同济大学
TONGJI UNIVERSITY

人工智能课程报告

# 基于广度优先搜索算法的 Horn 子句归结实验

学　　院：电子信息工程学院
学　　号：
学生姓名：
指导教师：武妍

2021 年 06 月 15 日

## 摘要

归结反演推理方法是一种系统性推理算法，可以使智能体通过一系列已给定的子句集，通过将两个子句归结合并的方式，判断某个结论是否成立。该方法可以被高效地应用于计算机中，是能在计算机中较好实现的一种推理技术，使得定理自动证明获得了长足进步。本实验采用将目标取反并入子句集，然后查看子句集能否推出空子句的证明方式，实现了一个子句集自动推出内核。该内核遵循上述二元归结规则，整体采用广度优先遍历搜索的方式推出子树，同时在实验报告最后，提出了包容归结策略的优化策略。本次实验的交互界面部分使用了 MFC 程序框架来实现与用户的交互，用户可以通过界面指示较为方便地进行子句集、目标语句的输入与归结过程的查看。



**关键词：Horn 子句，归结原理，逻辑推理，广度优先搜索，MFC 框架，人工智能**

# 目录

# 1 实验目的

熟悉和掌握归结原理的基本思想和基本方法，通过培养学生利用逻辑方法表示知识，并掌握采用机器推理来进行问题求解的基本方法。详细可以划分为：

（1）熟悉与掌握归结原理，能够使用归结原理完成一些逻辑问题的推理；

（2）根据归结原理的思考方式，写出程序能够让计算机自动归结；

（3）在过程中掌握利用机器推理来进行问题求解的基本方法。

# 2 实验内容

根据《人工智能课程设计》实验三指导书，结合本小组自身能力与完成情况，可以对本次实验的实验内容作出如下划分：

（1）根据一阶逻辑命题的相关知识，利用消除蕴涵词、Skolem 化等操作将指导书所给出的破案问题转化为条件子句集，并将谋杀者信息转化为目标语句。（由于这一部分的内容仅针对破案问题，且是由人工转化得来的，与程序设计无关，故将不在本次实验报告中详细叙述这一部分的内容，仅在成果展示部分进行一定程度的说明。）

（2）编写内核程序，能够将目标语句取反并入原始子句集，之后可以通过遍历搜索的方式判定该子句集能否归结出空子句，从而判断目标语句是否成立。

（3）为内核程序编写 UI 界面，使用 MFC 框架实现框体输入输出的交互，使得用户友好性得以一定程度的提升。

# 3 归结演绎推理

**归结演绎推理**是一种基于逻辑“反证法”的机械化定理证明方法。其基本思想是把永真性的证明转化为不可满足性的证明。即要证明 $P \to Q$ 永真，只要能够证明 $P \wedge \sim Q$ 为不可满足即可。谓词公式不可满足的充要条件是其子句集不可满足。因此，要把谓词公式转换为子句集，再用鲁滨逊归结原理求解子句集是否不可满足。如果子句集不可满足，则 $P \to Q$ 永真。

## 3.1 谓词公式与子句集

### 3.1.1 谓词公式性质

(1) 永真性: 如果谓词公式 P 对非空个体域 D 上的任一解释都取得真值 T，则称 P 在 D 上是永真的；如果 P 在任何非空个体域上均是永真的，则称 P 永真。

(2) 可满足性: 对于谓词公式 P，如果至少存在 D 上的一个解释，使公式 P 在此解释下的真值为 T，则称公式 P 在 D 上是可满足的。

### 3.1.2 子句集定义

(1) 原子谓词公式及其否定统称为文字。例如，P（x），Q（x），∼P（z），∼Q（x）等都是文字。

(2) 任何文字的析取式称为子句。例如，P（z）V Q（x），P（x，f（x））V Q（x，g（x））都是子句。

(3) 不包含任何文字的子句称为空子句。由于空子句不含有任何文字，也就不能被任何解释所满足，因此空子句是永假的，不可满足的。空子句一般被记为 NIL。

(4) 由子句或空子句所构成的集合称为子句集。

### 3.1.3 谓词公式化为子句集

**假设谓词公式为**

$(\forall x)[\sim P(x) \vee \sim Q(x)] \rightarrow (\exists y)[S(x, y) \wedge Q(x)] \wedge (\forall x)[P(x) \vee B(x)]$

**(1) 消去蕴含符号**

$(\forall x)\sim[\sim P(x) \vee \sim Q(x)] \vee (\exists y)[S(x, y) \wedge Q(x)] \wedge (\exists x)[P(x) \vee B(x)]$

**(2) 把否定符号移到每个谓词前面**

$(\forall x)[P(x) \wedge Q(x)] \vee (\exists y)[S(x, y) \wedge Q(x)] \wedge (\forall x)[P(x) \vee B(x)]$

**(3) 变量标准化**

$(\forall x)\ [P(x) \wedge Q(x)] \vee (\exists y)[S(x,y) \wedge Q(x)] \wedge (\forall w)[P(w) \vee B(w)]$

**(4) 消去存在量词，设 y 的函数是 f（x），则**

$(\forall x)[P(x) \wedge Q(x)] \vee [S(x,f(x)) \wedge Q(x)] \wedge (\forall w)[P(w) \vee B(w)]$

**(5) 化为前束形**

$(\forall x)(\forall w)[P(x) \wedge Q(x)] \vee [S(x,f(x)) \wedge Q(x)] \wedge [P(w) \vee B(w)]$

