HUNAN UNIVERSITY

离 散 数 学

实验报告

学生姓名/学号 甘晴void 20210801020X

专业班级 计科 210X

指导老师 杨圣洪

2022 年 6 月 9 日

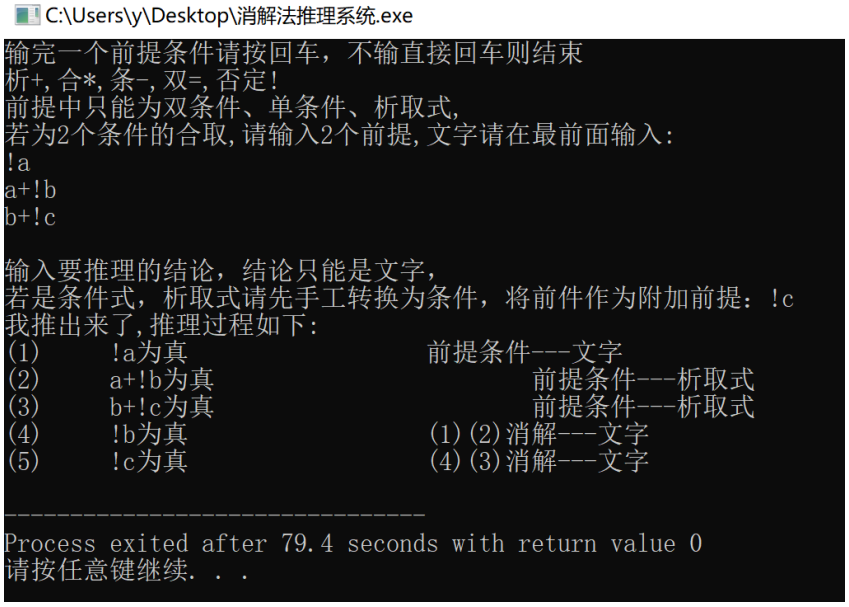
目录

1 题目要求

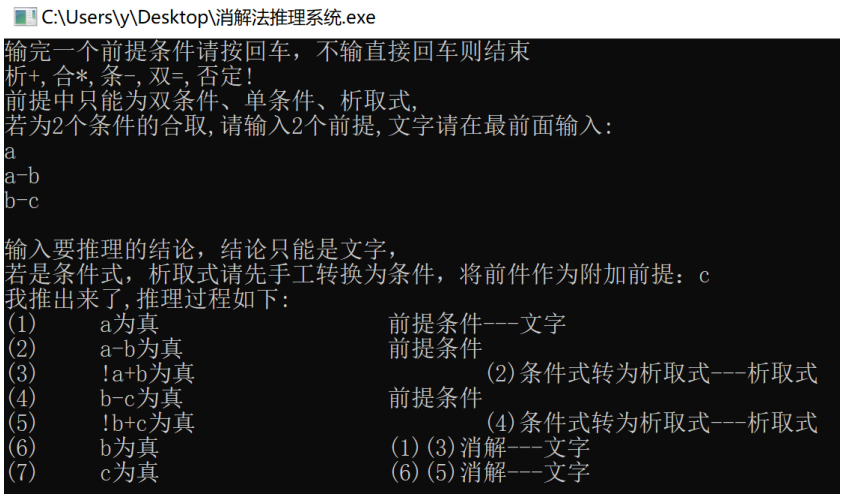
请设计消解推理系统（基于消解法推理，不能使用假言推理的自然推理系统）

2 测试数据

2.1 只包含析取式的简单测试



2.2 只包含单条件的简单测试



2.3 只包含双条件的简单测试

C:\Users\y\Desktop\消解法推理系统.exe

```
输完一个前提条件请按回车，不输直接回车则结束
析+,合*,条-,双=,否定!
前提中只能为双条件、单条件、析取式,
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:
!a
a=b
b=c

输入要推理的结论，结论只能是文字,
若是条件式，析取式请先手工转换为条件，将前件作为附加前提：!c
我推出来了,推理过程如下:
(1)     !a为真                  前提条件---文字
(2)     a=b为真                 前提条件
(3)     !a+b为真                        (2)双条件导出的析取式---析取式
(4)     a+!b为真                        (2)双条件导出的析取式---析取式
(5)     b=c为真                 前提条件
(6)     !b+c为真                        (5)双条件导出的析取式---析取式
(7)     b+!c为真                        (5)双条件导出的析取式---析取式
(8)     !b为真                  (1)(4)消解---文字
(9)     !c为真                  (8)(7)消解---文字
```

2.4 包含单条件、双条件和析取式的综合测试

C:\Users\y\Desktop\消解法推理系统.exe

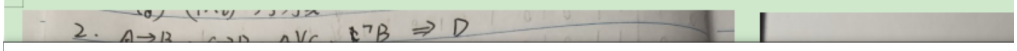
```
输完一个前提条件请按回车，不输直接回车则结束
析+,合*,条-,双=,否定!
前提中只能为双条件、单条件、析取式,
若为2个条件的合取,请输入2个前提,文字请在最前面输入:
a
!a+b
b-c
c=d

输入要推理的结论，结论只能是文字,
若是条件式，析取式请先手工转换为条件，将前件作为附加前提：d
我推出来了,推理过程如下:
(1)     a为真                   前提条件---文字
(2)     b-c为真                 前提条件
(3)     !b+c为真                        (3)条件式转为析取式---析取式
(4)     c=d为真                 前提条件
(5)     !c+d为真                        (5)双条件导出的析取式---析取式
