|  |
| --- |
| 字句归结实验报告 |
| 《人工智能课程设计》实验三 |
|  |
| **组 长 徐铭骋**  **组 员 谢安南 杨骏恒**  **专 业 计算机科学与技术**  **授课老师 武妍**  **时 间 2021.6.10** |
|  |

目录

[1 实验概述 1](#_Toc74692752)

[1.1 实验目的 1](#_Toc74692753)

[1.2 实验内容 1](#_Toc74692754)

[1.3 问题简介 1](#_Toc74692755)

[2 实验方案设计 2](#_Toc74692756)

[2.1 总体设计思路与总体架构 2](#_Toc74692757)

[2.1.1 总体设计思路 2](#_Toc74692758)

[2.1.2 总体架构 2](#_Toc74692759)

[2.1.3 总体流程图 3](#_Toc74692760)

[2.2 核心算法及基本原理 3](#_Toc74692761)

[2.3 模块设计 4](#_Toc74692762)

[2.4 其他创新内容或优化算法 5](#_Toc74692763)

[3 实验过程 6](#_Toc74692764)

[3.1 环境说明 6](#_Toc74692765)

[3.2 源代码文件清单与主要函数清单 6](#_Toc74692766)

[3.2.1 源代码文件清单 6](#_Toc74692767)

[3.2.2 主要函数清单 6](#_Toc74692768)

[3.3 实验结果展示 6](#_Toc74692769)

[3.3.1 字句归结使用说明及结果展示 6](#_Toc74692770)

[3.3.2 实验结论 16](#_Toc74692771)

[4 总结 17](#_Toc74692772)

[4.1 实验中存在的问题及解决方案 17](#_Toc74692773)

[4.2 心得体会 17](#_Toc74692774)

[4.3 后续改进方向 18](#_Toc74692775)

[4.4 总结 18](#_Toc74692776)

[参考文献 19](#_Toc74692777)

[成员分工与自评 19](#_Toc74692778)

[附 录 20](#_Toc74692779)

# 实验概述

## 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。

## 实验内容

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。

2. 选用一种编程语言，在逻辑框架中实现Horn子句的归结求解。

3. 对下列问题用逻辑推理的归结原理进行求解，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。

## 问题简介

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？

先将以上问题转换为一阶逻辑表示的语句，再通过转换过程转换为CNF的子句形式，再输入到程序中进行归结求解，得到结论是否归结成功。

# 实验方案设计

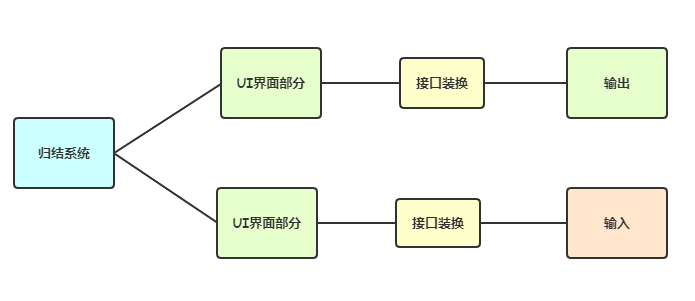
## 总体设计思路与总体架构

## 总体设计思路

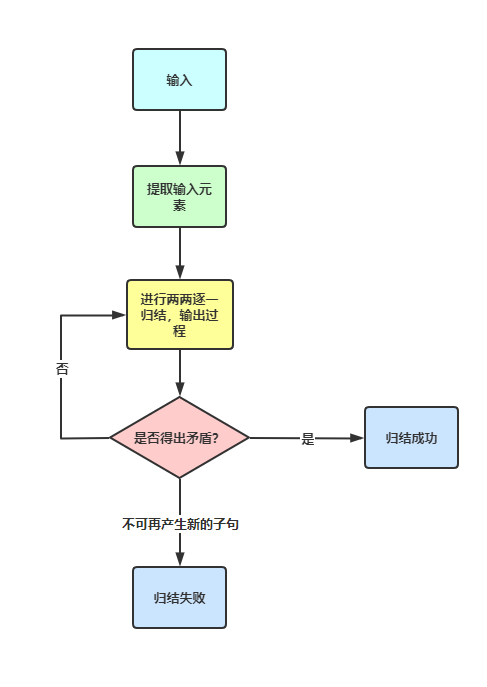
本次实验总体设计思路如下：

1. 定义实体类型用于装载对子句进行元素析取后的结果。例如!Hate(x,y)中将!、Hate、x、y分别提取并存入类型中。这一步为了后续归结或者合一时候的方便。
2. 定义转换接口，目的是为了将输入的字句转换为上述的实体类型，另外最终需要将归结步骤输出，所以定义反向转换接口用于将实体类型化为子句输出。
3. 设计子句合一函数，用于判断两个含一阶字句是否能够合一置换，并输出所有的合一置换结果。
4. 设计归结函数，用于归结子句，该部分为实验主体，最后根据该部分迭代归结，直到推出矛盾或者不能再产生新的归结。
5. 设计界面部分，该部分主要用界面来展示输入输出窗口以及结果显示，将所有的归结步骤显示到屏幕上，用于交互。