**(6) 化为标准形**

$(\forall x)(\forall w)[Q(x) \wedge P(x) \vee [Q(x) \wedge S(x,f(x))]] \wedge [P(w) \vee B(w)]$ $(\forall x)(\forall w)$
$Q(x) \wedge [P(x) \vee S(x,f(x))] \wedge [P(w) \vee B(w)]$

**(7) 略去全称量词**

Q(x) Λ [P(x) V S(x,f(x)) Λ [P(w) V B(w)]

**(8) 消去合取词，把母式用子句集表示**

{Q(x), P(x) V S(x,f(x)), P(w) V B(w)}

**(9) 子句变量标准化**

{Q(x), P(y) V S(y,f(y)), P(w) v B(w)}

## 3.2 归结 (消解) 原理

### 3.2.1 归结反演证明定理

(1) 将已知前提表示为谓词公式 F。

(2) 将待证明的结论表示为谓词公式 Q, 并否定得到 ~Q。

(3) 把谓词公式集 {F,~Q} 化为子句集 S。

(4) 应用归结原理对子句集 S 中的子句进行归结，并把每次归结得到的归结式都并入到 S 中。如此反复进行，若出现了空子句，则停止归结，此时就证明了 Q 为真。

### 3.2.2 归结反演求解问题

(1) 已知前提 F 用谓词公式表示;

(2) 把待求解的问题 Q 用谓词公式表示，并否定 Q, 再与 ANSWER 构成析取式 (~Q V ANSWER);

(3) 把谓词公式集 {F , (~Q V ANSWER)} 化为子句集 S;

(4) 对 S 应用归结原理进行归结;

(5) 若得到归结式 ANSWER, 则答案就在 ANSWER 中。

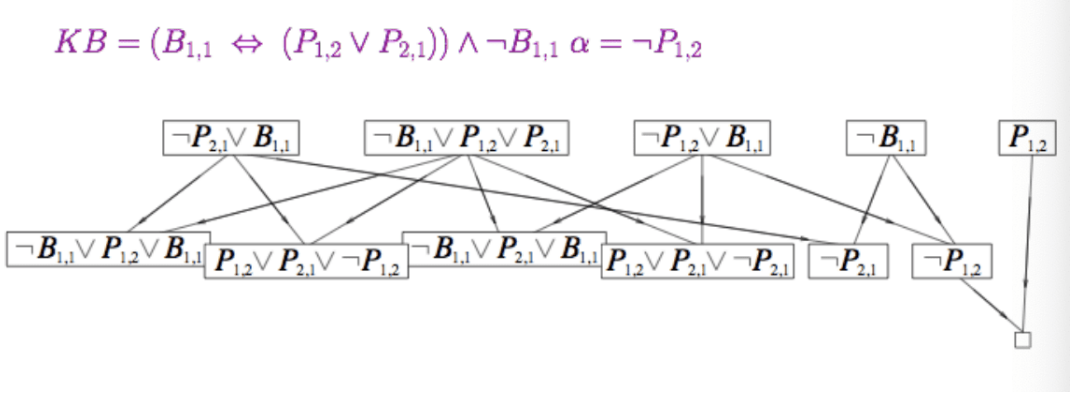


图 1: 归结原理示意图

# 4 MFC 简介

MFC 是微软基础类库，以 C++ 类的形式封装了 Windows API，并且包含一个应用程序框架。类中包含了大量的 windows 句柄封装类和很多 windows 的组件和内建控件的封装类。MFC 把 Windows SDK API 函数包装成了几百个类，MFC 给 Windows 系统提供面向对象的接口，支持可重用性、自包含性以及 OPP 原则。

# 5 实验设计

## 5.1 实验环境

表 1: 实验环境

| 环境 | 版本 |
|---|---|
| Windows 系统 | 10.0.19042 |
| CPU | Intel i7-8750H 2.2GHz |
| Visual Studio | 2019 |
| Qt | 5.15 |

## 5.2 谓词公式表示

表 2: 谓词公式表示

| 谓词公式 | 含义 |
|---|---|
| K(A,B) | A killed B |
| H(A,B) | A hates B |
| R(A,B) | A richer than B |

**破案问题**在一栋房子里发生了一件神秘的谋杀案，现在可以肯定以下几点事实：

(a) 在这栋房子里仅住有 A,B,C 三人；

(b) 是住在这栋房子里的人杀了 A；

(c) 谋杀者非常恨受害者；

(d)A 所恨的人，C 一定不恨；

(e) 除了 B 以外，A 恨所有的人；

(f)B 恨所有不比 A 富有的人；

(g)A 所恨的人，B 也恨；
(h) 没有一个人恨所有的人；
(i) 杀人嫌疑犯一定不会比受害者富有；
(j)A 不等于 B。

**逻辑谓词形式表示**

(b)K(A,A)|K(B,A)|K(C,A)
(c)K(x,A)→H(x,A) 消除蕴含词：∼K(x,A)|H(x,A)
(d)H(A,x)→∼H(C,x) 消除蕴含词：∼H(A,x)| H(C,x)
(e)H(A,A) , H(A,C)
(f)∼R(x,A)→H(B,x) 消除蕴含词：R(x,A)|H(B,x)
(g)H(A,x)→H(B,x) 消除蕴含词：∼H(A,x)|H(B,x)
(h)∼H(A,A)|∼H(A,B)|∼H(A,C)
∼H(B,A)|∼H(B,B)|∼H(B,C)
∼H(C,A)|∼H(C,B)|∼H(C,C)
(i)K(x,A)→∼R(x,A) 消除蕴含词后：∼K(x,A)|∼R(x,A)

