装 订 线

子句归结实验报告

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

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。

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. 实验方案设计

2.1 总体设计思路与总体架构

实验首先对破案问题进行分析，将上述破案问题化为子句，然后根据同可满足性化为命题逻辑，方便输入和进行处理。

在程序设计中，先输入已知事实、过程（限定蕴含式）、以及目标（要证明的结论）。

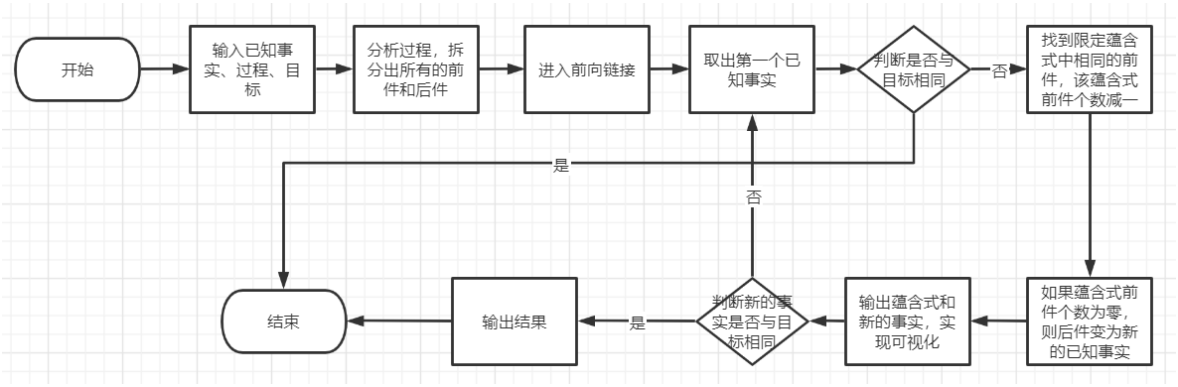
然后对过程进行分析，拆分出每个蕴含式的前件和后件，并对所有的限定蕴含式根据前件

个数进行排序。

然后利用前向链接，根据已知事实，对限定蕴含式进行推导，得到要证明的结果。同时实现证明过程的可视化。

由上述思路建立总体架构，共四大模块，分别为初始化、命题分析、命题推导、可视化。

流程图如下：



2.2 核心算法及基本原理

核心算法为前向链接。

前向链接算法 PL-FC- ENTAILS?(KB, q)判定单个命题词 q——查询——是否被限定子句的知识库所蕴涵。它从知识库中的已知事实(正文字)出发。如果蕴含式的所有前提已知，那么就把它的结论添加到已知事实集。例如，若 $L_{1,1}$和 Breeze 已知，而且$(L_{1,1} \wedge Breeze) \rightarrow B_{1,1}$在知识库中，那么 $B_{1,1}$被添加到知识库中。这个过程持续进行，直到查询 q 被添加或者无法进行更进一步的推理。详细的算法见下图，它的运行时间是线性的。

```
function PL-FC-ENTAILS?(KB, q) returns true or false
  inputs: KB, the knowledge base, a set of propositional definite clauses
          q, the query, a proposition symbol
  count ← a table, where count[c] is the number of symbols in c's premise
  inferred ← a table, where inferred[s] is initially false for all symbols
  agenda ← a queue of symbols, initially symbols known to be true in KB

  while agenda is not empty do
      p ← POP(agenda)
      if p = q then return true
      if inferred[p] = false then
          inferred[p] ← true
          for each clause c in KB where p is in c.PREMISE do
              decrement count[c]
              if count[c] = 0 then add c.CONCLUSION to agenda
  return false
```