(6)     c+!d为真                        (5)双条件导出的析取式---析取式
(7)     !a+b为真                        前提条件---析取式
(8)     b为真                   (1)(7)消解---文字
(9)     c为真                   (8)(3)消解---文字
(10)    d为真                   (9)(5)消解---文字
```

2.5 消解法作业真题测试

第2题

(8)请分别用真值表、假言推理、纯消解法证明 （说明：=>是推理符号，+是析取，!是否定，=是双条件） (A-B)，(C-D)，(A+C) => D

可直接写下方，或请将作业拍照或截屏并转到本机，格式为jpg/jpgeg/gif/png，因空间有限图像文件长度请小于3MB，可采用WPS或word



C:\Users\y\Desktop\消解法推理系统.exe

```
输完一个前提条件请按回车，不输直接回车则结束
析+,合*,条-,双=,否定!
前提中只能为双条件、单条件、析取式,
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:
a-b
c-d
a+c
!b

输入要推理的结论，结论只能是文字,
若是条件式，析取式请先手工转换为条件，将前件作为附加前提：d
我推出来了,推理过程如下:
(1)     !b为真                  前提条件---文字
(2)     !a+b为真                        (1)条件式转为析取式---析取式
(3)     c-d为真                 前提条件
(4)     !c+d为真                        (3)条件式转为析取式---析取式
(5)     a+c为真                 前提条件---析取式
(6)     a-b为真                 前提条件
(7)     !a为真                  (1)(2)消解---文字
(8)     c为真                   (7)(5)消解---文字
(9)     d为真                   (8)(4)消解---文字
```

3 改进要点

3.1 关于头文件的简化

此处修改点位于第 10 行。

可用万能头文件#include<bits/stdc++.h>减少头文件数量

3.2 关于调换 int setNiyou(……)与 inputPrimary(……)函数顺序的修改

此处修改点位于第 61 行。

此处需要调换这两个函数位置，否则会出现 inputPrimary(……)调用未定义函数 int setNiyou(……)的情况。

修改方法就是将 int setNiyou(struct tmd tmdrec[], int np, char ny0[], int j0, int j1, int nUsed0, int isCond0, int isLitter0)这个函数放置在 inputPrimary(……)前面，或者在程序开头事先声明。

