《人工智能课程设计》实验三：一阶逻辑归结实验

装 订 线

学　　号 1953080
姓　　名 田宇
专　　业 计算机科学与技术
授课老师 王俊丽

目录

装 订 线

1 实验概述

1.1 实验目的

熟悉和掌握归结原理的基本思想和基本方法，通过实验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。

1.2 实验内容及要求

(1) 对所给问题进行知识的逻辑表示，转换为子句，对子句进行归结求解。

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。

为了推理需要，增加如下常识：(j)A 不等于 B。

问：谋杀者是谁？

(2)选用一种编程语言，在逻辑框架中利用归结原理进行求解。

(3)要求界面显示每一步的求解过程。

(4)撰写实验报告，提交源代码（进行注释）、实验报告、汇报 PPT

2 实验方案设计

2.1 总体设计思路与总体框架

2.1.1 总体设计思路

通过设计 C++命令行程序完成一阶逻辑归结算法的实现，求解要求的题目，并对求解结果进行展示。首先手动将题目的自然语言逻辑语句整理为一阶逻辑子句，然后使用通过置换与合一算法，对子句进行不断地归结，最后根据归结情况输出相应结果。由于本题没有太多人机交互的需求，因此使用命令行程序进行展示，根据提示输入子句后按回车进行归结并输出结果。

2.1.2 具体做法

(1) 定义谓词和常量

设

Inhouse(x) 表示 x 住在房子里；

Kill(x,y) 表示 x 杀了 y；

Hate(x,y) 表示 x 恨 y；

Richer(x,y) 表示 x 比 y 富有；

Is(x,y) 表示 x 和 y 是同一人

A、B、C 表示房子里的 3 个人

(2) 把问题表示成谓词公式

自然语言描述	谓词公式
在这栋房子里仅住有 A,B,C 三人	∀x Inhouse(x)->(Is(x,A) ∨ Is(x,B) ∨ Is(x,C))
是住在这栋房子里的人杀了 A	∀x Kill(x,A)->Inhouse(x)
谋杀者非常恨受害者	∀x Kill(x,A)->Hate(x,A)
A 所恨的人，C 一定不恨	∀x Hate(A,x)->¬Hate(C,x)
除了 B 以外，A 恨所有的人	∀x ¬Is(x,B)->Hate(A,x)
B 恨所有不比 A 富有的人	∀x ¬Richer(x,A)->Hate(B,x)
A 所恨的人，B 也恨	∀x Hate(A,x)->Hate(B,x)
没有一个人恨所有的人	¬∃x ∀y Hate(x,y)
杀人嫌疑犯一定不会比受害者富有	∀x Kill(x,A)->¬Richer(x,A)
A 不等于 B	¬Is(A,B) ¬Is(B,A)

(3) 将谓词公式转化为子句集

Kill(A,A)|Kill(B,A)|Kill(C,A)

~Kill(x1,A)|Hate(x1,A)

~Hate(A,x2)|~Hate(C,x2)

Hate(A,A)

Hate(A,C)

Richer(x3,A)|Hate(B,x3)

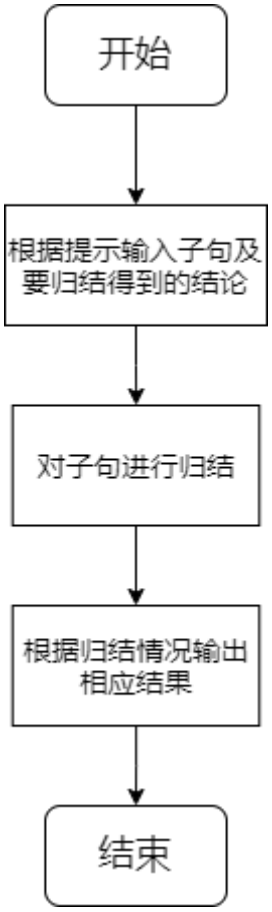
~Hate(A,x4)|Hate(B,x4)