2.3 模块设计

本实验共有四大模块分别为初始化、命题分析、命题推导、可视化。

初始化模块：输入已知事实、过程和目标。

命题分析模块：对过程进行分析，得到每一个限定蕴含式的前件、后件，并且对所有的限定蕴含式根据前件个数进行排序。

命题推导模块：根据已知事实，对限定蕴含式进行推导，得到目标。

可视化模块：在命题推导过程中输出证明步骤，实现可视化。

2.4 其他创新内容或优化算法

1）前件和值排序：对前向链接的简单实现，使用的是任意排序的蕴含式。而前件和值排序，根据每个蕴含式前件的多少排序，优先对前件少的蕴含式进行处理，以保证尽快出现新的事实，根据统计原则，可以更快的得到目标。

2）及早终止：对前向链接的简单实现，是在取出事实时，判断是否和目标相同。而及早终止是取出事实时的判断，加上得到新事实的判断，使得在刚出现目标时，就搜索停止，减少了搜索的时间和空间消耗。

3）map 快速匹配：利用 map<string, bool>，快速地对所有的前件进行匹配，同时避免了一个蕴含式出现多个相同前件，使得字符串处理更轻松，而且速度更快。

3.实验过程

3.1 环境说明

操作系统：Windows 10

开发语言：C++

开发环境：Visual Studio 2019

核心使用库：C++ STL 库

3.2 源代码文件清单，主要函数清单

代码清单：

clause_resolution.cpp

主要函数清单：

c 函数：程序入口，调用 message()、init()、analysis()、resolution()。

message 函数：给出说明信息。

init 函数：输入已知事实、过程和目标。

analysis 函数：对过程进行分析，得到每一个限定蕴含式的前件、后件，并且对所有的限定蕴含式根据前件个数进行排序。

resolution 函数：根据已知事实，对限定蕴含式进行推导，得到目标，同时在命题推导过程中输出证明步骤，实现可视化。

3.3 源代码

```
#include<iostream>
#include<map>
#include<queue>
#include<algorithm>
using namespace std;

//过程表达式
struct inferred
{
    string process;//过程
    int count = 0;//前提个数
    map<string, bool> hasKnown;//每个前提是否已知
    string conclusion;//结论
};
//用于每个过程排序
bool compare(inferred a, inferred b)
{
    return a.count < b.count;
}
inferred counts[50];//过程表达式总数
queue<string> agenda;//还未使用的事实
int processNum;//过程个数
string target;//目标

//说明信息
void message()
{
    cout << "请使用命令行运行该程序\n";
    cout << "本实验为字句归结\n";
    cout << "首先一步便是将破案问题化为子句\n";
    cout << "根据问题表述，以及同可满足性，化出子句如下：\n";
```

```
    cout << "5 条事实：\n";
    cout << "L(A) L(B) L(C) H(A,A) H(A,C)\n";
    cout << "33 条限定蕴含式：\n";
    cout << "H(A,A)->H(B,A)\n";
    cout << "H(A,B)->H(B,B)\n";
    cout << "H(A,C)->H(B,C)\n";
    cout << "H(A,A)->NH(C,A)\n";
    cout << "H(A,B)->NH(C,B)\n";
    cout << "H(A,C)->NH(C,C)\n";
    cout << "H(B,A)->NR(A,A)\n";
    cout << "H(B,B)->NR(B,A)\n";
    cout << "H(B,C)->NR(C,A)\n";
    cout << "NH(B,A)->R(A,A)\n";
    cout << "NH(B,B)->R(B,A)\n";
    cout << "NH(B,C)->R(C,A)\n";
    cout << "H(A,A)&H(A,B)->NH(A,C)\n";
    cout << "H(A,A)&H(A,C)->NH(A,B)\n";
    cout << "H(A,B)&H(A,C)->NH(A,A)\n";
    cout << "H(B,A)&H(B,B)->NH(B,C)\n";
    cout << "H(B,A)&H(B,C)->NH(B,B)\n";
    cout << "H(B,B)&H(B,C)->NH(B,A)\n";
    cout << "H(C,A)&H(C,B)->NH(C,C)\n";
    cout << "H(C,A)&H(C,C)->NH(C,B)\n";
    cout << "H(C,B)&H(C,C)->NH(C,A)\n";
    cout << "NL(A)->NK(A,A)\n";
    cout << "NH(A,A)->NK(A,A)\n";
    cout << "R(A,A)->NK(A,A)\n";
    cout << "NL(B)->NK(B,A)\n";
    cout << "NH(B,A)->NK(B,A)\n";
    cout << "R(B,A)->NK(B,A)\n";
    cout << "NL(C)->NK(C,A)\n";
    cout << "NH(C,A)->NK(C,A)\n";
    cout << "R(C,A)->NK(C,A)\n";
    cout << "L(A)&H(A,A)&NR(A,A)&NK(B,A)&NK(C,A)->K(A,A)\n";
    cout << "L(B)&H(B,A)&NR(B,A)&NK(A,A)&NK(C,A)->K(B,A)\n";
    cout << "L(C)&H(C,A)&NR(C,A)&NK(A,A)&NK(B,A)->K(C,A)\n";
    cout << "1 条结论：\n";
    cout << "K(A,A)\n";
    cout << "通过举例来说明这些子句\n";
    cout << "L(A)：A 住在这儿\n";
    cout << "H(A,C)：A 恨 C\n";
    cout << "R(C,A)：C 比 A 富有\n";
    cout << "K(A,A)：A 杀死了 A\n";
    cout << "NL(A)：A 不住在这儿，其他以 N 开头的同理\n\n";
}
```

```
//初始化
void init()
{
    //事实部分
    cout << "请输入事实个数\n";
    int factNum;//事实个数
    string fact;//事实