3.3 关于将末位元素纳入排序的修改

此处修改点位于第 248 行。

原来的代码为 for (j = i + 1; j < nLen - 1; j++)

存在的问题是最后的元素未被纳入排序内，这将对结果造成影响，故修改。

修改后的代码为 for (j = i + 1; j < nLen; j++)

修改的意义：修改循环条件使得最后的元素可排序。

3.4 关于头文件的简化

此处修改点位于第 254 行。

原来的代码为：

```
tmdp = gs0[i];
gs0[i] = gs0[k];
gs0[k] = tmdp;
```

存在的问题是影响代码的美观性，可以考虑使用重写的 swap 函数封装，实现效果。修改后的代码如下（需要重写一遍该 swaptmd 函数并防于本函数体前面）

修改：使用自定义的 swaptmd 排序。

修改后的代码为：swaptmd(gs0[i],gs0[k]);

4 代码实现

```
/*
 * 实验四
 * 利用消解法进行推理
 * by Arcticwolf
```

```
 * CSEE----Computer Science and Technology 2102
 *  信息科学与工程学院----计算机科学与技术  2102 班----梅炳寅
 *  学号 202108010206
 */

#include <string.h>
#include <iostream>
#include <cstdio>
#include <cmath>
#include <cstdlib>
#include <algorithm>
//修改点：可用#include<bits/stdc++.h>减少头文件数量

struct tmd
{
    char gs[120], gsLast[120], niyou[120];//前件与后件及理由
    int nLitter, nUsed, isLitter, isCond;
};

void nonoop2( char aa[] )
{
    //!!p=p
    int i = 0, j = 0;
    while (i < strlen( aa ) - 2)
    {
        //至少还有两个字符
        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 tmd tmdrec[], int np )
{
```

```
    int i = 0;
    for (i = 0; i < np; i++)
    {
        if (tmdrec[i].isLitter == 1)
            printf(  "(%d)\t%s  为 真 \t\t\t%s--- 文 字 \n",  i  +  1,  tmdrec[i].gs,
tmdrec[i].niyou );
        else if (tmdrec[i].isCond == 1)
            printf(  "(%d)\t%s+%s  为真\t\t\t%s---析取式\n",  i  +  1,  tmdrec[i].gs,
tmdrec[i].gsLast, tmdrec[i].niyou );
        else
            printf( "(%d)\t%s 为真\t\t\t%s\n", i + 1, tmdrec[i].gs, tmdrec[i].niyou );
    }
}

/* 修改点 1：调换函数位置，否则会出现调用未定义函数的情况 */
int setNiyou( struct tmd tmdrec[], 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 stmdpj0[20], stmdpj1[20];
    int nLen1 = 0, j = 0, nLenj0 = 0, nLenj1 = 0;
    nLen1 = strlen( ny0 );
    itoa( j0 + 1, stmdpj0, 10 );
    nLenj0 = strlen( stmdpj0 );//前一个依据
    itoa( j1 + 1, stmdpj1, 10 );
    nLenj1 = strlen( stmdpj1 );//后一个依据
    if (j0 == -1)
    {
        //原始前提
        for (j = 0; j < nLen1; j++)
            tmdrec[np].niyou[j] = ny0[j];
        tmdrec[np].niyou[j] = '\0';
    }
    else if (j1 == -1)//由前一步推理所得结论
    {
        tmdrec[np].niyou[0] = '(';
        for (j = 0; j < nLenj0; j++)
            tmdrec[np].niyou[j + 1] = stmdpj0[j];
        tmdrec[np].niyou[j + 1] = ')';

        for (j = 0; j < nLen1; j++)
            tmdrec[np].niyou[j + 2 + nLenj0] = ny0[j];
        tmdrec[np].niyou[j + 2 + nLenj0] = '\0';
    }
```

```
    else
    {
        //由前二步推理所得
        tmdrec[np].niyou[0] = '(';
        for (j = 0; j < nLenj0; j++)
            tmdrec[np].niyou[j + 1] = stmdpj0[j];
        tmdrec[np].niyou[j + 1] = ')';

        tmdrec[np].niyou[nLenj0 + 2] = '(';
        for (j = 0; j < nLenj1; j++)
            tmdrec[np].niyou[nLenj0 + 3 + j] = stmdpj1[j];
        tmdrec[np].niyou[nLenj0 + 3 + j] = ')';

        for (j = 0; j < nLen1; j++)
            tmdrec[np].niyou[nLenj0 + nLenj1 + 4 + j] = ny0[j];
        tmdrec[np].niyou[nLenj0 + nLenj1 + 4 + j] = '\0';
    }

    tmdrec[np].nUsed = nUsed0;//附加前提从未使用过 nUsed0,int isCond0,int
isLitter0
    tmdrec[np].isCond = isCond0;//是条件式
    tmdrec[np].isLitter = isLitter0;//是文字
}

void swaptmd(tmd &a,tmd &b)
{
     tmd temp;
     temp=a;
     a=b;
     b=temp;
}


int inputPrimary( struct tmd gs0[] )
{
    struct tmd tmdp;
    char pstate[120];
    char *ny0 = "前提条件";
    char *ny1 = "条件式转为析取式";
    char *ny2 = "双条件导出的析取式";