~Hate(x5,A)|~Hate(x5,B)|~Hate(x5,C)

~Kill(x6,A)|~Richer(x6,A)

(4) 应用归结原理进行归结（通过代码实现）

2.1.3 总体思路流程图



2.2 核心算法及基本原理

本实验的核心算法为归结反演算法，在归结算法的实现过程中需要用到置换与合一算法。归结反演算法具体过程如下：

(1) 将定理证明的前提谓词公式转化为子句集 F

(2) 将求证的目标表示成合适的谓词公式 G（目标公式）

(3) 将目标公式的否定式~G 转化成子句的形式，并加入到子句集 F 中，得到子句集 S

(4) 应用归结原理对子句集中的子句进行归结，并把每次归结得到的归结式都并入 S 中。如此反复进行，若归结得到一个空子句 NIL，则停止归结，证明了 G 为真。

本题实现思路：

循环枚举子句集中的所有子句，用每一条子句与其他子句进行匹配，并通过置换合一生成新子句加入子句集。如此不断进行，直到归结出空子句或得到矛盾，此时结束循环并输出结果。

2.3 模块设计；

将子句归结封装为一个 Resolution 类，类内描述子句的信息进行相应拆分与封装，进行完

整的子句归结模块设计，具体如下：

2.3.1 参数结构体

```
/*************
 * 参数结构体，用于描述原子句的参数，包括谓词名、常量、变量三种类型
 * */
struct Parameter
{
    static const int PREDICATE = 0, //谓词
        CONSTANT = 1,                //常量
        VARIABLE = 2;                //变量
    int type;                        //参数类型
    string name;                     //参数名字符串

    /*************
     * 空参构造，此时参数类型不为三种类型之一
     * */
    Parameter() : type(-1) {}

    /*************
     * 2 参构造
     * @param type       参数类型
     * @param name       参数名
     * */
    Parameter(int type, const string &name) : type(type), name(name) {}

    /*************
     * 重载<，用于 map
     * @param temp   用于比较的另一个参数对象常引用
     * */
    bool operator<(const Parameter &temp) const { return name < temp.name; }
};
```

2.3.2 原子句结构体

```
/*************
 * 原子句结构体，用于描述一个原子句
 * */
struct Atomic_clause
{
    vector<Parameter> paras; //原子句的参数，其中参数[0]固定为谓词
    bool _true;              //原子句是否为真

    /*************
     * 空参构造，默认为真
     * */
    Atomic_clause() : _true(1) {}

    /*************
     * 展示原子句，TODO：改为<<
     * @param _debug     是否为 debug 模式（可输出更多信息供调试）
     * */
    void show(bool _debug = 0) const;

    /*************
     * 清空原子句
```

```
     * */
    void clear() { paras.clear(), _true = 1; }
};
```

2.3.3 子句结构体

```
/************
 * 子句结构体，用于描述一个子句，一般由多个以或相连的原子句构成
 * */
struct Clause
{
    static const int NONE = -1; //无father或mother
    vector<Atomic_clause> sub;  //原子句数组
    int mother;                 //归结生成该子句的双亲之一
    int father;                 //归结生成该子句的另一个双亲
    bool valid;                 //该子句是否可用

    /************
     * 带默认参数构造函数
     * */
    Clause(int mother = Clause::NONE, int father = Clause::NONE, bool valid = 1
) : mother(mother), father(father), valid(valid) {}

    /************
     * 重载==
     * */
    bool operator==(const Clause &temp) const;

    /************
     * 展示子句，TODO：改为<<
     * @param _debug     是否为debug模式（可输出更多信息供调试）
     * */
    void show(bool _debug = 0) const;
};
```