    cin >> factNum;
    cout << "请输入每个事实\n";
    for (int i = 0; i < factNum; i++)
    {
        cin >> fact;
        agenda.push(fact);//入队
    }

    //过程部分
    cout << "请输入过程(限定蕴含式)个数\n";
    cin >> processNum;
    cout << "请输入每个过程\n";
    for (int i = 0; i < processNum; i++)
        cin >> counts[i].process;

    //目标部分
    cout << "请输入目标\n";
    cin >> target;
}

//分析
void analysis()
{
    //对过程进行分析，得到前件个数，并将前件标为 false
    string s;
    for (int i = 0; i < processNum; i++)
    {
        //获取过程的字符串长度
        int processSize = counts[i].process.size();

        //对字符串进行划分
        for (int j = 0; j < processSize; j++)
        {
            //划分前提
            if (counts[i].process[j] == '&')
            {
                counts[i].count++;
                counts[i].hasKnown[s] = false;
                s = "";
```

装 订 线

```
            }

            //前提之后是后件
            else if (counts[i].process[j] == '-')
            {
                counts[i].count++;
                counts[i].hasKnown[s] = false;
                s = "";
                counts[i].conclusion = counts[i].process.substr(j + 2);
                break;
            }

            //延长字符串
            else
            {
                s += counts[i].process[j];
            }
        }
    }

    //根据前件个数排序
    sort(counts, counts + processNum,compare);
}

//归结
void resolution()
{
    //输出事实
    cout << "\n 根据事实： ";
    int agendaSize = agenda.size();
    for (int i = 0; i < agendaSize; i++)
    {
        string fact = agenda.front();
        cout << fact<<" ";
        agenda.pop();
        agenda.push(fact);
    }
    cout << endl;

    //进行归结
    while (!agenda.empty())
    {
        //取出一个事实
        string known = agenda.front();
        agenda.pop();
```

```
        //事实等于目标
        if (known == target)
        {
            cout << "得到结果：" << target << endl;
            return;
        }

        //找和事实相同的前件
        else
        {
            for (int i = 0; i < processNum; i++)
            {
                //当这条蕴含式还有前件
                if (counts[i].count != 0)
                {
                    //找到和事实相同的前件
                    if (counts[i].hasKnown.find(known) != counts[i].hasKnown
.end())
                    {
                        counts[i].hasKnown[known] = true;
                        counts[i].count--;
                    }

                    //这条表达式没有前件了
                    if (counts[i].count == 0)
                    {
                        agenda.push(counts[i].conclusion);//后件转为事实
                        cout << "结合：" << counts[i].process << endl;
                        cout << "得出新事实：
" << counts[i].conclusion << endl << endl;

                        //刚得到的事实等于目标
                        if (counts[i].conclusion == target)
                        {
                            cout << "得到结果：" << target << endl;
                            return;
                        }
                    }

                }
            }
        }
    }
}
int main()
{
```

```
    message();//给出说明信息
    init();//初始化
    analysis();//命题分析
    resolution();//归结
    return 0;
}
```