    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 析+,合*,条-,双
=,否定!\n 前提中只能为双条件、单条件、析取式,\n 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 );
        nLen = strlen( pstate );
        if (nLen == 0)
        {
            break;
        }//设置 nUsed,isLitter,isCond,nLittle 的值
        //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];//整个表达式进入 gs
                    gs0[i].gs[k] = '\0';
                    gs0[i].gsLast[0] = '\0';
                    setNiyou( gs0, i, ny0, -1, -1, 0, 0, 0 );//前提类型，无，无，未使
用，不是析取式，不是文字

                    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];
                    gs0[i].gs[k + 1] = '\0';
                    nonoop2( gs0[i].gs );

                    for (k = j + 1; k < nLen; k++)
                        gs0[i].gsLast[k - j - 1] = pstate[k];
                    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] == '=')//标注双条件 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]; }//双条件全部进 gs
                    gs0[i].gs[k] = '\0';
                    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
                                                                    //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，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 p 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!q 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] == '+')//标注析取式 p+q，也要分解到 gs 与 gsLast
中
                {
                    gs0[i].nLitter = strlen( pstate );
                    for (k = 0; k < j; k++)
                        gs0[i].gs[k] = pstate[k];       //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++)
        /*  修改点 2：修改循环条件使得最后的元素可排序  */
```

```
            for (j = i + 1; j < nLen; j++)
                if (gs0[k].nLitter > gs0[j].nLitter)
                    k = j;
            if (k > i)
            {
                /* 修改点 3：使用自定义的 swaptmd 排序 */
                swaptmd(gs0[i],gs0[k]);
//              tmdp = gs0[i];
//              gs0[i] = gs0[k];
//              gs0[k] = tmdp;
            }
        }
        return nLen;
}

int main()
{
        struct tmd gs0[100];//推理前提条件
        char result0[128]; //结论
        struct tmd tmdrec[1024];//最多 1000 步

        char stmdp[128];
        char lastNiYou[128] = " ";//上一个推理式的理由
        char *ny01 = "消解";
        int i = 0, j = 0, k = 0;
        int np = 1, np0 = 0, isOk = 0;
        int i0 = 0, nPosLitter = 0, nPosCond = 0;//文字起始的位置，首个文字的位置，消解式的位置
        np0 = inputPrimary( gs0 );
        //输入结论
        printf( "输入要推理的结论，结论只能是文字，\n 若是条件式，析取式请先手工转换为条件，将前件作为附加前提：" );
        gets( result0 );
        fflush( stdin );
        for (i = 0; i < np0; i++)
        {
            tmdrec[i] = gs0[i];//所有原始公式转抄到 tmdrec 中
        }
        np = i;//推理队列的尾部指针
        nPosLitter = 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) && (tmdrec[i].isLitter != 1))
        i++;
    if (i >= np)
        break;//找不到文字我就没法推理了
    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 = i;//记录文字的位置
    strcpy( stmdp, tmdrec[i].gs );//保存当前文字的内容
    np0 = np - 1;
    while (np > np0) //从当前文字的下一个位置起寻找析取式，则一路往下走
    {
        np0 = np;
        for (i = 0; i < np; i++)//找到一个没有用过的戏曲式
            if ((tmdrec[i].isCond == 1) && (tmdrec[i].nUsed == 0))
                break;
        if (i == np)
            break;//没有找到则结束推理，所有条件式都用到了
