E2_22336216

人工智能实验

中山大学计算机学院

人工智能

本科生实验报告

（2023学年春季学期）

课程名称：Artificial Intelligence

教学班级	DCS315	专业（方向）	计算机科学与技术（系统结构）
学号	22336216	姓名	陶宇卓

1 实验题目

实验二：归结推理

2 实验内容

一、算法原理

（一）储存数据的数据结构

在本人代码中，我采用三重列表KB进行命题的存储，KB[i][j]代表第i个子句的第j个原子公式。这个公式也以列表形式存储，KB[i][j][n]可以用来代表谓词，常量，变量等。下面是一个例子：

假定输入为alpineclub.txt：

alpineclub.txt Plain Text

```
11
A(tony)
A(mike)
A(john)
L(tony,rain)
L(tony,snow)
(¬A(x),S(x),C(x))
(¬C(y), ¬L(y,rain))
(L(z,snow), ¬S(z))
(¬L(tony,u), ¬L(mike,u))
(L(tony,v), L(mike,v))
(¬A(w), ¬C(w), S(w))
```

则KB为：

KB Plain Text

```
[
[['A', 'tony']],
[['A', 'mike']],
[['A', 'john']],
[['L', 'tony', 'rain']],
[['L', 'tony', 'snow']],
[['¬A', 'x'], ['S', 'x'], ['C', 'x']],
[['¬C', 'y'], ['¬L', 'y', 'rain']],
[['L', 'z', 'snow'], ['¬S', 'z']],
[['¬L', 'tony', 'u'], ['¬L', 'mike', 'u']],
[['L', 'tony', 'v'], ['L', 'mike', 'v']],
[['¬A', 'w'], ['¬C', 'w'], ['S', 'w']]
]
```

(二) 归结合一算法

定理：S |–() 当且仅当 S |= ()， S |= () 当且仅当 S 是不可满足的

核心思想是双重循环暴力求解。先把文件内容转换为列表 KB ，一层为 i ，二层为 j ，然后枚举 KB[i] 里的元素和 KB[j] 里的元素，当且仅当两个元素互为对方的否定时，新建一个newkb 列表，然后把除这两者之外的 KB[i] 和 KB[j] 内的元素全部放入 newkb 里面，然后按规定格式放入 KB 末尾，循环直到 newkb 为空，那就意味着得到了空子句，推理结束。

合一时，传入两个原子公式，有两种情况：一个为变量，另一个为常量；两个为常量。第一种情况需要遍历原子公式，将变量替换情况变为元组写入列表assignment，然后返回。第二种情况将相反公式