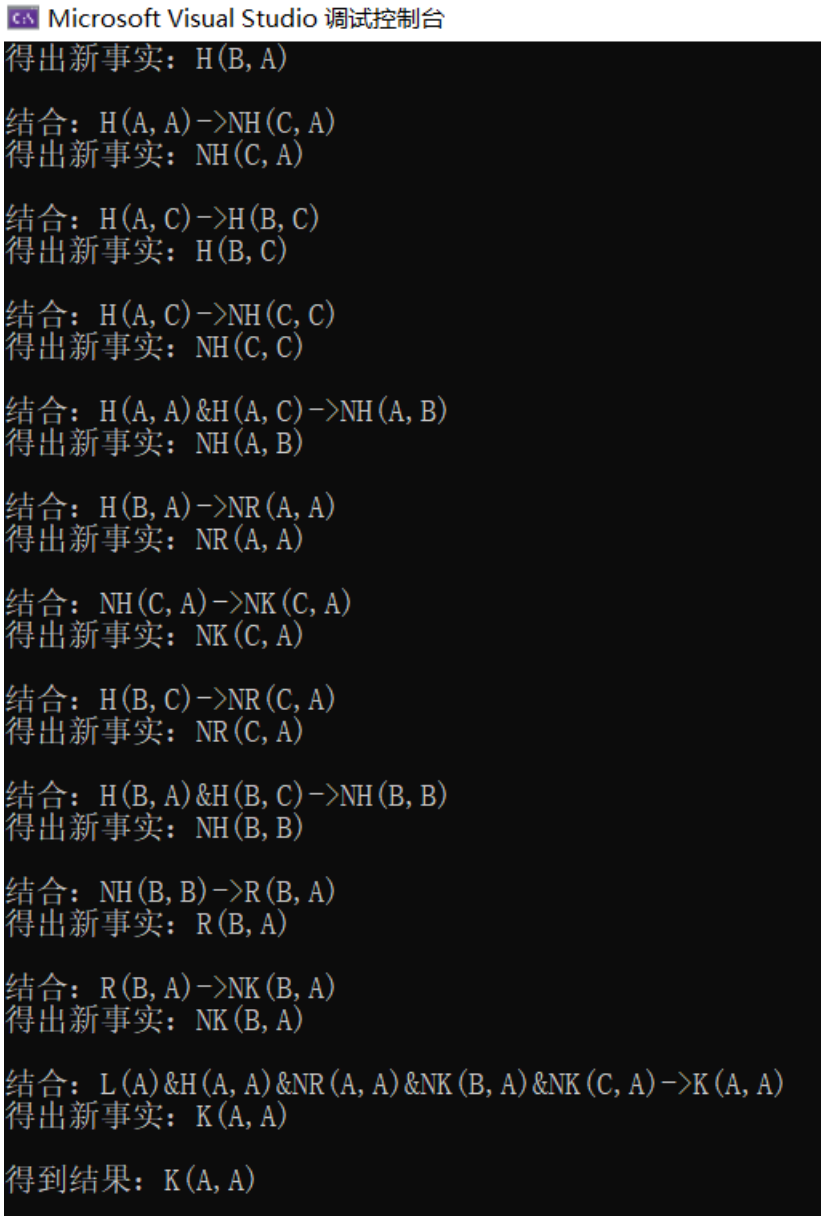
3.4 实验结果展示

Microsoft Visual Studio 调试控制台

```
本实验为字句归结
首先一步便是将破案问题化为子句
根据问题表述，以及同可满足性，化出子句如下：
5条事实：
L(A) L(B) L(C) H(A,A) H(A,C)
33条限定蕴含式：
H(A,A)->H(B,A)
H(A,B)->H(B,B)
H(A,C)->H(B,C)
H(A,A)->NH(C,A)
H(A,B)->NH(C,B)
H(A,C)->NH(C,C)
H(B,A)->NR(A,A)
H(B,B)->NR(B,A)
H(B,C)->NR(C,A)
NH(B,A)->R(A,A)
NH(B,B)->R(B,A)
NH(B,C)->R(C,A)
H(A,A)&H(A,B)->NH(A,C)
H(A,A)&H(A,C)->NH(A,B)
H(A,B)&H(A,C)->NH(A,A)
H(B,A)&H(B,B)->NH(B,C)
H(B,A)&H(B,C)->NH(B,B)
H(B,B)&H(B,C)->NH(B,A)
H(C,A)&H(C,B)->NH(C,C)
H(C,A)&H(C,C)->NH(C,B)
H(C,B)&H(C,C)->NH(C,A)
NL(A)->NK(A,A)
NH(A,A)->NK(A,A)
R(A,A)->NK(A,A)
NL(B)->NK(B,A)
NH(B,A)->NK(B,A)
R(B,A)->NK(B,A)
NL(C)->NK(C,A)
NH(C,A)->NK(C,A)
R(C,A)->NK(C,A)
L(A)&H(A,A)&NR(A,A)&NK(B,A)&NK(C,A)->K(A,A)
L(B)&H(B,A)&NR(B,A)&NK(A,A)&NK(C,A)->K(B,A)
L(C)&H(C,A)&NR(C,A)&NK(A,A)&NK(B,A)->K(C,A)
1条结论：
K(A,A)
通过举例来说明这些子句
L(A)：A住在这儿
H(A,C)：A恨C
R(C,A)；C比A富有
K(A,A)：A杀死了A