2.3.4 子句归结类

```
/************
 * 归结类，用于描述一次归结
 * */
class Resolution
{
private:
    static const int HAS_ANS = 1, //归结得到答案
        HAS_PARADOX = -1,         //归结得到悖论
        NO_ANS = 0;               //无法完成归结或还未完成归结

    vector<Clause> clauses; //子句集
    Clause goal_c;          //目标（结论）子句
    int ansno;              //归结答案状态
    int _step;              //用于输出归结步骤时记录步数

    /************
     * 将字符串转为新子句
     * @param s      输入要转为子句的字符串
```

```
     * @param op     默认为 0，将转出的新子句加入子句集；op==1 时表示此时处理的是结论串不加入子句集
     * */
    bool addClause(string s, int op = 0);

    /*************
     * 判断子句{@code c1}是否为子句{@code c2}的祖先子句，即子句{@code c2}是否由{@code c1}归结而来
     * @param c1     子句 c1 在子句集中的下标
     * @param c2     子句 c2 在子句集中的下标
     * */
    bool isAncestor(int c1, int c2);

    /*************
     * 用一个子句对其他子句进行一次归结
     * @param pos    用来归结其他子句的子句在子句集中的位置
     * @return       返回 ansno
     * */
    int oneResolution(int pos);

    /*************
     * 判断两个原子句是否满足合一条件
     * @param a1     原子句 1
     * @param a2     原子句 2
     * */
    bool canMerge(const Atomic_clause &a1, const Atomic_clause &a2);

    /*************
     * 将两个子句进行置换合一
     * @param father     用于置换合一的第一个子句在子句集中下标
     * @param mother     用于置换合一的第二个子句在子句集中下标
     * @param _del       置换合一时合掉的原子句在第二个子句中的下标
     * @return       返回 ansno
     * */
    int mergeClause(int father, int mother, int _del);

    /*************
     * 用于递归输出归结步骤
     * @param c  当前要输出的子句
     * */
    void _outSteps(const Clause &c);

public:
    /*************
     * 空参构造，用于初始化类内变量，输出提示并按照提示输入所有子句和结论
     * */
    Resolution();

    /*************
     * 归结搜索函数，用于实现归结算法
     * @return   返回 ansno
     * */
    int resolutionSearch();

    /*************
     * 先进行判断和初始化，在调用_outStep 函数递归输出归结步骤
     * @return   能否输出归结步骤，若 ansno==NO_ANS 则不能输出
```

```
     * */
    bool outSteps();
};
```

2.4 其他创新内容及优化算法

（1)全部代码采用面向对象的程序设计方法,将子句归结的整个算法封装为一个 Resolution 类，可以有效避免全局变量的使用，大大提高了代码健壮性和可移植性。

（2）采用 Header Only Library 设计理念将所有和求解有关的类和结构体写到一个 hpp 文件中，使代码移植性得到提高。

（3）根据该题特点，可以将 Inhouse 谓词和 Is 谓词去掉，从而简化归结过程。

3 实验过程

3.1 环境说明

3.1.1 操作系统

Windows 10

3.1.2 开发语言

C++

3.1.2 开发环境

编辑器：VS code（当前最新版）

核心使用库：STL 中的 vector 和 map

3.2 源代码文件清单及主要函数清单

(1) main.cpp

程序主体，负责实例化 Resolution 对象并调用归结算法，同时根据归结结果输出提示并判断是否输出归结步骤。

(2) Resolution.hpp

归结算法的核心，包括 Parameter、Atomic_clause、Clause、Resolution 结构体和类的声明和相应函数实现，函数实现包括输入字符串转为子句、完整归结算法以及归结步骤输出。

3.3 实验结果展示

3.3.1 程序展示

初始界面及输入提示如下:

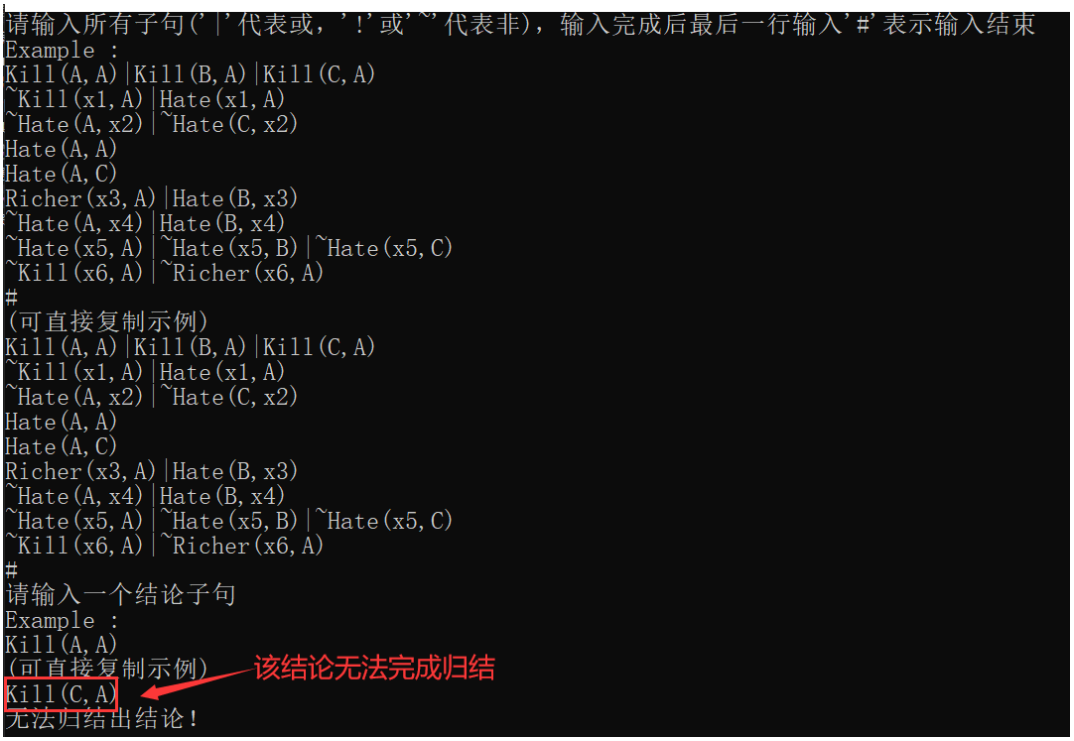
```
请输入所有子句('|'代表或，'!'或'~'代表非)，输入完成后最后一行输入'#'表示输入结束
Example :
Kill(A,A)|Kill(B,A)|Kill(C,A)
~Kill(x1,A)|Hate(x1,A)
~Hate(A,x2)|~Hate(C,x2)
Hate(A,A)
Hate(A,C)
Richer(x3,A)|Hate(B,x3)
~Hate(A,x4)|Hate(B,x4)
~Hate(x5,A)|~Hate(x5,B)|~Hate(x5,C)
~Kill(x6,A)|~Richer(x6,A)
#
(可直接复制示例)
```