**子句集 S** 一共有 11 个子句：
S={ K(A,A)|K(B,A)|K(C,A),
∼K(x,A)|H(x,A),
∼H(A,x)|∼H(C,x),
H(A,A),
H(A,C),
R(x,A)|H(B,x),
∼H(A,x)|H(B,x),
∼H(A,A)|∼H(A,B)|∼H(A,C),
∼H(B,A)|∼H(B,B)|∼H(B,C),
∼H(C,A)|∼H(C,B)|∼H(C,C),
∼K(x,A)|∼R(x,A) }

## 5.3 功能函数设计

表 3

| 函数名称 | 功能说明 |
| --- | --- |
| bool find_function(string name, int& id) | 检查函数名是否储存在 vector[1] |
| bool find_identifier(string name, identifier& it) | 检查常量或变量是否储存在 vector[1] |
| void init() | 初始化常量和变量<br>默认常量为 A,B,C, 变量为 x,y,z[1] |
| bool input_atom(clause& pc, string& clause_input, int atom_start, int atom_end) | 分离一个原子句，在以下两个子句输入函数中进行调用[1] |
| bool input_clause() | 输入子句集[1] |
| bool input_target() | 输入目标子句[1] |
| bool operator< (identifier a, identifier b) | 运算符重载，便于进行常量变量的比较[2] |
| ostream& operator« (ostream& out, clause& c) | 输出流重载，便于直接输出子句[2] |
| bool isequal(atom& atom1, atom& atom2) | 判断函数类型是否一致，从而便于归结过程中进行替换[2] |
| bool is_child(int a, int b) | 判断 a 是否是 b 的孩子结点，即 a 是否能由 b 归结得到[3] |
| int create_new_clause(int parent_2, int parent_1, int todelete)[4] | 归结产生新子句[3] |
| int replace(int num)[5] | 寻找可以被替代并产生新子句的子句, 即 parent_1[3] |
| int resolution() | 归结函数, 寻找事实子句，即 parent_2[3] |
| void print(clause& pc) | 打印归结过程[3] |

[1] 子句输入相关函数
[2] 子句处理相关函数
[3] 归结过程函数
[4] parent_1 为被替换的子句，parent_2 为事实子句，todelete 是待删除的原子句
[5] 其中 num 为事实子句的位置

## 5.4 程序交互界面

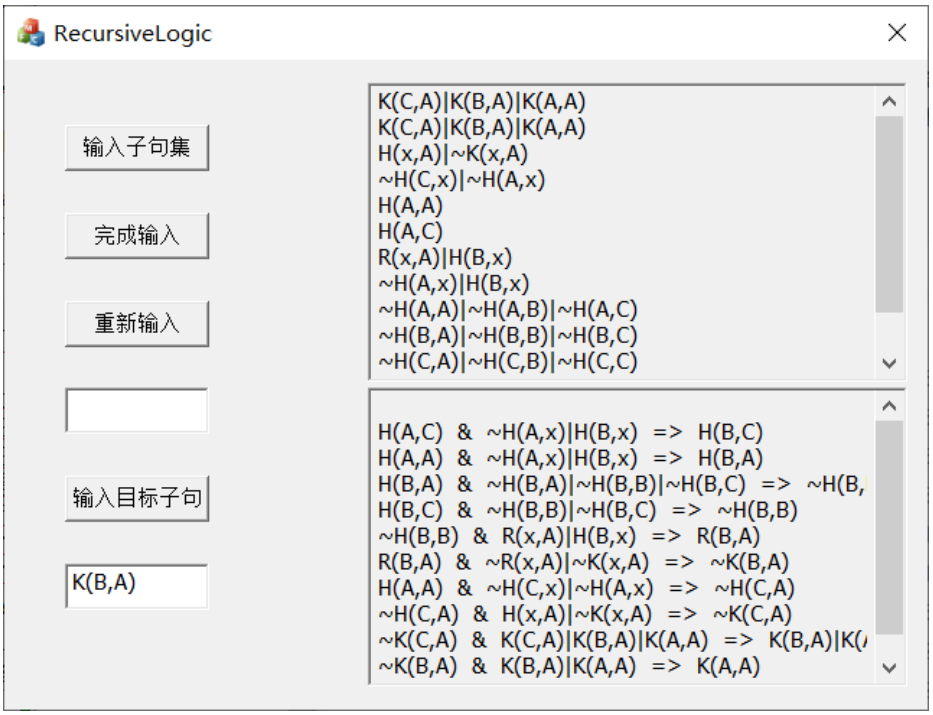


图 2: 程序交互界面

## 5.5 界面功能函数

Listing 1: **MFC** 头文件

```
//mfc头文件
#include<afxwin.h>

//1、应用程序类CWinApp,MyAPP派生类
class MyApp : public CWinApp{
public:
    //父类的虚函数，需要派生类进行重写
    //也是MFC函数的入口函数
    virtual BOOL InitInstance();
};

//2、框架类CFrameWnd，MyFrame派生类
class MyFrame : public CFrameWnd{
public:
    //构造函数
```

```
16      MyFrame();
17  }
```

Listing 2: **MFC** 程序入口

```
1   //对应类的头文件
2   #include "mfc.h"
3
4   //有且只有一个的全局应用程序类的对象
5   MyApp app;
6   //程序入口
7   BOOL MyApp::InitInstance(){
8       //1、创建类框架
9       MyFrame *frame = new MyFrame;
10      //2、显示窗口
11      frame -> ShowWindow(SW_SHOWNORMAL);
12      //3、更新窗口
13      frame -> UpdateWindow();
14      //4、保存框架类对象指针
15      m_pMainWnd = frame;
16      return TRUE;
17  }
18
19  //重写构造函数
20  MyFrame::MyFrame(){
21      //创建窗口
22      Create(NULL,TEXT("mfc"));
23  }
```