NL(A)：A不住在这儿，其他以N开头的同理

请输入事实个数
5
```

Microsoft Visual Studio 调试控制台

```
请输入事实个数
5
请输入每个事实
L(A) L(B) L(C) H(A,A) H(A,C)
请输入过程(限定蕴含式)个数
33
请输入每个过程
H(A,A)->H(B,A)
H(A,B)->H(B,B)
H(A,C)->H(B,C)
H(A,A)->NH(C,A)
H(A,B)->NH(C,B)
H(A,C)->NH(C,C)
H(B,A)->NR(A,A)
H(B,B)->NR(B,A)
H(B,C)->NR(C,A)
NH(B,A)->R(A,A)
NH(B,B)->R(B,A)
NH(B,C)->R(C,A)
H(A,A)&H(A,B)->NH(A,C)
H(A,A)&H(A,C)->NH(A,B)
H(A,B)&H(A,C)->NH(A,A)
H(B,A)&H(B,B)->NH(B,C)
H(B,A)&H(B,C)->NH(B,B)
H(B,B)&H(B,C)->NH(B,A)
H(C,A)&H(C,B)->NH(C,C)
H(C,A)&H(C,C)->NH(C,B)
H(C,B)&H(C,C)->NH(C,A)
NL(A)->NK(A,A)
NH(A,A)->NK(A,A)
R(A,A)->NK(A,A)
NL(B)->NK(B,A)
NH(B,A)->NK(B,A)
R(B,A)->NK(B,A)
NL(C)->NK(C,A)
NH(C,A)->NK(C,A)
R(C,A)->NK(C,A)
L(A)&H(A,A)&NR(A,A)&NK(B,A)&NK(C,A)->K(A,A)
L(B)&H(B,A)&NR(B,A)&NK(A,A)&NK(C,A)->K(B,A)
L(C)&H(C,A)&NR(C,A)&NK(A,A)&NK(B,A)->K(C,A)
请输入目标
K(A,A)

根据事实：L(A) L(B) L(C) H(A,A) H(A,C)
结合：H(A,A)->H(B,A)
得出新事实：H(B,A)

结合：H(A,A)->NH(C,A)
得出新事实：NH(C,A)
```