## 总体架构



## 总体流程图



## 核心算法及基本原理

归结推理算法

本实验的核心算法为归结推理规则对于一阶逻辑子句的提升版本。

 在命题逻辑的归结算法中，单元归结推理如下图所示：

其中每个l为一个文字，而li与m是互补文字。那么单元归结规则选取一个字句和一个文字，生成一个新的子句。

单元归结规则可推广为全归结规则，



而提升为一阶逻辑后，其归结过程为：对于两个已经完成变量标准化没有共享变量的两个子句，如果包含互补文字则可对它们进行归结。如果一个命题文字是另一个命题文字的否定式，则这两个命题文字是互补的；如果一个一阶逻辑文字能和另一个一阶逻辑文字的否定式合一，则这两个一阶逻辑文字是互补的。由此可得到：

其中UNITY(li,┐mj)=θ。换言之，归结原理的本质就是通过合一置换消除互补文字，从而形成一个新子句的过程。

该过程被称为二元归结规则，它正好对两个文字进行归结。二元归结本身不能产生完备的推理过程。全归结规则对每个可合一的子句中的文字子句进行归结。另外一种方法是把归并也扩展到一阶逻辑。命题逻辑的归并是如果两个文字相同，则将这两个文字减少到一个。一阶逻辑的归并是如果两个文字可合一，则将这两个文字减少到一个。合一置换必须应用于整个子句。二元归结和归并的结合是完备的。

## 模块设计

本次实验有大致以下几个模块：

（1）子句内存互相转换模块

本模块中，能将输入的符合一定要求的子句转换成本系统的内部存储类型并进行后续操作；也能在输出时将内部储存类型转换为析取范式子句进行输出。

（2）字句归结模块

本模块中，将通过循环来归结出新的子句，具体方式为，对初始子句两两进行归结，产生新的子句之后，再从头开始，将原子句与新子句进行归结，新子句自身之间进行归结，如此反复，直到找出矛盾或者无法再产生新的子句为止。

（3）合一模块

本模块中，主要用于判断一阶逻辑子句是否能够归结，而一阶逻辑子句能够归结首先要满足它们能够合一，所以该模块判断了两个子句是否可以合一并且归结，若能，能返回所有的归结结果，若不能，则返回空。

（4）UI界面模块

本模块中，主要实现UI界面框体、按钮与用户的交互，在一定程度上提升用户的交互体验。

该模块的实现逻辑即当用户按下按钮后，该模块就从输入框中读取用户输入的内容，并将相应的内容传入输入解析模块；而在归结推理过程中，该模块还负责输出信息，并将这些信息展示在输出框中。

## 其他创新内容或优化算法

本次实验中，我们用到的主要优化算法部分是归结生成子句后的舍去策略。

首先，若归结出的子句与原子句完全相同，则该子句显然不会被加入子句集中。其次，若新生成的子句被原有子句所蕴含，则新生成的子句不会被加入子句集中。

以破案问题的子句举例，破案问题中存在如下子句：

1: !Kill(x1,A)|H(x1,A)

2: !H(A,x2)|!H(C,x2)

3: !Kill(A,A)

我们可以很容易发现1与2可以进行子句归结，归结结果为：

4: !Kill(A,A)|!H(C,A)

当我们仔细查看4子句时，不难发现因为3为真，所以4的一个文字!Kill(A,A)也为真，所以4子句其实是一个永真句。

正是因为有3号子句的存在，其蕴含了4号子句，所以4号子句的引入将不会产生新的有用的信息，这样，就应该放弃4号子句的产生，以此减少子句的加入，减少搜索的次数，提高整体搜索的效率。

# 实验过程

## 环境说明

|  |  |
| --- | --- |
| 操作系统 | Windows10 |
| 开发语言 | Python, Html, Javascript |
| 开发环境 | IDLE (Python 3.9.4 64-bit) |
| 核心使用库 | Flask框架、jQuery库 |