# 6 实验评价

## 6.1 实验分工

表 4: 实验分工

| 实验内容 | 姓名 |
|---|---|
| 归结算法设计 | 唐骏龙 |
| 程序交互界面 | 赵昊堃 |
| 报告与答辩 PPT | 苗成林 |

## 6.2 实验问题与解决方案

**问题 1** 如何方便地将所输入的子句集进行分离并分别储存？

**解决方案**考虑首先建立变量常量、原子句和子句的结构体，再借助 C++ 中的 vector 容器来进行存储，即初始化几个 vector 用于存储变量常量、函数名以及子句，利用 vector 中的 iterator 迭代器来帮助遍历 vector 中的元素，若发现当前输入的元素未存储，则可以借助 vector 中自带的函数，如 push_back 等将其存入容器内。

**问题 2** 如何实现子句归结过程？

**解决方案**考虑先调用 resolution 函数遍历当前子句集，查找事实子句 (即只有一个函数且其中只含有常量的子句，如 H(A,C))，再调用 replace 函数查找可以被替代的子句，即含有与事实子句符号相反原子句 (如 ~H(A,C)) 的子句，两个子句都找到后，最后调用 create_new_clause 函数实现替换归结，替换过程中考虑使用 C++ 中的 map 容器构建变量与常量的键值对，方便进行替换。实例化一个新的子句，将其父母结点设为上文提到的两个子句，再将子句中的变量 (如果存在) 全部替换成常量，即可生成一个新的子句，最后将其加入到子句集中即可。

**问题 3** 如何将归结后的子句进行输出？

**解决方案**考虑使用输出流重载的方式将子句进行输出，重载函数中具体方法为从左到右遍历该子句，先判断子句结构体中的符号，然后通过相应位置判断输出左右括号，常量变量以及取或符号，这样一来就可以直接在 cout 语句中输出整个子句的内容。在打印归结过程时通过调用递归函数 print，从归结树的根开始自上而下打印归结过程。

## 6.3 归结算法优化

**包容归结策略**核心是要清楚所有被知识库中的已有语句包容（即，比该语句更特例）的语句。换言之，该法的核心就是要保证每一次新加入的子句都要包含

原子句中所不具有的新信息——所谓新信息就是这条新语句不能由当前子句集中已有的条件推导为永真句。以破案问题的子句举例，破案问题中存在如下子句：

C2: ∼S(x1,A)|H(x1,A)

C3: ∼H(A,x2)|∼H(C,x2)

T1: ∼S(A,A)

我们可以很容易发现 C2 与 C3 可以进行子句归结，归结结果为：

p: ∼S(A,A)|∼H(C,A)

在不使用包容归结策略的前提下，根据最基本的 BFS 策略，p 子句不与 lines 中的任意一个子句重复，所以应当入栈。但当我们仔细查看 p 子句时，不难发现因为 T1 为真，所以 p 的一个文字 ∼S(A,A) 也为真，所以 p 子句其实本质上是一个永真句。

正是因为 T1 子句的存在，而且 T1 子句包容了 p 子句的内容，所以 p 子句的引入将不会再携带有任何新的信息，在这种情况下，就应该放弃 p 子句的引入，以使尽可能少的子句被压入栈，从而降低搜索节点数。

该功能的实现就是将每一个试图入栈的语句进行先进行一次分析，将 lines 中目前已存的所有子句与 p 子句进行比较分析，如果栈中存在一个子句的所有文字都在 p 中有与之相同的文字对应，那么 p 子句就被该语句包含，则不应该让该语句入栈。

除此以外，值得一提的是，由于包容归结策略所拒绝入栈的子句都是不携带新信息的子句，换言之，该策略没有拒绝任一携带新信息的子句入栈，所以该策略是完备的，不会存在因为优化而导致归结推理出错的情况。