        while (i < np)//若找到了这样的条件式
        {
            if ((tmdrec[i].isCond == 1))//若是条件式
            {
                //与上条命令的来源不同，或者但是同为前提条件也是可以的，即首个字符不是（
                if (((strcmp( lastNiYou, tmdrec[i].niyou ) != 0) || ((strcmp( lastNiYou, tmdrec[i].niyou ) == 0) && tmdrec[i].niyou[0] != '(')))
                {

                    if ((tmdrec[i].gs[0] == '!') && (stmdp[0] != '!') && (strlen( tmdrec[i].gs ) - strlen( stmdp ) == 1))       // !p+q p cuo
                    {
                        //如果析取式的前件与 stmdp 即可消解，则将后件保存的 stmdp 中
                        j = 0;
                        while (j < strlen( stmdp ))//依次比较每个字符
                        {
                            if (tmdrec[i].gs[j + 1] != stmdp[j])
                                break;//有一个不相等则结束比较
                            j++;
                        }

                        if (j >= strlen( stmdp ))   //如果比到最后仍然相等，则这二个可消解
                        {
```

```
                                strcpy( lastNiYou, tmdrec[i].niyou );
                                tmdrec[nPosLitter].nUsed++;   //这个文字用过一
次了
                                tmdrec[i].nUsed++;   //这个析取式用过一次了

                                strcpy( stmdp, tmdrec[i].gsLast );   //将次消解结
果保存到推导序列中
                                strcpy( tmdrec[np].gs, stmdp );   //将当前推出来
的结果保存起来
                                tmdrec[np].gsLast[0] = '\0';   //后件清空，保存当
前条件
                                setNiyou( tmdrec, np, ny01, nPosLitter, i, 0, 0, 1 );
//前提类型，有，无，未使用，不是条件
                                nPosLitter = np;   //记录当前文字的序号
                                np++;
                                if (strcmp( result0, stmdp ) == 0)
                                {
                                    isOk  =  1;      //推出结论同条原是调节的下
一轮
                                    break;
                                }
                            }
                        }
                        else if ((tmdrec[i].gsLast[0] == '!') && (stmdp[0] != '!') &&
(strlen( tmdrec[i].gsLast ) - strlen( stmdp ) == 1))          //a+!b    b dui
                        {
                            //如果析取式的后件与 stmdp 即可消解，则将前件保
存到 stmdp 中
                            j = 0;
                            while (j < strlen( stmdp ))   //依次比较每一个字符
                            {
                                if (tmdrec[i].gsLast[j + 1] != stmdp[j])
                                    break;      //有一个不相等则结束比较
                                j++;
                            }
                            if (j >= strlen( stmdp ))   //如果比到最后仍然相等，则
这两个可消解
                            {
                                strcpy( lastNiYou, tmdrec[i].niyou );
                                tmdrec[nPosLitter].nUsed++;   //这个文字用过一
次了
                                tmdrec[i].nUsed++;   //这个析取式用过一次了

                                strcpy( stmdp, tmdrec[i].gs );   //将次消解结果保
```

```
存到推导序列中
                    strcpy( tmdrec[np].gs, stmdp );  //将当前推出来
的结果保存起来
                    tmdrec[np].gsLast[0] = '\0';  //后件清空，保存当
前条件
                    setNiyou( tmdrec, np, ny01, nPosLitter, i, 0, 0, 1 );
//前提类型，有，无，未使用，不是条件
                    nPosLitter = np;  //记录当前文字的序号
                    np++;
                    if (strcmp( result0, stmdp ) == 0)
                    {
                        isOk = 1;      //推出结论同条原是调节的下