输入所有子句和结论后进行归结，若归结成果或得到悖论则输出归结过程

```
~Kill(x1,A)|Hate(x1,A)
~Hate(A,x2)|~Hate(C,x2)
Hate(A,A)
Hate(A,C)
Richer(x3,A)|Hate(B,x3)
~Hate(A,x4)|Hate(B,x4)
~Hate(x5,A)|~Hate(x5,B)|~Hate(x5,C)
~Kill(x6,A)|~Richer(x6,A)
#
请输入一个结论子句
Example :
Kill(A,A)
(可直接复制示例)
Kill(A,A)
归结出结论，具体归结过程如下：
Step1    : Hate(A,C)  &  !Hate(A,x4)|Hate(B,x4)  ==>  Hate(B,C)
Step2    : Hate(A,A)  &  !Hate(A,x4)|Hate(B,x4)  ==>  Hate(B,A)
Step3    : Hate(B,A)  &  !Hate(x5,A)|!Hate(x5,B)|!Hate(x5,C)  ==>  !Hate(B,B)|!Hate(B,C)
Step4    : Hate(B,C)  &  !Hate(B,B)|!Hate(B,C)  ==>  !Hate(B,B)
Step5    : !Hate(B,B)  &  Richer(x3,A)|Hate(B,x3)  ==>  Richer(B,A)
Step6    : Richer(B,A)  &  !Kill(x6,A)|!Richer(x6,A)  ==>  !Kill(B,A)
Step7    : Hate(A,A)  &  !Hate(A,x2)|!Hate(C,x2)  ==>  !Hate(C,A)
Step8    : !Hate(C,A)  &  !Kill(x1,A)|Hate(x1,A)  ==>  !Kill(C,A)
Step9    : !Kill(C,A)  &  Kill(A,A)|Kill(B,A)|Kill(C,A)  ==>  Kill(A,A)|Kill(B,A)
Step10   : !Kill(B,A)  &  Kill(A,A)|Kill(B,A)  ==>  Kill(A,A)
```

若无法完成归结则输出相应提示

装 订 线



3.3.2 实验结论

本次实验按照实验要求，成功实现了归结算法并能用程序求解破案问题，完成了实验要求。

4 总结

4.1 实验中存在问题及解决方案

（1）最处进行实验时对归结算法的理解不够准确，导致无论如何也不能归结出结果，对此我进行了诸多尝试，最终通过查阅资料、与同学讨论成功理解并解决了该问题。

（2）最初试验时使用了 Skolem 函数和 ANSWER 谓词，导致模型较为复杂难以求解，最后在对模型进行简化后成功求解。

（3）最处进行实验时未考虑输出归结步骤的问题，并采用 list 链表存储子句集，导致难以输出结果，最后在 Clause 结构体中加入存储生成该子句的双亲子句变量并将 list 链表改为 vector 容器解决。

4.2 心得体会

本次实验通过自己编写程序完成了一个简单归结算法的实现，并能够应用该程序求解一个破案实际问题，使我对于 AI 分析问题并进行逻辑推理的原理有了更深刻的理解，同时对于编程实现逻辑推理有了许多认识与理解。

同时通过将求解算法封装成类的编程方法，让我对于提升程序健壮性的方法有了新的认识。

4.3 后续改进方向

（1）虽然本题对于人机交互 UI 的需求性较低，但是完全采用命令行的方式实现程序不够美观，应考虑后续添加简单的前端界面。

（2）为了简化模型没有采用 Skolem 函数和 ANSWER 谓词，虽然足够求解该题，但是对于相关归结问题不完全具有通用性，考虑后续重新在模型中加入 Skolem 函数和 ANSWER 谓词提升程序通用性。

（3）本次程序由问题描述到子句的转变全部由人工完成，其中自然语言的处理部分难以直接编程实现，但是可以考虑编程实现谓词公式到子句集的转变。

4.4 实验总结

通过本次实验，我在实践中对归结算法的原理与实现有了更深刻的认识，并在实际问题的求解过程中对于 AI 处理逻辑推理的原理有了更多理解与体会。

5 参考文献

[1] Stuart J. Russell，Peter Norvig. 世界计算机教材精选 人工智能 一种现代的方法 第 3 版. 北京：清华大学出版社，2018.07.

6 成员分工与自评

6.1 成员分工

成员：1953080 田宇(组长)

分工：由组长田宇同学实现本次实验的全部工作（包括程序设计、代码编写实现、报告撰写、汇报 PPT 制作等）

6.2 自评

本次实验总体完成情况良好，能够完成实验要求实现的全部内容。

但是由于我最初对于归结算法理解有误，耽误了较多时间重新理解与学习，导致最后时间不够，实现的结果界面相对不太美观。

总体来看，在独立完成本次实验的过程中我对于归结算法解决实际问题整体上有了全方面的认识，缺点是程序界面不够美观。