## 源代码文件清单与主要函数清单

## 源代码文件清单

manage.py: 字句归结主要源文件，用于实现子句的归结算法和对接网页进行结果展示。

index.html: 网页源文件，用于实现网页设计界面和交互效果。

jquery.js: JavaScript 工具库，用于提供的API 兼容浏览器，简化html操作。

## 主要函数清单

convert\_lang(clause\_in) ： 该函数用于将内部存储转换为子句输出。

convert(str\_in ) ： 用于将输入子句转换为内部存储结构。

unify(clause\_a,clause\_b) ： 合一函数，用于判断一阶逻辑子句是否能合一并输出所有的归结合一结果。

resolve(a,b) ： 归结函数主体实现，用于归结子句得出矛盾或者无法继续归结。

index1()：生成前端主页面。

index2()：获取前端用户输入的子句集和归结目标。

## 实验结果展示

### 3.3.1 字句归结使用说明及结果展示

因为本次实验子句归结中，需要先将原始问题转换为子句，按照本小组程序的输入要求，将原始的命题转换成以下子句：

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

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

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

Hate(A,A)

Hate(A,C)

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

!Richer(x4,A)|!Hate(B,x4)

Hate(A,x5)|!Hate(B,x5)

!Hate(A,x6)|Hate(B,x6)

!Hate(A,A)|!Hate(A,B)|!Hate(A,C)

!Hate(B,A)|!Hate(B,B)|!Hate(B,C)

!Hate(C,A)|!Hate(C,B)|!Hate(C,C)

!Richer(x7,A)|!Kill(x7,A)

在python环境下运行程序，登录本地<http://127.0.0.1:5001/>进入程序主界面。

当运行字句归结程序后进入以下画面：



接下来可以开始进行归结，第一次进入界面时已经在原有输入框中完成本次实验破案问题的子句输入，目标输入可以改，下面展示依次归结Kill(A,A)，Kill(B,A)，Kill(C,A)来找出真凶。

首先对Kill(A,A)进行归结：



点击开始归结后，右下角框图中得到归结成功的结果，证明A是自杀的。

再对Kill(B,A)进行归结：



点击开始归结后，右下角框图中得到归结失败的结果，证明A不是B杀的。

同样对Kill(C,A)进行归结：



点击开始归结后，右下角框图中得到归结失败的结果，证明A不是C杀的。

所以可以得到结论，A是自杀的。

以下是归结A自杀的推理过程：

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

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

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

Hate(A,A)

Hate(A,C)

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

!Richer(x4,A)|!Hate(B,x4)

Hate(A,x5)|!Hate(B,x5)

!Hate(A,x6)|Hate(B,x6)

!Hate(A,A)|!Hate(A,B)|!Hate(A,C)

!Hate(B,A)|!Hate(B,B)|!Hate(B,C)

!Hate(C,A)|!Hate(C,B)|!Hate(C,C)

!Richer(x7,A)|!Kill(x7,A)

!Kill(A,A)

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

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

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

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

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

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

!Hate(A,A)|!Kill(C,A)

!Richer(A,A)|!Kill(B,A)

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

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

!Hate(C,A)

!Hate(C,C)

!Hate(B,x5)|!Hate(C,x5)

Hate(B,A)

!Hate(A,B)|!Hate(A,C)

Hate(B,C)

!Hate(A,A)|!Hate(A,B)

!Hate(B,x4)|Hate(B,x4)

Hate(A,x5)|Richer(x5,A)

!Hate(B,B)|!Hate(B,C)|Richer(A,A)

!Hate(B,A)|!Hate(B,C)|Richer(B,A)

!Hate(B,A)|!Hate(B,B)|Richer(C,A)

!Kill(x7,A)|Hate(B,x7)

!Hate(A,x6)|!Richer(x6,A)

Hate(B,x6)|!Hate(B,x6)

!Hate(A,A)|!Hate(A,C)|!Hate(B,B)

!Hate(B,B)|!Hate(B,C)|!Hate(A,A)

!Hate(B,A)|!Hate(B,C)|!Hate(A,B)

!Hate(B,A)|!Hate(B,B)|!Hate(A,C)

!Hate(A,A)|Kill(B,A)|Kill(A,A)

!Richer(A,A)|Kill(C,A)|Kill(A,A)

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

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

!Richer(B,A)|Kill(A,A)|Hate(C,A)

!Richer(A,A)|Kill(B,A)|Hate(C,A)

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

!Kill(C,A)

!Hate(B,C)|Richer(B,A)|!Kill(B,A)

!Hate(B,B)|Richer(C,A)|!Kill(B,A)

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

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

Richer(x5,A)|!Hate(C,x5)

!Hate(A,B)

!Richer(A,A)

!Hate(A,C)|!Hate(B,B)

!Hate(B,B)|!Hate(B,C)

!Richer(C,A)

!Hate(A,A)|!Hate(B,B)

!Hate(B,A)|!Hate(B,B)

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

!Hate(B,C)|Richer(A,A)|Richer(B,A)

!Hate(B,B)|Richer(A,A)|Richer(C,A)

!Hate(B,A)|Richer(B,A)|Richer(C,A)