一轮
                        break;
                    }
                }
            }
            else if ((tmdrec[i].gs[0] != '!') && (stmdp[0] == '!') &&
(strlen( tmdrec[i].gs ) - strlen( stmdp ) == -1))        //  p+q !p
            {
                //如果析取式的后件与 stmdp 即可消解，则将前件保
存到 stmdp 中
                j = 0;
                while (j < strlen( tmdrec[i].gs ))  //依次比较每一个字
符
                {
                    if (stmdp[j + 1] != tmdrec[i].gs[j])
                        break;      //有一个不相等则结束比较
                    j++;
                }
                if (j >= strlen( tmdrec[i].gs ))
                {
                    strcpy( lastNiYou, tmdrec[i].niyou );
                    tmdrec[nPosLitter].nUsed++;  //这个文字用过一
次了
                    tmdrec[i].nUsed++;  //这个析取式用过一次了

                    strcpy( stmdp, tmdrec[i].gsLast );  //将次消解结
果保存到推导序列中
                    strcpy( tmdrec[np].gs, stmdp );  //将当前推出来
的结果保存起来
                    tmdrec[np].gsLast[0] = '\0';  //后件清空，保存当
前条件
                    setNiyou( tmdrec, np, ny01, nPosLitter, i, 0, 0, 1 );
```

```
//前提类型，有，无，未使用，不是条件
                                nPosLitter = np;   //记录当前文字的序号
                                np++;
                                if (strcmp( result0, stmdp ) == 0)
                                {
                                    isOk  =  1;      //推出结论同条原是调节的下
一轮
                                    break;
                                }
                            }

                        }
                        else if ((tmdrec[i].gsLast[0] != '!') && (stmdp[0] == '!') &&
(strlen( tmdrec[i].gsLast ) - strlen( stmdp ) == -1))          //p+q !q
                        {
                            //如果析取式的后件与 stmdp 即可消解，则将前件保
存到 stmdp 中
                            j = 0;
                            while (j < strlen( tmdrec[i].gsLast ))//依次比较每一个
字符
                            {
                                if (stmdp[j + 1] != tmdrec[i].gsLast[j])
                                    break;      //有一个不相等则结束比较
                                j++;
                            }
                            if (j >= strlen( tmdrec[i].gsLast ))//如果比到最后仍然
相等，则这两个可消解
                            {
                                strcpy( lastNiYou, tmdrec[i].niyou );
                                tmdrec[nPosLitter].nUsed++; //这个文字用过一
次了
                                tmdrec[i].nUsed++;              //这个条件用过一
次了

                                strcpy( stmdp, tmdrec[i].gs );       //将此中间结
果保存到推导序列中
                                strcpy( tmdrec[np].gs, stmdp );         //将当前推
出来的结果保存起来
                                tmdrec[np].gsLast[0] = '\0';         //后件清空，保
存当前条件
                                setNiyou( tmdrec, np, ny01, nPosLitter, i, 0, 0,
1 );//前提类型，有，无，未使用，不是条件式
                                nPosLitter = np;                    //记录当前文
字的序号
```

```
                                np++;
                                if (strcmp( result0, stmdp ) == 0)
                                {
                                    isOk  =  1;      //推出结论同原始条件的下一
轮
                                    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)
                break;      //我推出来了，不要再找下一个文字了
        }
        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( tmdrec, np );
}
```