1]='\0';

nonoop2(gs0[i].gs);

//gs0[i].gs[0]='!';

for(k=j+1;k<nLen;k++)

gs0[i].gsLast[k-j-1]=pstate[k];//条件后件进gsLast

gs0[i].gsLast[k-j-1]='\0';

setNiyou(gs0,i,ny1,i-1,-1,0,1,0);//前提类型，无，无，未使用，是条件式，不是文字

break;

}

else if(pstate[j]=='=')//标注条件式p=q

{

//先保存双条件

gs0[i].nLitter=strlen(pstate);

for(k=0;k<strlen(pstate);k++)

gs0[i].gs[k]=pstate[k];//条件前件进gs

gs0[i].gs[k]='\0';

gs0[i].gsLast[0]='\0';//gsLast为空

setNiyou(gs0,i,ny0,-1,-1,0,0,0);//前提类型，无，无，未使用，是条件式，不是文字

//p-q即!p+q

i++;

gs0[i].nLitter=strlen(pstate);

gs0[i].gs[0]='!';

for(k=0;k<j;k++)

gs0[i].gs[k+1]=pstate[k];//p进gs

gs0[i].gs[k+1]='\0';

for(k=j+1;k<nLen;k++)

gs0[i].gsLast[k-j-1]=pstate[k];//q进gsLast

gs0[i].gsLast[k-j-1]='\0';

setNiyou(gs0,i,ny2,i-1,-1,0,1,0);//前提类型，无，无，未使用，不是条件式，不是文字

nonoop2(gs0[i].gs);//去掉可能存在的!!

//q-p=p+!q

i++;

gs0[i].nLitter=gs0[i-1].nLitter;

for(k=0;k<j;k++)

gs0[i].gs[k]=pstate[k];//条件前件进gs

gs0[i].gs[k]='\0';

gs0[i].gsLast[0]='!';

for(k=j+1;k<nLen;k++)

gs0[i].gsLast[k-j-1+1]=pstate[k];//条件后件进gsLast

gs0[i].gsLast[k-j-1+1]='\0';

setNiyou(gs0,i,ny2,i-2,-1,0,1,0);//前提类型，无，无，未使用，是条件式，不是文字

nonoop2(gs0[i].gsLast);//去掉可能存在的!!

break;

}

else if(pstate[j]=='+')//标注条件式p+q

{

//先保存双条件

gs0[i].nLitter=strlen(pstate);

for(k=0;k<j;k++)

gs0[i].gs[k]=pstate[k];//双条件全部进gs

gs0[i].gs[k]='\0';

for(k=0;k<nLen;k++)

gs0[i].gsLast[k-j-1]=pstate[k];//双条件全部进gs

gs0[i].gsLast[k-j-1]='\0';

setNiyou(gs0,i,ny0,-1,-1,0,1,0);//前提类型，无，无，未使用，是条件式，不是文字

break;

}

}

if(j>=nLen)//不是条件式，也不是文字，则是普通的条件

{

gs0[i].nLitter=strlen(pstate);

for(k=0;k<nLen;k++)

gs0[i].gs[k]=pstate[k];//公式全进gs

gs0[i].gs[k]='\0';

gs0[i].gsLast[0]='\0';//gsLast为空串

setNiyou(gs0,i,ny0,-1,-1,0,0,0);//前提类型，无，无，未使用，不是条件式，不是文字

}

}

i++;//当前公式处理完以后，指针i的值增1

}

nLen=i;//按字符串的长度排序

for(i=0;i<nLen-1;i++)

{

k=i;

for(j=i+1;j<nLen-1;j++)

if(gs0[k].nLitter>gs0[j].nLitter)

k=j;

if(k>i)

{

tmp=gs0[i];

gs0[i]=gs0[k];

gs0[k]=tmp;

}

}

return nLen;

}

int main()