Hate(B,x6)|Richer(x6,A)

!Hate(A,A)|!Hate(A,C)|Richer(B,A)

!Hate(B,C)|!Hate(A,A)|Richer(B,A)

!Hate(B,A)|!Hate(A,C)|Richer(B,A)

!Hate(B,x6)|!Richer(x6,A)

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

!Kill(x7,A)|Hate(A,x7)

!Hate(B,x6)|Hate(A,x6)

!Hate(C,x5)|!Hate(A,x5)

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

Richer(x5,A)|Hate(B,x5)

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

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

Hate(A,x5)|!Kill(x5,A)

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

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

!Hate(B,C)|Richer(B,A)|Kill(C,A)|Kill(A,A)

!Hate(B,B)|Richer(C,A)|Kill(C,A)|Kill(A,A)

!Hate(A,A)|!Richer(B,A)|Kill(A,A)

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

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

!Hate(A,A)|Kill(B,A)

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

!Hate(B,B)|!Kill(B,A)

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

!Hate(B,x4)|!Hate(C,x4)

!Kill(x7,A)|!Hate(C,x7)

!Hate(B,x6)|!Hate(C,x6)

!Hate(B,C)|Richer(B,A)

!Hate(B,B)|Richer(C,A)

Richer(B,A)|!Hate(A,C)

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

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

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

!Richer(x6,A)|Richer(x6,A)

!Kill(B,A)|!Hate(B,A)|Richer(C,A)

Hate(B,x6)|!Kill(x6,A)

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

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

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

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

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

!Hate(B,A)|Richer(C,A)|Kill(C,A)|Kill(A,A)

!Hate(A,A)|!Hate(A,C)|Kill(C,A)|Kill(A,A)

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

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

!Richer(B,A)|Hate(C,A)

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

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

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

Richer(B,A)|!Kill(B,A)

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

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

!Kill(x5,A)|!Hate(C,x5)

Hate(B,B)|Kill(A,A)|!Hate(A,A)

!Hate(B,B)

!Richer(B,A)|Kill(A,A)

Kill(B,A)

Richer(B,A)

!Hate(A,A)|!Kill(B,A)

!Hate(B,A)|!Kill(B,A)

Hate(A,x4)|Richer(x4,A)

Hate(A,x6)|Richer(x6,A)

!Hate(C,x4)|Richer(x4,A)

!Hate(C,x6)|Richer(x6,A)

!Hate(A,x4)|!Richer(x4,A)

Hate(B,x5)|!Hate(B,x5)

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

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

!Kill(x6,A)|Hate(A,x6)

!Kill(x5,A)|Hate(B,x5)

!Hate(C,x4)|!Hate(A,x4)

!Hate(C,x6)|!Hate(A,x6)

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

Hate(B,x4)|!Kill(x4,A)

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

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

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

!Hate(B,A)|Richer(C,A)|Hate(C,A)|Kill(A,A)

!Hate(A,A)|!Hate(A,C)|Hate(C,A)|Kill(A,A)

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

!Hate(A,C)|Kill(C,A)|Kill(A,A)

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

!Hate(A,A)|Kill(C,A)|Kill(A,A)

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

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

!Hate(B,A)|Richer(C,A)|Kill(C,A)

!Hate(A,A)|!Hate(A,C)|Kill(C,A)

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

Hate(A,B)|Kill(A,A)|!Hate(A,A)

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

Hate(B,B)|!Hate(A,A)

Hate(B,B)|Kill(A,A)

!Kill(x6,A)|!Hate(C,x6)

!Kill(B,A)

Hate(A,x4)|!Kill(x4,A)

!Hate(C,x4)|!Kill(x4,A)

Hate(B,x5)|!Hate(A,x5)

!Kill(x5,A)|!Richer(x5,A)

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

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

!Hate(B,A)|Richer(C,A)|!Hate(A,A)|Kill(A,A)

!Hate(A,A)|!Hate(A,C)|Kill(A,A)

Richer(A,A)|Richer(C,A)|Kill(C,A)|Kill(A,A)

Richer(C,A)|Kill(C,A)|Kill(A,A)

Richer(A,A)|Kill(C,A)|Kill(A,A)

!Hate(A,C)|Kill(A,A)|Hate(C,A)

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

!Hate(A,A)|Kill(A,A)|Hate(C,A)

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

!Hate(B,A)|!Hate(B,C)|Hate(C,A)

!Hate(B,C)|Hate(C,A)

!Hate(C,B)|Hate(C,A)

!Hate(B,A)|Richer(C,A)|Hate(C,A)

!Hate(A,A)|!Hate(A,C)|Hate(C,A)

!Hate(B,A)|!Hate(A,C)|Hate(C,A)

!Richer(B,A)