# 附录

Listing 3: 源代码

```

// RecursiveLogicDlg.cpp : 实现文件
//

#include "stdafx.h"
#include "RecursiveLogic.h"
#include "RecursiveLogicDlg.h"
#include "afxdialogex.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif


#include <iostream>
#include <vector>
#include <string>
#include <map>
using namespace std;

vector<string> cname;//常量名
vector<string> vname;//变量名
vector<string> fname;//函数名

                                        //常量与变量
enum identifier_type { variable, constant };
typedef struct {
        identifier_type type;//区分常量或变量
        int id;//编号
}identifier;

//原子式
typedef struct {
        vector<identifier> element;//原子式中包含的常量和变量
```

```
35          int function_id;//函数编号
36          bool ispos;//判断是否有取反符号
37  }atom;
38
39  //子句
40  typedef struct {
41          vector<atom> element;//子句中包含的原子句，若个数为1，说明为
                事实子句，可对其他子句进行替换
42          int parent_1;//来源1，一般为被替换的复合子句，没有则置为-1
43          int parent_2;//来源2，一般为进行替换的事实子句，没有则置为-1
44          bool isvalid;//判断事实子句是否还能进行替换
45  }clause;
46
47  vector<clause> clause_set;//子句集
48  clause target_clause;//目标子句
49
50
51
52
53  //检查函数是否已经储存在vector中
54  bool find_function(string name, int& id)
55  {
56          vector<string>::iterator func = find(fname.begin(), fname.end
                (), name);
57          if (func == fname.end())
58                  return false; //未找到
59          else
60          {
61                  id = func - fname.begin();
62                  return true;  //找到并取出其id
63          }
64  }
65
66  //检查常量或变量是否已经储存在vector中
67  bool find_identifier(string name, identifier& it)
68  {
69          //常量
```

```
70          vector<string>::iterator iden = find(cname.begin(), cname.end
                (), name);
71          if (iden != cname.end())
72          {
73                  it.id = iden - cname.begin();
74                  it.type = constant;
75                  return true;
76          }
77          //变量
78          iden = find(vname.begin(), vname.end(), name);
79          if (iden != vname.end())
80          {
81                  it.id = iden - vname.begin();
82                  it.type = variable;
83                  return true;
84          }
85
86          //MessageBox( "输入错误，子句中含有未规定的变量/常量名！" );
87          return false;
88  }
89
90  //读入一个原子句
91  bool input_atom(clause& pc, string& clause_input, int atom_start, int
        atom_end)
92  {
93          int identifier_start, identifier_end, i;
94          atom presnet_atom;
95          //判断是否有取反符号
96          if (clause_input[atom_start] == '~')
97          {
98                  presnet_atom.ispos = false;
99                  atom_start++;
100         }
101         //取出函数名
102         string present_fname;
103         for (i = atom_start; i < atom_end; i++)
104                 if (clause_input[i] == '(')
```

```
105                 {
106                         present_fname = clause_input.substr(
                                atom_start, i - atom_start);
107                         identifier_start = i + 1;
108                         break;
109                 }
110
111         if (i == atom_end)
112         {
113                 cout << "输入不符合子句格式！" << endl;
114                 return false;
115         }
116         //取出函数名在vector中的id
117         int id;
118         bool find = find_function(present_fname, id);
119         if (!find)
120         {
121                 fname.push_back(present_fname);
122                 presnet_atom.function_id = fname.size() - 1;  //未找
                        到则存入
123         }
124         else
125                 presnet_atom.function_id = id;  //找到则取出其id
126
127                                                                 //
                                                                    外
                                                                    层
                                                                    调
                                                                    用
                                                                    时
                                                                    取
                                                                    参
                                                                    量
128         for (i = identifier_start; i < atom_end; i++)
129                 if ((clause_input[i] == ',') || (clause_input[i] == '
                        )'))
130                 {
```

```
131                         identifier_end = i;
132                         identifier it;
133                         //将常量或变量存入原子句的element中
134                         if (!find_identifier(clause_input.substr(
                                identifier_start, identifier_end -
                                identifier_start), it))
135                             return false;
136                         presnet_atom.element.push_back(it);
137                         identifier_start = i + 1;
138                 }
139
140         pc.element.push_back(presnet_atom);
141         return true;
142 }
143
144 //输入子句
145 bool input_clause()
146 {
147         cout << "请逐行输入子句集( | 为或， ~ 为非)，以另起一行的#结
                束: " << endl;
148         cout << "声明: 可使用变量为x,y,z,常量为A,B,C" << endl;
149         cout << "（以下为破案问题子句集，可直接复制使用）" << endl;
150         cout << "K(C,A)|K(B,A)|K(A,A)" << endl;
151         cout << "H(x,A)|~K(x,A)" << endl;
152         cout << "~H(C,x)|~H(A,x)" << endl;
153         cout << "H(A,A)" << endl;
154         cout << "H(A,C)" << endl;
155         cout << "R(x,A)|H(B,x)" << endl;
156         cout << "~H(A,x)|H(B,x)" << endl;
157         cout << "~H(A,A)|~H(A,B)|~H(A,C)" << endl;
158         cout << "~H(B,A)|~H(B,B)|~H(B,C)" << endl;
159         cout << "~H(C,A)|~H(C,B)|~H(C,C)" << endl;
160         cout << "~R(x,A)|~K(x,A)" << endl;
161         cout << "#" << endl;
162         while (1)
163         {
164                 clause pc;
```

```
165                 pc.parent_1 = -1;
166                 pc.parent_2 = -1;
167                 pc.isvalid = true;
168
169                 string clause_input;