3.5 实验结论

子句归结实验，在使用前向链接时，可以很容易看出前向链接是可靠的：每个推理本质上都是假言推理规则的应用。前向链接也是完备的：每个被蕴涵的原子语句都可以推导得出。验证这一点的最简单方法是考察 inferred 表的最终状态(在算法到达不动点以后,不会再出现新的理)。

本次实验成功实现了所给问题的逻辑表示，子句的转换，并且在实现了 Horn 子句的归结求解。同时界面可以显示每一步的求解过程。针对破案问题，可以对其进行归结，并得到最终目

标。但策略选择、算法设计、技巧添加、可视化方面仍有较大改进空间。

4.总结

4.1 实验中存在的问题及解决方案

1）在前件中出现后件内容。经过 DEBUG，发现是在判定后件，并获取后件后，没有及时跳出循环。及时 break 掉循环即可。

2）判断已知事实是否和前件相同，以及是否出现一个蕴含式包含相同的前件较为麻烦。通过使用 map<string, bool>，可以方便判断。

4.2 心得体会

回顾这次实验，感觉十分的充实，通过认真学习、亲自动手，使我进一步了解了 horn 字句归结、前向链接算法等的基本知识和设计方法，为今后的学习奠定了良好的基础。

理论课让我们系统地学习了逻辑推理的相关理论。而实验课给了我们一个很好的把理论应用到实践的平台，让我们能够很好的把书本知识转化到实际能力，提高了对于理论知识的理解，认识和掌握。

其次，实验很好地解决了我们实践能力不足且得不到很好锻炼机会的矛盾，通过实验，提高了自身的实践能力和思考能力，并且能够通过实验很好解决自己对于理论的学习中存在的一些知识盲点。

以此次实验为例，不仅完成了实验设计，充分理解了 horn 字句归结、前向链接算法，而且研究了斯柯伦标准型、同可满足性、各类归结策略、前件和值排序、map 快速匹配等，使我更好地学习了 horn 字句归结的设计，扩展了自己的视野，提高了自己的实践能力。

4.3 后续改进方向

1）研究并加入 David-Putnam 算法，重点掌握成分分析、智能回溯、随机重新开始等。

2）研究并加入局部搜索算法，平衡贪婪性和随机性的问题。

3）转为一阶逻辑的推理证明。利用前向链接和反向链接进行推理，并对比两者。同时解决冗余推理、无限循环等问题。

4）继续研究归结策略，用于其他问题解决，例如规划生成、程序综合、程序分析、程序验证等。

4.4 总结

Horn 子句归结，涉及到逻辑推理的各方面知识，可以利用归结策略，使用前向链接、反向链接等各种算法解决问题。在一般算法的基础上，加入和值排序、及早终止等技巧等将有效减小问题规模，加快算法执行。另外在程序编写过程中，要充分利用 STL 库，例如 map、queue、sort 等，可以优化程序，同时提高性能。

本次实验按照预期完成，成功设计了字句归结方法，较为高效地解决了破案问题。

同时对程序而言，各类回溯算法、局部搜索算法、各类归结策略，如锁归结策略、支持集策略等仍具有研究价值和改进空间。另外也可以将归结原理用于其他的应用，如规划生成、程序综合、程序分析、程序验证等。

5.参考文献

[1]Stuart Russell, Peter Norvig.人工智能:一种现代的方法[M].清华大学出版社:北京,2013:82-95.