{

struct tm gs0[100];//推理前提条件

char result0[128];//结论

struct tm tmrec[1024];//最多1000步

char stmp[128][128];

char lastNiYou[128]=" ";//上个推理式的理由

char ny01[100]="消解";

int i=0,k=0,j=0,j1=0,np=1,np0=0,isOk=0;

int i0=0,nPosLitter[128]={0},nPosCond=0;//文字起始位置，首个文字的位置，消解式的位置

//输入前提条件

np0=inputPrimary(gs0);

//输入结论

printf("输入要推理的结论，结论只能式文字，\n");

printf("若是条件式、析取式请先手工转化为条件，将前件作为附加前提：");

gets(result0);

fflush(stdin);

for(i=0;i<np0;i++)//所有原始公式转抄到了tmrec中

tmrec[i]=gs0[i];

np=i;//推理队列的尾部指针

//nPosLitter[j]=0;//文字的位置号

nPosCond=0;//条件的位置号

isOk=0;

i0=-1;

while(1)

{

i=i0+1;//寻找下一个文字，i是起始位置，np是命令串的长度

while((i<np)&&(tmrec[i].isLitter!=1))

i++;

if(i>=np)//找不到文字，就无法推理

break;

i0=i;//记录从源头查询的首个文字的位置号，下次从此号往后寻找

nPosLitter[j1]=i;//记录文字的位置

strcpy(stmp[j1],tmrec[i].gs);//保存当前文字的内容

if(strcmp(result0,stmp[j1])==0)//推出结论同条原始条件的下一轮

{

isOk=1;

break;

}

np0=np-1;

while(np>np0)//从当前文字的下一个位置起寻找条件式，则一路往下走

{

np0=np;

for(i=0;i<np;i++)//找到一个没有用过的条件式

if((tmrec[i].isCond==1)&&(tmrec[i].nUsed==0))

break;

if(i==np)//没有找到则结束推理，所有的条件式都用到了

break;

while(i<np) //若找到了这样的条件

{

if(tmrec[i].isCond==1)//若是条件式

{

for(k=0;k<=j1;k++)

if((strcmp(lastNiYou,tmrec[i].niyou)!=0)||((strcmp(lastNiYou,tmrec[i].niyou)==0)&&(tmrec[i].niyou[0]!='(')))

{

//与上条命令的来源不相同，或者但是同为前提条件式可以的，即首个字符不是（

if((tmrec[i].gs[0]=='!')&&(stmp[k][0]!='!')&&(strlen(tmrec[i].gs)-strlen(stmp[k])==1))

{

//如果析取式的前件与stem即可消解，则将后件保存到stme中

j=0;

while(j<strlen(stmp[k]))//依次比较每一个字符

{

if(tmrec[i].gs[j+1]!=stmp[k][j])//有一个不相等则结束比较

break;

j++;

}

if(j>=strlen(stmp[k]))//如果比到最后仍相等，则这二个可消解

{

strcpy(lastNiYou,tmrec[i].niyou);

//tmrec[nPosLitter[k]].nUsed++;//这个文字用过一次了

tmrec[i].nUsed++;//这个析取式用过一次了

j1++;

strcpy(stmp[j1],tmrec[i].gsLast);//将此消解结果保存到推导序列中

strcpy(tmrec[np].gs,stmp[j1]);//将当前推出来的结果保存起来

tmrec[np].gsLast[0]='\0';//后件清空，保存当前条件

setNiyou(tmrec,np,ny01,nPosLitter[k],i,0,0,1);//前提类型，有，无，未使用，不是条件式

nPosLitter[j1]=np;

np++;

if(strcmp(result0,stmp[j1])==0)//推出结论同条原始条件的下一轮

{

isOk=1;

break;

}

}

}

if((tmrec[i].gsLast[0]=='!')&&(stmp[k][0]!='!')&&(strlen(tmrec[i].gsLast)-strlen(stmp[k])==1))

{

//如果析取式的后件与stem即可消解，则将后件保存到stme中

j=0;

while(j<strlen(stmp[k]))//依次比较每一个字符

{

if(tmrec[i].gsLast[j+1]!=stmp[k][j])//有一个不相等则结束比较

break;

j++;