Hate(A,B)|Kill(A,A)

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

Hate(B,B)

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

Kill(C,A)|!Hate(A,C)

Kill(C,A)|!Hate(A,A)

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

Hate(C,A)|Kill(A,A)|Richer(A,A)|Richer(C,A)

Kill(C,A)|Richer(A,A)|Richer(C,A)

Richer(x5,A)|!Richer(x5,A)

!Hate(A,x4)|Hate(A,x4)

!Hate(A,A)|Hate(A,B)

!Hate(A,A)|!Hate(B,A)|!Hate(B,C)

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

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

!Hate(B,A)|Richer(C,A)|Kill(A,A)

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

Richer(C,A)|Hate(C,A)|Kill(A,A)

Richer(A,A)|Hate(C,A)|Kill(A,A)

!Hate(B,A)|!Hate(A,A)|Kill(A,A)

!Hate(A,A)|Kill(A,A)

!Hate(B,C)|!Hate(A,A)

!Hate(C,B)|!Hate(A,A)

!Hate(B,A)|Richer(C,A)|!Hate(A,A)

!Hate(A,A)|!Hate(A,C)

Richer(C,A)|Kill(C,A)

Richer(A,A)|Kill(C,A)

Kill(A,A)|Hate(C,A)

!Hate(A,C)|Hate(C,A)

!Hate(A,A)|Hate(C,A)

!Hate(B,A)|Hate(C,A)

Richer(A,A)|Richer(C,A)|Hate(C,A)

Richer(C,A)|Hate(C,A)

Richer(A,A)|Hate(C,A)

!Hate(A,C)|Kill(A,A)

Kill(C,A)

Hate(A,B)

!Hate(B,A)|!Hate(B,C)

!Hate(B,A)|Kill(A,A)

Kill(A,A)

!Hate(B,C)

!Hate(C,B)

!Hate(B,A)|Richer(C,A)

!Hate(A,C)

!Hate(A,A)

!Hate(B,x5)|!Richer(x5,A)

!Kill(x4,A)|!Richer(x4,A)

!Hate(A,x5)|!Richer(x5,A)

Hate(C,A)

!Richer(x4,A)|Richer(x4,A)

!Hate(B,A)

Richer(A,A)|Richer(C,A)

Richer(C,A)

Richer(A,A)

!Hate(A,x5)|Hate(A,x5)

None

### 3.3.2 实验结论

所有实验结果展示如上，本次实验结论根据人为推理，与子句归结的结果相同，证实了该程序的正确性。

# 总结

## 4.1 实验中存在的问题及解决方案

问题1：

实验测试过程中发现对于任意给定的正确的输入，无法归结出所需求的空语句。

解决方案：

Resolve函数书写存在问题，归结出的空子句无法正确返回，而是返回直接跳跃返回BOOL值False,对空子句的逻辑单独处理后，可以使得空子句作为空列表返回。

问题2：

归结过程的不完备，存在正确输入无法归结出的子句。

解决方案：

最简合一替换的方案具有多重性，未考虑到其它替换方案导致生成不了所需子句，将子句中所含谓词一一对应比照后可生成所有合一替换方案。

问题3：

知识库中归结产生大量子句，导致归结运行时间大大延长。

解决方案：

归结过程中产生的新的子句可能未包含新信息，将其加入知识库中对于问题求解并无帮助，删除未归结出新信息的结果子句，可减少归结算法运行时间。

## 4.2 心得体会

从此次实验中，我们得到了以下心得和体会：此次实验主要是掌握Horn子句的归结原理应用。对于给定的一阶逻辑命题，我们先选择通过消除蕴含和等价、否定内移、变量标准化、消去存在量词、将公式化为前束型、化为合取范式、略去全称量词、消去合取符号、子句变量标准化这9个通用步骤，即可将知识库中的一阶逻辑命题转化为子句。本题中将字句作为知识库输入，将目标子句取反加入知识库中。如果能成功归结出空子句，则可证明目标子句成立，同时由于归结算法的完备性，若不能归结出空子句，则可证明目标子句无法成立。只有两个存在互补文字的子句才能归结出含有新信息的子句。同时合一作为一阶逻辑推理算法的关键，在归结算法中具有重要意义，避免直接将一阶逻辑命题转化为命题逻辑（对于对象较多的情况，这种转换将使得知识库中的内容大大增加，降低了归结算法的效率）。

## 4.3 后续改进方向

对于后续改进方向，主要可以有以下两点：

第一点是对于归结函数的改进，其一，在本次实验的归结函数中，对于任何两个子句的归结是全面的，但还是有重复的部分产生，所以后续改进中可以减少归结果过程的重复性。其二，可以改变一下归结的顺序，使可能得出矛盾的解更快被得出，可以设计优先队列，先对原子句少的子句对进行归结，这样能提高效率。