去掉然后直接合一即可。

此外还有一个列表parent，用于存储归结操作的历史，以便后续进行剪枝操作。

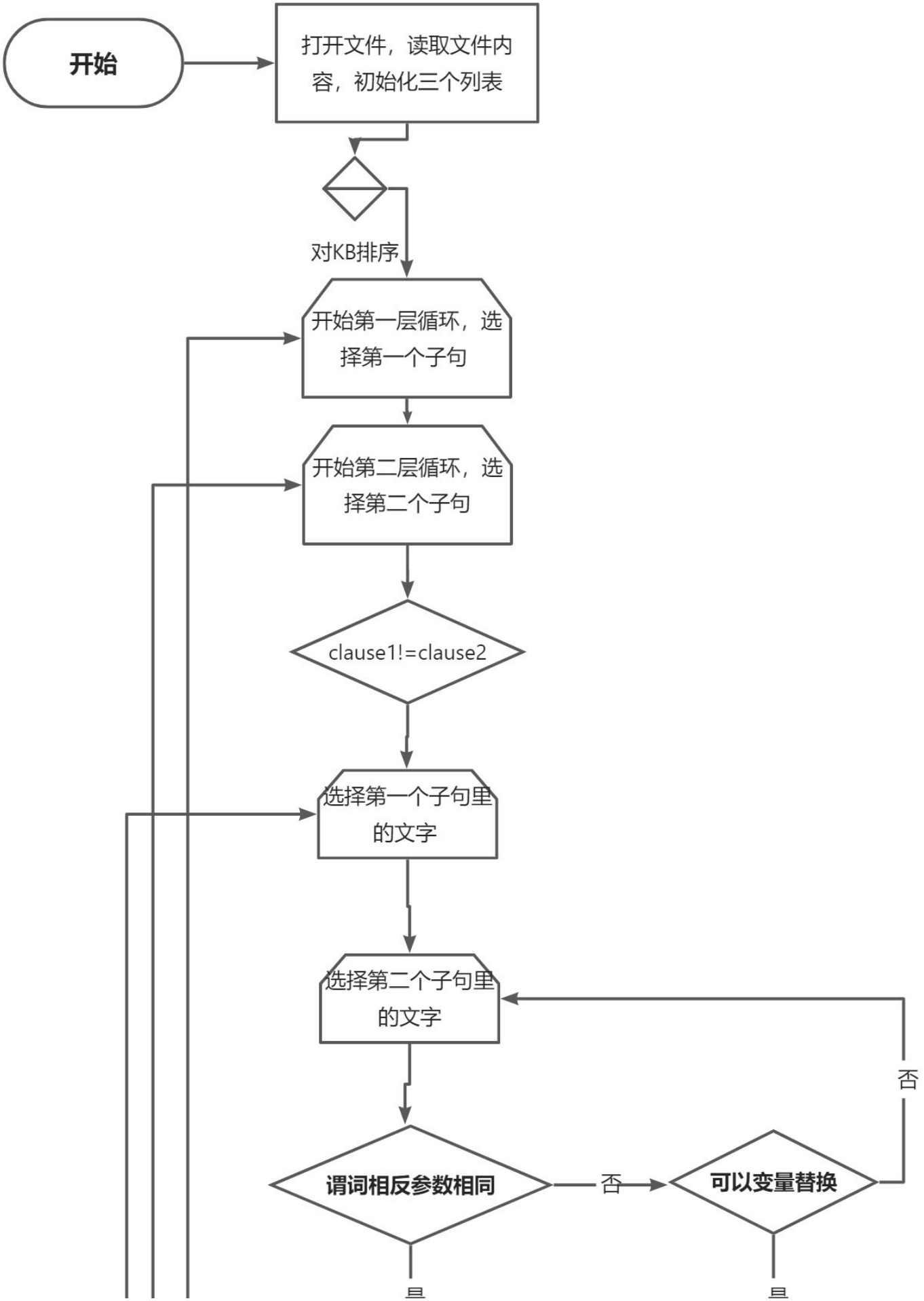
(三) 剪枝

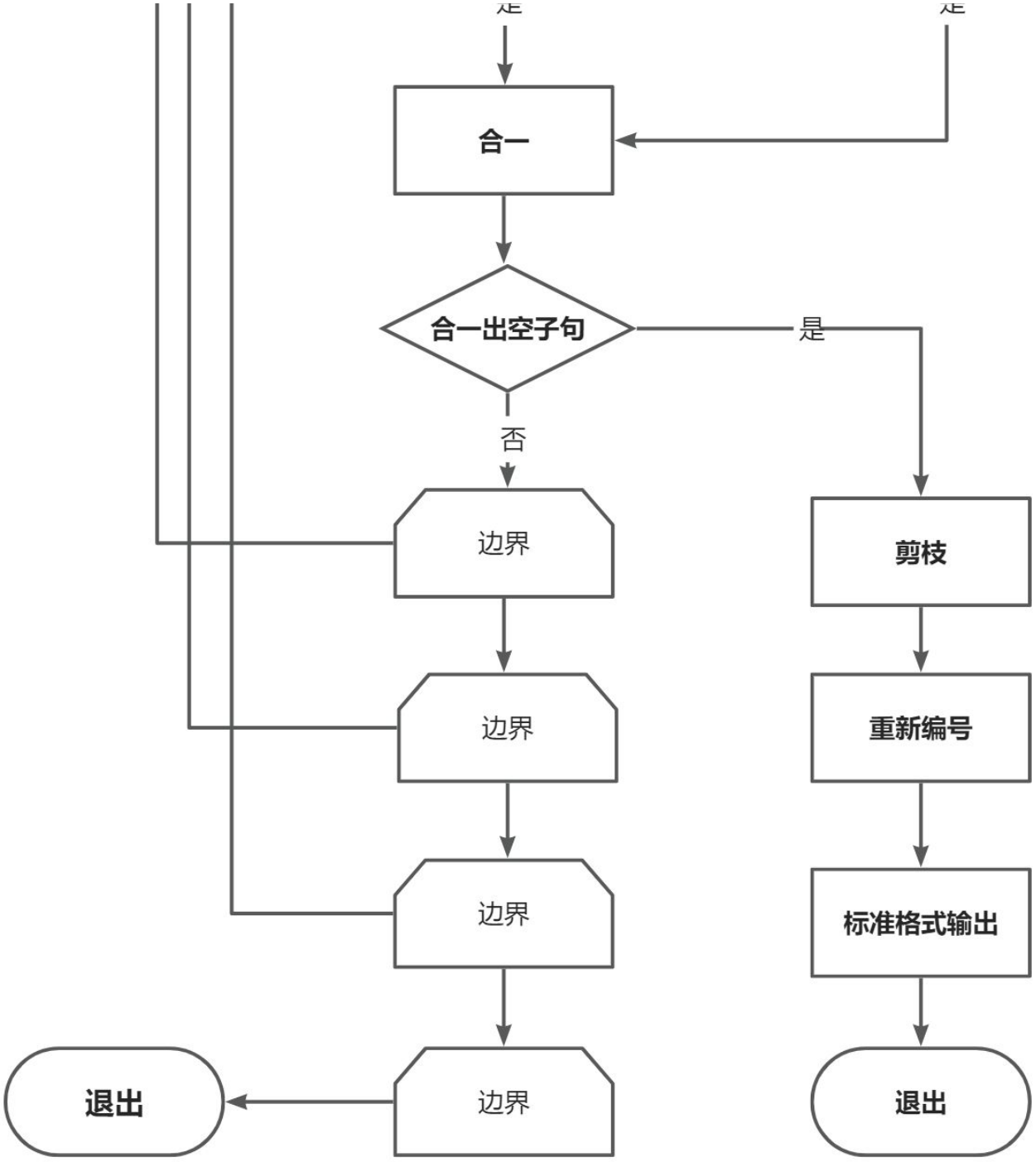
使用二叉树结构的层序遍历方法进行剪枝。先将空子句所属的KB，parent和assignment压入队列。从归结操作的历史记录中提取出需要的句子，然后将这些句子进行剪枝，以减少搜索空间。

(四) 重新标号

对剪枝后的知识库进行重新标号，使用字典对应新的编号。

流程图如下：


开始
打开文件，读取文件内容，初始化三个列表
对KB排序
开始第一层循环，选择第一个子句
开始第二层循环，选择第二个子句
clause1!=clause2
选择第一个子句里的文字
选择第二个子句里的文字
谓词相反参数相同
否
可以变量替换
否
是
是