170                 cin >> clause_input;
171                 if (clause_input == "#")
172                         break;
173
174                 unsigned int i, atom_start = 0, atom_end = 0;
175                 for (i = 0; i < clause_input.size(); i++)
176                         if ((clause_input[i] == '|') || (i ==
                            clause_input.size() - 1))
177                         {
178                                 atom_end = (i == clause_input.size()
                                    - 1) ? i + 1 : i;
179                                 if (!input_atom(pc, clause_input,
                                    atom_start, atom_end))
180                                         return false;
181                                 atom_start = i + 1;
182                         }
183
184                 clause_set.push_back(pc);
185         }
186         return true;
187 }
188
189 //输入目标子句
190 bool input_target(string str)
191 {
192         cout << endl;
193         cout << "请输入目标子句（如K(A,A)）: " << endl;
194         target_clause.parent_1 = -1;
195         target_clause.parent_2 = -1;
196         target_clause.isvalid = true;
197
198         string clause_input;
```

```
199         //cin >> clause_input;
200         //(CW2A(m_target)) >> clause_input;
201         clause_input = str;
202         unsigned int i, atom_start = 0, atom_end = 0;
203         for (i = 0; i < clause_input.size(); i++)
204                 if ((clause_input[i] == '|') || (i == clause_input.
                        size() - 1))
205                 {
206                         atom_end = (i == clause_input.size() - 1) ? i
                                + 1 : i;
207                         if (!input_atom(target_clause, clause_input,
                            atom_start, atom_end))
208                                 return false;
209                         atom_start = i + 1;
210                 }
211
212         return true;
213 }
214
215
216
217 //运算符重载
218 bool operator<(identifier a, identifier b)
219 {
220         return a.id > b.id;
221 }
222
223 //输出流重载，便于输出子句
224 string out( clause& c)
225 {
226         string out="";
227         for (unsigned int i = 0; i < c.element.size(); i++)
228         {
229                 if (!c.element[i].ispos)
230                         out += "~";
231                 out += fname[c.element[i].function_id] + "(";
232                 for (unsigned int j = 0; j < c.element[i].element.
```

```
                    size(); j++)
233                 {
234                         if (c.element[i].element[j].type == variable)
235                                 out += vname[c.element[i].element[j].
                                        id];
236                         else
237                                 out += cname[c.element[i].element[j].
                                        id];
238                         if (j != c.element[i].element.size() - 1)
239                                 out += ",";
240                 }
241                 out += ")";
242                 if (i != c.element.size() - 1)
243                         out += "|";
244         }
245
246         return out;
247 }
248
249 //判断函数类型是否一致
250 bool isequal(atom& atom1, atom& atom2)
251 {
252         if ((atom1.function_id == atom2.function_id) && (atom1.
                element.size() == atom2.element.size()))
253         {
254                 for (unsigned int i = 0; i < atom1.element.size() &&
                        i < atom2.element.size(); i++)
255                 {
256                         if ((atom1.element[i].type == variable) || (
                                atom2.element[i].type == variable))
257                                 continue;
258                         if (atom1.element[i].id != atom2.element[i].
                                id)
259                                 return false;
260                 }
261                 return true;
262         }
```

```
263         return false;
264 }
265
266
267
268 //判断a是否是b的孩子结点
269 bool is_child(int a, int b)
270 {
271         if ((clause_set[a].parent_1 == -1) || (clause_set[a].parent_2
                == -1))
272                 return false;
273         if ((clause_set[a].parent_1 == b) || (clause_set[a].parent_2
               == b))
274                 return true;
275         return(is_child(clause_set[a].parent_1, b) || is_child(
               clause_set[a].parent_2, b));
276 }
277
278 //通过归结产生新子句，其中parent_1为被替换的子句，parent_2为事实子
       句，todelete是待删除的原子句
279 int create_new_clause(int parent_2, int parent_1, int todelete)
280 {
281         clause new_clause;
282         map<identifier, identifier> replace_map;//实例化进行替换的键
              值对，前为变量，后为常量
283         for (unsigned int i = 0; i < clause_set[parent_2].element[0].
               element.size(); i++)
284                 if (clause_set[parent_1].element[todelete].element[i
                      ].type == variable)
285                         replace_map[clause_set[parent_1].element[
                              todelete].element[i]] = clause_set[
                              parent_2].element[0].element[i];
286
287         new_clause.element = clause_set[parent_1].element;
288         new_clause.parent_1 = parent_1;
289         new_clause.parent_2 = parent_2;
290         new_clause.isvalid = true;
```

```
291         new_clause.element.erase(vector<atom>::iterator(new_clause.
                element.begin() + todelete));
292
293         for (unsigned int i = 0; i < new_clause.element.size(); i++)
294                 for (unsigned int j = 0; j < new_clause.element[i].
                        element.size(); j++)
295                         if ((new_clause.element[i].element[j].type ==
                                variable) &&
296                                 (replace_map.end() != replace_map.
                                    find(new_clause.element[i].
                                    element[j])))
297                                 new_clause.element[i].element[j] =