第二点是对于适配性的改进，因为本实验要求解决该问题，所以对于输入的要求还是比较局限的，比如常量为‘A’,‘B’,‘C’，变量为x。后续改进中可以引入更多的类型或字符来使归结输入更具有普适性。

## 4.4 总结

（1）该实验经过小组的合作、编写，最终基本完成要求，也对结果进行了测试和分析，找出了杀害A的凶手。

（2）在实验的过程中，我们回顾了合一置换、一阶逻辑命题、归结原理等知识，对相关知识的理解更为深入了。

（3）通过项目初步实现了基于知识的逻辑Agent。

# 参考文献

1. Russell, S.J. 人工智能：一种现代的方法[m](第三版). 清华大学出版社, 2013

# 成员分工与自评

**分工：**

谢安南：

主要负责合一置换等内核算法的编写与优化函数的编写，实验结果的测试与分析，部分实验报告的撰写。

杨骏恒：

主要负责网页UI界面的设计与编写，归结算法的编写，ppt的制作。

徐铭骋：

主要负责子句转换函数的编写，部分归结算法内核的编写，实验结果的测试与分析，实验报告的撰写。

**自评：**

我们认为我们小组基本上完成了本次实验所要求的全部内容，较好地实现了字句归结的算法程序设计，并能保证其正确性与效率，在UI界面，我们使用了网页来进行设计，对网页的使用更有了进一步熟练的掌握。总的来说算是较好地完成了本次大作业。

# 附 录

附录一：manage.py

#导入flask

from flask import Flask,render\_template,request,escape,url\_for,jsonify

import cmath

import tkinter as tk

import time

import copy

import re

#用于将内部存储转换为子句输出

def convert\_lang(clause\_in):

ret = ''

for i in clause\_in:

if(clause\_in.index(i)):

ret += '|'

if(i[1] == False):

ret += '!'

ret += i[0] + '(' + i[2] + ',' + i[3] + ')'

if len(ret)==0:

ret="None"

return ret

#用于将输入子句转换为内部存储结构

def convert(str\_in ):

pattern = re.compile(r'([a-z]+\d+|[a-z]+)', re.I)

pattern1 = re.compile(r'(\|)')

pattern2 = re.compile(r'(![a-z]+\(|[a-z]+\()', re.I)

m = pattern.findall(str\_in)

m1 = pattern1.findall(str\_in)

m2 = pattern2.findall(str\_in)

ret = []

for i in range (len(m1) + 1):

tmp = []

for j in range (4):

if(j==0 and m2[i][0]=='!'):

tmp.append(False)

elif(j==0):

tmp.append(True)

else:

tmp.append(m[3\*i+j-1])

ret.append(tmp)

for i in ret:

temp=i[0]

i[0]=i[1]

i[1]=temp

return ret

#合一函数，用于判断一阶逻辑子句是否能合一并输出所有的归结合一结果

def unify(clause\_a,clause\_b):

subst=[]

for a in clause\_a:

for b in clause\_b:

tmp=[]

if(a[0]!=b[0]or a[1]==b[1]):

continue

elif(a[2]==b[2] and a[3]==b[3]):

continue

elif(a[2]!=b[2] and a[2][0]!='x' and b[2][0]!='x'):

continue

elif(a[3]!=b[3] and a[3][0]!='x' and b[3][0]!='x'):

continue

if(a[2]!=b[2]):

if(a[2][0]=='x'):

tmp.append(a[2])

tmp.append(b[2])

else:

tmp.append(b[2])

tmp.append(a[2])

if(a[3]!=b[3]):

if(a[3][0]=='x'):

tmp.append(a[3])

tmp.append(b[3])

else:

tmp.append(b[3])

tmp.append(a[3])

subst.append(tmp)

clause\_c=copy.deepcopy(clause\_a)

clause\_d=copy.deepcopy(clause\_b)

ret=[]

tmp=[]

tmp.append(clause\_c)

tmp.append(clause\_d)

ret.append(tmp)

for t in subst:

tmp=[]

clause\_c=copy.deepcopy(clause\_a)

clause\_d=copy.deepcopy(clause\_b)

for a in clause\_c:

if(len(t)>0):

if a[2]==t[0]:

a[2]=t[1]

if a[3]==t[0]:

a[3]=t[1]

if(len(t)>2):

if a[2]==t[2]:

a[2]=t[3]

if a[3]==t[2]:

a[3]=t[3]

for a in clause\_d:

if(len(t)>0):

if a[2]==t[0]:

a[2]=t[1]

if a[3]==t[0]:

a[3]=t[1]

if(len(t)>2):

if a[2]==t[2]:

a[2]=t[3]