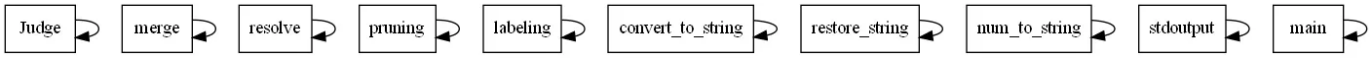
二、伪代码

```
函数 Judge(clause1, clause2):
    初始化 hash 列表
    对于 clause1 和 clause2 中的每个谓词:
        如果谓词不在变量列表中，并且两个谓词相同，则继续下一个谓词
        否则如果 clause1 中的谓词不在变量列表中，而 clause2 中的谓词在变量列表中:
            将 clause2 中的谓词和 clause1 中的谓词形成哈希对，并添加到 hash 列表中
        否则返回 False
    返回 hash 列表

函数 merge(KB, parent, assignment, clause1_index, i, clause2_index, j, hash_result=None):
    复制 clause1 和 clause2
    删除复制后的 clause1 和 clause2 中的第 i 和第 j 个谓词
    如果 hash_result 不为空:
        对于 hash_result 中的每个哈希对:
            在复制后的 clause2 中替换谓词
    将合一的历史记录添加到 parent 列表中
    将变量替换的结果添加到 assignment 列表中
    将新的知识库子句添加到知识库 KB 中
    如果新的知识库子句为空:
        返回 True，表示归结成功
    否则返回 False，表示归结失败

函数 resolve(KB, assignment, parent):
    对于知识库 KB 中的每个句子 clause1:
        对于知识库 KB 中的每个句子 clause2:
            如果 clause1 与 clause2 不是同一个句子:
                对于 clause1 中的每个谓词 predicate1:
                    对于 clause2 中的每个谓词 predicate2:
                        如果 predicate1 和 predicate2 是相反的谓词，并且参数相同:
                            如果合一成功:
                                返回
                        否则如果可以进行变量替换:
                            如果合一成功:
                                返回

函数 pruning(n, KB, assignment, parent):
    初始化 pruningkb 列表
```

```
    初始化队列 q，将归结操作的最后一步添加到队列中
    将归结操作的最后一步添加到 pruningkb 中
    当队列不为空时：
        取出队列中的当前项 cur
        如果当前项的第一个元素较大，则先将其添加到队列中，再将其添加到 pruningkb 中
        否则，先将其添加到 pruningkb 中，再将其添加到队列中
    返回 pruningkb 列表

函数 labeling(n, pruningkb):
    初始化 newindex 字典，用于存储新的编号
    初始化 seen_indexes 集合，用于记录已经见过的索引
    对于 pruningkb 中的每个项 item：
        获取项的父子句的索引 indexes
        对于 indexes 中的每个索引 index：
            如果索引不在 newindex 中，并且不在 seen_indexes 中：
                将索引添加到 newindex 中，并将其添加到 seen_indexes 中
    对 newindex 进行排序，并为每个索引分配新的编号
    返回 newindex 字典

函数 convert_to_string(lst):
    初始化结果字符串 result
    对于列表中的每个元组 item：
        将元组的第一个元素作为键，第二个元素作为值，拼接成字符串并添加到结果字符串中
    去除结果字符串的末尾逗号
    如果结果字符串为空，则返回空字符串
    否则，在结果字符串外层加上括号并返回

函数 restore_string(lst):
    初始化结果字符串 result
    对于列表中的每个元素 item：
        如果是第一个元素，则添加开头的字符串
        否则，添加逗号
    返回结果字符串去除末尾逗号并加上右括号

函数 num_to_string(kb, line, num):
    如果知识库 KB 中的某行只有一个谓词：
        返回空字符串
    否则，返回 ASCII 字符 a 加上 num 的字符串表示

函数 stdoutput(n, kb, pruningkb, newindex):
    初始化计数器 count 为 n
    对于 pruningkb 中的每个项 j，以及它的索引 i：
```

```
            如果索引 i 等于 pruningkb 的长度减去 1:
                打印结果，表示已到达最后一步
            否则:
                打印结果，包括合一的结果
            对于项 j 中的每个子句 k:
                如果 k 不是最后一个子句:
                    打印子句并添加逗号
                否则:
                    打印子句

    变量 variable 初始化为包含一些变量的列表

    主函数 main():
        从文件中读取知识库内容，包括知识库的个数和句子内容
        将读取的句子转换为谓词及其参数的形式，并存储在 KB 列表中
        初始化 assignment 和 parent 列表，用于记录变量替换和父子句关系
        对知识库进行归结推理，并记录推理历史
        对推理历史进行剪枝操作，并返回剪枝后的知识库
        对剪枝后的知识库进行重新标号，并返回新的编号字典
        输出剪枝后的知识库内容
```

三、关键代码展示



Judge函数用于形成两个可以进行变量替换的最一般合一元s:

Judge Python

```python
def Judge(clause1, clause2):
    # 变量替换
    hash = []
    for i in range(1, len(clause1)):
        if clause1[i] not in variable and clause2[i] not in variable and clause1[i] == clause2[i]:
            continue
        elif clause1[i] not in variable and clause2[i] in variable:
            hash.append((clause2[i], clause1[i]))
        else:
            return False
    return hash

```

merge函数用于将两个子句合一并追加到KB列表末尾：

merge Python

```python
def merge(KB, parent, assignment, clause1_index, i, clause2_index, j, hash_result=None):
    # 合一
    copyclause1 = copy.deepcopy(KB[clause1_index])
    copyclause2 = copy.deepcopy(KB[clause2_index])
    del copyclause1[i]
    del copyclause2[j]
    if hash_result:
        for hash_pair in hash_result:
            for index, predicate in enumerate(copyclause2):
                while hash_pair[0] in predicate:
                    copyclause2[index][predicate.index(hash_pair[0])] = hash_pair[1]

    parent.append([clause1_index, i, clause2_index, j])
    assignment.append(hash_result if hash_result else [])
    newkb = list(map(list, set(map(tuple, (copyclause1 + copyclause2)))))
    KB.append(newkb)
    if not newkb:
        return True
        # 能归结
    return False
```

resolve函数传入KB, assignment, parent，对子句集进行归结：

resolve Python

```python
def resolve(KB, assignment, parent):
    # 归结
    start_time = time.time()
    for clause1_index, clause1 in enumerate(KB):
        for clause2_index, clause2 in enumerate(KB):
            if clause1_index == clause2_index:
                continue
            for i, predicate1 in enumerate(clause1):
                for j, predicate2 in enumerate(clause2):
                    # 谓词相反
                    if (predicate1[0] == '¬' + predicate2[0] or predicate2[0] == '¬' + predicate1[0]) and len(predicate1) == len(predicate2):
                        if predicate1[1:] == predicate2[1:]:
                            if merge(KB, parent, assignment, clause1_index, i, clause2_index, j):
                                end_time = time.time()  # 记录结束时间
                                elapsed_time = end_time - start_time  # 计算时间差
                                if elapsed_time > 10:  # 如果时间超过10秒
                                    print("无法归结")
                                    exit()  # 终止整个程序
                                return
                        else:
                            hash_result = Judge(predicate1, predicate2)
                            if hash_result:
                                if merge(KB, parent, assignment, clause1_index, i, clause2_index, j, hash_result):
                                    end_time = time.time()  # 记录结束时间
                                    elapsed_time = end_time - start_time  # 计算时间差
                                    if elapsed_time > 10:  # 如果时间超过10秒
                                        print("无法归结")
                                        exit()  # 终止整个程序
                                    return

```

pruning采用二叉树的层序遍历对KB进行剪枝：

pruning Python

```python
def pruning(n, KB, assignment, parent):
    # 使用二叉树结构层序遍历剪枝
    pruningkb = []
    q = queue.Queue()
    q.put(parent[-1])
    pruningkb.append([KB[-1], parent[-1], assignment[-1]])

    # 只有非知识库内的句子才会有变量替换
    while not q.empty():
        cur = q.get()
        # 大的先进队列(后推出来)，符合推理常理
        if cur[0] > cur[2]:
            if cur[0] >= n:
                pruningkb.append([KB[cur[0]], parent[cur[0]], assignment[cur[0]]])
                q.put(parent[cur[0]])
            if cur[2] >= n:
                pruningkb.append([KB[cur[2]], parent[cur[2]], assignment[cur[2]]])
                q.put(parent[cur[2]])
        else:
            if cur[2] >= n:
                pruningkb.append([KB[cur[2]], parent[cur[2]], assignment[cur[2]]])
                q.put(parent[cur[2]])
            if cur[0] >= n:
                pruningkb.append([KB[cur[0]], parent[cur[0]], assignment[cur[0]]])
                q.put(parent[cur[0]])
    return pruningkb
```

labeling函数对剪枝后的pruningkb进行标号：

labeling Python

```python
def labeling(n, pruningkb):
    # 重新标号，使用字典对应
    newindex = {i: None for i in range(n)}
    seen_indexes = set()
    for item in pruningkb:
        indexes = item[1]
        # parent[0] and parent[2]
        for index in (indexes[0], indexes[2]):
            if index not in newindex and index not in seen_indexes:
                newindex[index] = None
                seen_indexes.add(index)
    newindex = sorted(newindex.keys())
    newindex = {x: newindex.index(x) + 1 for x in newindex}
    return newindex
```