}

if(j>=strlen(stmp[k]))//如果比到最后仍相等，则这二个可消解

{

strcpy(lastNiYou,tmrec[i].niyou);

//tmrec[nPosLitter[j1]].nUsed++;//这个文字用过一次了

tmrec[i].nUsed++;//这个析取式用过一次了

j1++;

strcpy(stmp[j1],tmrec[i].gs);//将此消解结果保存到推导序列中

strcpy(tmrec[np].gs,stmp[j1]);//将当前推出来的结果保存起来

tmrec[np].gsLast[0]='\0';//后件清空，保存当前条件

setNiyou(tmrec,np,ny01,nPosLitter[k],i,0,0,1);//前提类型，有，无，未使用，不是条件式

nPosLitter[j1]=np;

np++;

if(strcmp(result0,stmp[j1])==0)//推出结论同条原始条件的下一轮

{

isOk=1;

break;

}

}

}

if((tmrec[i].gs[0]!='!')&&(stmp[k][0]=='!')&&(strlen(tmrec[i].gs)-strlen(stmp[k])==-1))

{

//如果析取式的前件与stem即可消解，则将后件保存到stme中

j=0;

while(j<strlen(stmp[k]))//依次比较每一个字符

{

if(tmrec[i].gs[j]!=stmp[k][j+1])//有一个不相等则结束比较

break;

j++;

}

if(j>=strlen(stmp[k]))//如果比到最后仍相等，则这二个可消解

{

strcpy(lastNiYou,tmrec[i].niyou);

//tmrec[nPosLitter[j1]].nUsed++;//这个文字用过一次了

tmrec[i].nUsed++;//这个析取式用过一次了

j1++;

strcpy(stmp[j1],tmrec[i].gsLast);//将此消解结果保存到推导序列中

strcpy(tmrec[np].gs,stmp[j1]);//将当前推出来的结果保存起来

tmrec[np].gsLast[0]='\0';//后件清空，保存当前条件

setNiyou(tmrec,np,ny01,nPosLitter[k],i,0,0,1);//前提类型，有，无，未使用，不是条件式

nPosLitter[j1]=np;

np++;

if(strcmp(result0,stmp[j1])==0)//推出结论同条原始条件的下一轮

{

isOk=1;

break;

}

}

}

if((tmrec[i].gsLast[0]!='!')&&(stmp[k][0]=='!')&&(strlen(tmrec[i].gsLast)-strlen(stmp[k])==-1))

{

//如果析取式的后件与stem即可消解，则将后件保存到stme中

j=0;

while(j<strlen(stmp[k]))//依次比较每一个字符

{

if(tmrec[i].gsLast[j]!=stmp[k][j+1])//有一个不相等则结束比较

break;

j++;

}

if(j>=strlen(stmp[k]))//如果比到最后仍相等，则这二个可消解

{

strcpy(lastNiYou,tmrec[i].niyou);

//tmrec[nPosLitter[j1]].nUsed++;//这个文字用过一次了

tmrec[i].nUsed++;//这个析取式用过一次了

j1++;

strcpy(stmp[j1],tmrec[i].gs);//将此消解结果保存到推导序列中

strcpy(tmrec[np].gs,stmp[j1]);//将当前推出来的结果保存起来

tmrec[np].gsLast[0]='\0';//后件清空，保存当前条件

setNiyou(tmrec,np,ny01,nPosLitter[k],i,0,0,1);//前提类型，有，无，未使用，不是条件式

nPosLitter[j1]=np;

np++;

if(strcmp(result0,stmp[j1])==0)//推出结论同条原始条件的下一轮

{

isOk=1;

break;

}

}

}

}

if(isOk==1)

break;

}

i++;//判断下一个表达式是否为条件，是否为可推理的条件式

}

}

if(isOk==1)//以成功推导，不用再找下一个文字

break;

}

if(isOk==1)

printf("我推出来了，推理过程如下：\n");

else

printf("我推不出来，推理过程如下：\n");

printYsh(tmrec,np);

}