if a[3]==t[2]:

a[3]=t[3]

tmp.append(clause\_c)

tmp.append(clause\_d)

ret.append(tmp)

return ret

#归结函数主体实现，用于归结子句得出矛盾或者无法继续归结

def resolve(a,b):

c=[]

a\_length=len(a)

b\_length=len(b)

for i in range(0,a\_length):

for j in range(0,b\_length):

d=[]

if(a[i][0]==b[j][0] and a[i][1]!=b[j][1] and a[i][2]==b[j][2] and a[i][3]==b[j][3]):

if(a\_length==1 and b\_length>1):

for k in range(0,b\_length):

if(k!=j):

d.append(b[k])

return d

elif(a\_length>1 and b\_length==1):

for k in range(0,a\_length):

if(k!=i):

d.append(a[k])

return d

elif(a\_length>1 and b\_length>1):

a1=copy.deepcopy(a)

b1=copy.deepcopy(b)

del a1[i]

del b1[j]

for y in range(0,len(b1)):

d.append(b1[y])

for x in range(0,len(a1)):

T=0

for y in range(0,len(b1)):

if(a1[x][0]==b1[y][0] and a1[x][1]==b1[y][1] and a1[x][2]==b1[y][2] and a1[x][3]==b1[y][3]):

T=1

if(T==0):

d.append(a1[x])

return d

elif(a\_length==1 and b\_length==1):

return d

return False

#用于测试子句中是否含有变量

def test\_variable(a,b):

for t in a:

if(t[2][0]=='x' or t[3][0]=='x'):

return True

for t in b:

if(t[2][0]=='x' or t[3][0]=='x'):

return True

return False

#用于检测生成子句是否与已有重复

def exam(Horn,new):

b=set()

for tmp1 in new:

b.add(tuple(tmp1))

for a in Horn:

c=set()

for tmp1 in a.clause:

c.add(tuple(tmp1))

if c.issubset(b) :

return False

return True

#节点类，将归结过程显示更加有逻辑性

class Node:

def \_\_init\_\_(self, clause, father,mother):

self.clause = clause

self.father = father

self.mother = mother

#创建一个app应用

#\_\_name\_\_指向程序所在的包

app=Flask(\_\_name\_\_)

#生成前端主页面

@app.route('/',methods=['GET','POST'])

def index1():

return render\_template("index.html")

#获取前端用户输入的子句集和归结目标

@app.route('/2',methods=['GET','POST'])

def index2():

if request.method=='POST':

#获取前端用户输入的子句集

textarea1=request.form.get('textarea')

text=textarea1.splitlines()

#获取前端用户输入的归结目标

input1=request.form.get('input')

Horn=[]

for i in range(0,len(text)-1):

t=Node(convert(text[i]),0,0)

Horn.append(t)

target=convert(input1)

target[0][1]=not target[0][1]

t=Node(target,0,0)

Horn.append(t)

while True :

t=len(Horn)

NULL=False

for tb\_i in range(0,t):

for tb\_j in range(tb\_i+1,t):

new=unify(Horn[tb\_i].clause,Horn[tb\_j].clause)

for tt in new:

tmp=resolve(tt[0],tt[1])

if tmp==False:

continue

if exam(Horn,tmp):

Horn.append(Node(tmp,tb\_i+1,tb\_j+1))

else:

continue

if len(tmp)==0:

NULL=True

break

if NULL:

break

if NULL:

break

if NULL:

result=""

for i in Horn:

result=result+convert\_lang(i.clause)+"\n"

print(Horn.index(i)+1," ",convert\_lang(i.clause),i.father,i.mother)

print("归结成功")

answer="归结成功"

break

if(t==len(Horn)):

result=""

for i in Horn:

result=result+convert\_lang(i.clause)+"\n"

print(Horn.index(i)+1," ",convert\_lang(i.clause),i.father,i.mother)

print("归结失败")

answer="归结失败"

break

return jsonify({'input':input1,'textarea':textarea1,'result':result,'answer':answer})

if \_\_name\_\_=='\_\_main\_\_':

#web服务器的入口

app.run(debug=True,port=5001)

附录二：index.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<script src="{{url\_for('static',filename='jquery.js')}}"></script>

<style>

body {

background-color: rgba(187, 183, 160, 0.473);

}

#top\_div{

position: absolute;

top: 4%;

left: 24%;

overflow: hidden;

width: 52%;

height: 12%;

background: rgba(189, 233, 247, 0.425);

border-radius: 5px;

-webkit-transform: translate3d(-100%, 0, 0);

transform: translate3d(-0%, 0, 0);

box-shadow: 0 10px 35px rgba(0, 0, 0, 0.2);