                                    replace_map[new_clause.element[i
                                    ].element[j]];
298
299         if (new_clause.element.size() == 0)
300                 return -1;  //归结出空子句，即原子句存在逻辑矛盾，返
                    回-1
301
302         clause_set.push_back(new_clause);   //将新子句加入到子句集中
303
304         if (new_clause.element.size() == target_clause.element.size()
                &&
305                 isequal(new_clause.element[0], target_clause.element
                    [0]) &&
306                 new_clause.element[0].ispos == target_clause.element
                    [0].ispos)
307                 return 1;  //归结出的子句与目标子句完全一致，归结成
                    功，返回1
308
309         return 0;
310 }
311
312 //寻找可以被替代并产生新子句的子句，即parent_1，其中num为事实子句的位
    置
313 int replace(int num)
314 {
```

```
315         for (unsigned int i = 0; i < clause_set.size(); i++)
316                 if (clause_set[i].isvalid && !is_child(num, i))
317                 {
318                         for (unsigned int j = 0; j < clause_set[i].
                                element.size(); j++)
319                             if (isequal(clause_set[num].element
                                    [0], clause_set[i].element[j]) &&
320                                 clause_set[num].element[0].
                                    ispos != clause_set[i].
                                    element[j].ispos) //两个
                                    子句符号相反，可抵消
321                             {
322                                 int state = create_new_clause
                                    (num, i, j);
323                                 if (state == -1 || state ==
                                    1)
324                                     return state;
325                             }
326                 }
327
328         return 0;
329 }
330
331 //归结函数,寻找事实子句，即parent_2
332 int resolution()
333 {
334         for (unsigned int i = 0; i < clause_set.size(); i++)
335                 if (clause_set[i].element.size() == 1) //子句中只有一
                    个原子句，说明该子句为事实子句
336                 {
337                         clause_set[i].isvalid = false; //说明已被使用
338                         int state = replace(i);
339                         if (state == -1 || state == 1)
340                             return state;
341                 }
342
343         return 0;
```

```
344 }
345
346 //打印归结过程
347 string print(clause& pc)
348 {
349         if (pc.parent_2 == -1 && pc.parent_1 == -1)
350                 return "";
351         string s1 = print(clause_set[pc.parent_2]);
352         string s2 = print(clause_set[pc.parent_1]);
353
354         return s1 + s2 + out(clause_set[pc.parent_2]) + "  &  " + out
            (clause_set[pc.parent_1]) + "  =>  " + out(pc) + "\r\n";
355 }
356
357
358 // 用于应用程序“关于”菜单项的 CAboutDlg 对话框
359
360 class CAboutDlg : public CDialogEx
361 {
362 public:
363         CAboutDlg();
364
365 // 对话框数据
366 #ifdef AFX_DESIGN_TIME
367         enum { IDD = IDD_ABOUTBOX };
368 #endif
369
370         protected:
371         virtual void DoDataExchange(CDataExchange* pDX);    // DDX/
            DDV 支持
372
373 // 实现
374 protected:
375         DECLARE_MESSAGE_MAP()
376 };
377
378 CAboutDlg::CAboutDlg() : CDialogEx(IDD_ABOUTBOX)
```

```
379 {
380 }
381
382 void CAboutDlg::DoDataExchange(CDataExchange* pDX)
383 {
384         CDialogEx::DoDataExchange(pDX);
385 }
386
387 BEGIN_MESSAGE_MAP(CAboutDlg, CDialogEx)
388 END_MESSAGE_MAP()
389
390
391 // CRecursiveLogicDlg 对话框
392
393
394
395 CRecursiveLogicDlg::CRecursiveLogicDlg(CWnd* pParent /*=NULL*/)
396         : CDialogEx(IDD_RECURSIVELOGIC_DIALOG, pParent)
397         , m_subs(_T(""))
398         , m_target(_T(""))
399         , m_subsout(_T(""))
400         , m_targetout(_T(""))
401 {
402         m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
403         string x = "x";
404         string y = "y";
405         string z = "z";
406         string A = "A";
407         string B = "B";
408         string C = "C";
409         vname.push_back(x);
410         vname.push_back(y);
411         vname.push_back(z);
412         cname.push_back(A);
413         cname.push_back(B);
414         cname.push_back(C);
415         m_inputfinished = false;
```

```
416         m_needclear = false;
417
418
419         m_subsout += "请逐行输入子句集( | 为或， ~ 为非)，以另起一行
                的#结尾: \r\n";
420         m_subsout += "声明: 可使用变量为x,y,z,常量为A,B,C\r\n";
421         m_subsout +=  "（以下为破案问题子句集，可直接复制使用）\r\n";
422         m_subsout += "K(C,A)|K(B,A)|K(A,A)\r\n";
423         m_subsout += "H(x,A)|~K(x,A)\r\n";
424         m_subsout += "~H(C,x)|~H(A,x)\r\n";
425         m_subsout += "H(A,A)\r\n";
426         m_subsout += "H(A,C)\r\n";
427         m_subsout +=  "R(x,A)|H(B,x)\r\n";
428         m_subsout += "~H(A,x)|H(B,x)\r\n";
429         m_subsout += "~H(A,A)|~H(A,B)|~H(A,C)\r\n";
430         m_subsout += "~H(B,A)|~H(B,B)|~H(B,C)\r\n";
431         m_subsout += "~H(C,A)|~H(C,B)|~H(C,C)\r\n";
432         m_subsout += "~R(x,A)|~K(x,A)\r\n";
433         m_subsout += "#\r\n";
434
435         m_subsout = "";
436 }
437
438 void CRecursiveLogicDlg::DoDataExchange(CDataExchange* pDX)
439 {
440         CDialogEx::DoDataExchange(pDX);
441         DDX_Text(pDX, IDC_EDITSUBS, m_subs);
442         DDX_Text(pDX, IDC_EDITTARGET, m_target);
443         DDX_Text(pDX, IDC_SUBSOUT, m_subsout);
444         DDX_Text(pDX, IDC_TARGETOUT, m_targetout);
445 }
446
447 BEGIN_MESSAGE_MAP(CRecursiveLogicDlg, CDialogEx)
448         ON_WM_SYSCOMMAND()
449         ON_WM_PAINT()
450         ON_WM_QUERYDRAGICON()
451         ON_BN_CLICKED(IDC_SUBS, &CRecursiveLogicDlg::OnBnClickedSubs)
```

```
452         ON_BN_CLICKED(IDC_TARGET, &CRecursiveLogicDlg::
                OnBnClickedTarget)
453         ON_BN_CLICKED(IDC_FINISH, &CRecursiveLogicDlg::
                OnBnClickedFinish)
454         ON_BN_CLICKED(IDC_CLEAR, &CRecursiveLogicDlg::
                OnBnClickedClear)
455 END_MESSAGE_MAP()
456
457
458 // CRecursiveLogicDlg 消息处理程序
459
460 BOOL CRecursiveLogicDlg::OnInitDialog()
461 {