}

#div0 {

float: left;

position: absolute;

top: 20%;

left: 10%;

overflow: hidden;

width: 28%;

height: 42%;

background: rgba(189, 233, 247, 0.425);

border-radius: 5px;

-webkit-transform: translate3d(-100%, 0, 0);

transform: translate3d(-0%, 0, 0);

box-shadow: 0 10px 35px rgba(0, 0, 0, 0.2);

}

#div2 {

float: left;

position: absolute;

top: 66%;

left: 10%;

overflow: hidden;

width: 28%;

height: 26%;

background: rgba(189, 233, 247, 0.425);

border-radius: 5px;

-webkit-transform: translate3d(-100%, 0, 0);

transform: translate3d(-0%, 0, 0);

box-shadow: 0 10px 35px rgba(0, 0, 0, 0.2);

}

#div3 {

float: right;

position: absolute;

top: 20%;

right: 10%;

overflow: hidden;

width: 46%;

height: 12%;

background: rgba(189, 234, 247, 0);

border-radius: 5px;

-webkit-transform: translate3d(-100%, 0, 0);

transform: translate3d(-0%, 0, 0);

box-shadow: 0 10px 35px rgba(0, 0, 0, 0);

}

#div1 {

float: right;

position: absolute;

top: 36%;

right: 10%;

overflow: hidden;

width: 46%;

height: 56%;

background: rgba(189, 233, 247, 0.425);

border-radius: 5px;

-webkit-transform: translate3d(-100%, 0, 0);

transform: translate3d(-0%, 0, 0);

box-shadow: 0 10px 35px rgba(0, 0, 0, 0.2);

}

.button {

display: inline-block;

padding: 12px 30px;

font-size: 24px;

cursor: pointer;

text-align: center;

text-decoration: none;

outline: none;

color: #fff;

background-color: #04263d;

border: none;

border-radius: 15px;

box-shadow: 0 9px #999;

}

.button:hover {

background-color: #054977

}

.button:active {

background-color: #054977;

box-shadow: 0 5px #666;

transform: translateY(4px);

}

</style>

</head>

<body>

<div id="top\_div">

<h1 style="text-align:center;">子句归结实验</h1>

</div>

<div id="div0">

<div style="width: 100%; height: 24%;">

<h1 style="text-align:center;">子句集输入(#结尾)</h1>

</div>

<div style="width: 100%; height: 76%;">

<textarea style="width: 84%; height: 72%; margin-left: 8%;" id="textarea">Kill(C,A)|Kill(B,A)|Kill(A,A)&#10;Hate(x1,A)|!Kill(x1,A)&#10;!Hate(C,x2)|!Hate(A,x2)&#10;Hate(A,A)&#10;Hate(A,C)&#10;Richer(x3,A)|Hate(B,x3)&#10;!Richer(x4,A)|!Hate(B,x4)&#10;Hate(A,x5)|!Hate(B,x5)&#10;!Hate(A,x6)|Hate(B,x6)&#10;!Hate(A,A)|!Hate(A,B)|!Hate(A,C)&#10;!Hate(B,A)|!Hate(B,B)|!Hate(B,C)&#10;!Hate(C,A)|!Hate(C,B)|!Hate(C,C)&#10;!Richer(x7,A)|!Kill(x7,A)&#10;#</textarea>

</div>

</div>

<div id="div2">

<div style="width: 100%; height: 36%;">

<h1 style="text-align:center;">目标输入</h1>

</div>

<div style="width: 100%; height: 64%;">

<input style="width: 84%; margin-left: 8%; height: 24%;" type="text" name="input" id="input" value="Kill(A,A)">

</div>

</div>

<div id="div3">

<div style="margin-left: 12%; width: 42%; margin-top: 16px;">

<button onclick="get\_result()" class="button">开始归结</button>

<button onclick="del()" class="button" style="float: right;">清除</button>

</div>

</div>

<div id="div1">

<div style="width: 100%;">

<h1 style="text-align:center;" id="answer">点击上方按钮查看归结结果</h1>

</div>

<textarea style="width: 84%; height: 70%; margin-left: 8%; margin-top: 2%;" id="p"></textarea>

</div>

<script>

var textarea=document.getElementById("textarea");

var input=document.getElementById("input");

var p=document.getElementById("p");

var answer1=document.getElementById("answer");

var aaa;

function get\_result(){

// console.log(textarea.value);

// console.log(input.value);

$.ajax({

url: "/2",

data: {

"input": input.value, "textarea": textarea.value

},

type: "post",

success: function (data) {

console.log(data);

aaa=data["result"];

p.value=aaa;

answer1.innerHTML=data["answer"];

}

})

}

function del(){

textarea.value="";

input.value="";

answer1.innerHTML="点击上方按钮查看归结结果";

p.value="";

}

</script>

</body>

</html>