462         CDialogEx::OnInitDialog();
463
464         // 将“关于...”菜单项添加到系统菜单中。
465
466         // IDM_ABOUTBOX 必须在系统命令范围内。
467         ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
468         ASSERT(IDM_ABOUTBOX < 0xF000);
469
470         CMenu* pSysMenu = GetSystemMenu(FALSE);
471         if (pSysMenu != NULL)
472         {
473                 BOOL bNameValid;
474                 CString strAboutMenu;
475                 bNameValid = strAboutMenu.LoadString(IDS_ABOUTBOX);
476                 ASSERT(bNameValid);
477                 if (!strAboutMenu.IsEmpty())
478                 {
479                         pSysMenu->AppendMenu(MF_SEPARATOR);
480                         pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
                                strAboutMenu);
481                 }
482         }
483
484         // 设置此对话框的图标。  当应用程序主窗口不是对话框时，框架将
```

```
            自动
485         //  执行此操作
486         SetIcon(m_hIcon, TRUE);                 // 设置大图标
487         SetIcon(m_hIcon, FALSE);                // 设置小图标
488
489         // TODO: 在此添加额外的初始化代码
490
491         return TRUE;  // 除非将焦点设置到控件，否则返回 TRUE
492 }
493
494 void CRecursiveLogicDlg::OnSysCommand(UINT nID, LPARAM lParam)
495 {
496         if ((nID & 0xFFF0) == IDM_ABOUTBOX)
497         {
498                 CAboutDlg dlgAbout;
499                 dlgAbout.DoModal();
500         }
501         else
502         {
503                 CDialogEx::OnSysCommand(nID, lParam);
504         }
505 }
506
507 // 如果向对话框添加最小化按钮，则需要下面的代码
508 //  来绘制该图标。  对于使用文档/视图模型的 MFC 应用程序，
509 //  这将由框架自动完成。
510
511 void CRecursiveLogicDlg::OnPaint()
512 {
513         if (IsIconic())
514         {
515                 CPaintDC dc(this); // 用于绘制的设备上下文
516
517                 SendMessage(WM_ICONERASEBKGND, reinterpret_cast<
                        WPARAM>(dc.GetSafeHdc()), 0);
518
519                 // 使图标在工作区矩形中居中
```

```
520                 int cxIcon = GetSystemMetrics(SM_CXICON);
521                 int cyIcon = GetSystemMetrics(SM_CYICON);
522                 CRect rect;
523                 GetClientRect(&rect);
524                 int x = (rect.Width() - cxIcon + 1) / 2;
525                 int y = (rect.Height() - cyIcon + 1) / 2;
526
527                 // 绘制图标
528                 dc.DrawIcon(x, y, m_hIcon);
529         }
530         else
531         {
532                 CDialogEx::OnPaint();
533         }
534 }
535
536 //当用户拖动最小化窗口时系统调用此函数取得光标
537 //显示。
538 HCURSOR CRecursiveLogicDlg::OnQueryDragIcon()
539 {
540         return static_cast<HCURSOR>(m_hIcon);
541 }
542
543
544
545 void CRecursiveLogicDlg::OnBnClickedSubs()
546 {
547         // TODO: 在此添加控件通知处理程序代码
548
549         UpdateData(true);
550
551         char buf[512] = { 0 };
552         string clause_input = (const char *)m_subs;// .GetBuffer(
            m_subs.GetLength());
553
554         clause pc;
555         pc.parent_1 = -1;
```

```
556         pc.parent_2 = -1;
557         pc.isvalid = true;
558
559         if (m_needclear)
560         {
561                 m_subsout = "";
562                 m_needclear = false;
563                 clause_set.clear();
564         }
565
566         if (m_inputfinished)
567         {
568                 if (IDYES == MessageBox("确定要放弃所有已输入的子句,
                        重新开始输入吗？", "重新开始输入", MB_YESNO))
569                 {
570                         m_inputfinished = false;
571                         clause_set.clear();
572                         m_subsout = "";
573                 }
574                 else
575                         return;
576         }
577
578         int start = 0;
579         do
580         {
581                 unsigned int i, atom_start = 0, atom_end = 0;
582                 for (i = 0; i < clause_input.size(); i++)
583                         if ((clause_input[i] == '|') || (i ==
                                clause_input.size() - 1))
584                         {
585                                 atom_end = (i == clause_input.size()
                                        - 1) ? i + 1 : i;
586                                 if (!input_atom(pc, clause_input,
                                        atom_start, atom_end))
587                                 {
588                                         MessageBox("子句输入错误！");
```

```
589                                         return;
590                                 }
591                                 atom_start = i + 1;
592                         }
593
594                 clause_set.push_back(pc);
595         } while (0);// m_subs.Find("\r\n", start)>0);
596
597         m_subsout += m_subs + "\r\n";
598         m_subs = "";
599         UpdateData(false);
600 }
601
602
603 void CRecursiveLogicDlg::OnBnClickedTarget()
604 {
605         // TODO: 在此添加控件通知处理程序代码
606         UpdateData(true);
607
608         if (m_inputfinished == false)
609         {
610                 MessageBox("请先输入子句集！");
611                 return;
612         }
613
614         string s=m_target.GetBuffer(0);
615         if (!input_target(s))
616                 MessageBox("目标子句输入错误！");
617
618         m_targetout = "\r\n";
619
620         int state = resolution();
621         m_targetout += print(clause_set.back()).c_str();
622
623         if (state == 1)
624                 m_targetout += "归结成功！";
625         else if (state == 0)
```

```
626                 m_targetout += "归结失败！";
627         else if (state == -1)
628                 m_targetout += "子句逻辑出现错误!";
629
630         m_inputfinished = false;
631         m_needclear = true;
632         UpdateData(false);
633 }
634
635
636 void CRecursiveLogicDlg::OnBnClickedFinish()
637 {
638         // TODO: 在此添加控件通知处理程序代码
639         if (1)
640         {
641                 m_inputfinished = true;
642                 MessageBox("可以输入目标子句进行测试了！");
643                 return;
644         }
645 }
646
647
648 void CRecursiveLogicDlg::OnBnClickedClear()
649 {
650         // TODO: 在此添加控件通知处理程序代码
651         m_subsout = "";
652         UpdateData(false);
653         clause_set.clear();
654 }
```