接下来的函数实现了标准格式输出：

Python

```python
def convert_to_string(lst):
    # 变量替换
    # 初始化一个空字符串
    result = ""
    # 遍历列表中的元组
    for item in lst:
        # 将元组的第一个元素作为键，第二个元素作为值，拼接成字符串
        result += f"{item[0]}={item[1]},"
    # 去除最后一个逗号
    result = result.rstrip(",")
    # 返回结果字符串
    if result == "":
        return result
    return '(' + result + ')'


def restore_string(lst):
    # KB还原回正常形式
    # 初始化一个空字符串
    result = " "
    # 遍历列表中的元素
    for i, item in enumerate(lst):
        # 如果是第一个元素，添加开头的字符串
        if i == 0:
            result += item + '('
        else:
            result += item + ','
    # 返回结果字符串
    return result[:-1] + ') '


def num_to_string(kb, line, num):
    if len(kb[line]) == 1:
        return ''
    else:
        return chr(num + 97)


def stdoutput(n, kb, pruningkb, newindex):
    count = n
    for i, j in enumerate(pruningkb):
```

```
42          if i == len(pruningkb) - 1:
43              print(count + i + 1, f"R[{newindex[j[1][0]]},{newindex[j
    [1][2]]}] = []")
44          else:
45              #
46              print(count + i + 1,
47                    f"R[{newindex[j[1][0]]}{num_to_string(kb, j[1][0],
    j[1][1])},{newindex[j[1][2]]}{num_to_string(kb, j[1][2], j[1][3])}]{c
    onvert_to_string(j[2])} =",
48                    end='')
49          for k in range(len(j[0])):
50              if k is not len(j[0]) - 1:
51                  print(restore_string(j[0][k]), end=',')
52              else:
53                  print(restore_string(j[0][k]))
```

全部代码：

```python
import re
import queue
import copy
import time


def Judge(clause1, clause2):
    """
    判断两个子句是否可以进行变量替换。

    参数:
        clause1 (list): 第一个原子公式。
        clause2 (list): 第二个原子公式。

    返回:
        hash (list): 两个公式中对应变量的替换集。
        False (bool): 如果两个原子公式不能进行变量替换。
    """
    hash = []
    for i in range(1, len(clause1)):
        if clause1[i] not in variable and clause2[i] not in variable and clause1[i] == clause2[i]:
            continue
        elif clause1[i] not in variable and clause2[i] in variable:
            hash.append((clause2[i], clause1[i]))
        else:
            return False
    return hash


def merge(KB, parent, assignment, clause1_index, i, clause2_index, j, hash_result=None):
    """
    合并知识库中的两个子句。

    参数:
        KB (list): 知识库。
        parent (list): 知识库中每个子句的父子句列表。
        assignment (list): 知识库中每个子句的变量分配列表。
        clause1_index (int): 知识库中第一个子句的索引。
        i (int): 从第一个子句中删除的原子公式的索引。
```

```
        clause2_index (int): 知识库中第二个子句的索引。
        j (int): 从第二个子句中删除的原子公式的索引。
        hash_result (list, optional): 两个子句中对应变量的替换列表。

    返回:
        True (bool): 如果新子句为空。
        False (bool): 如果新子句不为空。
    """
    copyclause1 = copy.deepcopy(KB[clause1_index])
    copyclause2 = copy.deepcopy(KB[clause2_index])
    del copyclause1[i]
    del copyclause2[j]
    if hash_result:
        for hash_pair in hash_result:
            for index, predicate in enumerate(copyclause2):
                while hash_pair[0] in predicate:
                    copyclause2[index][predicate.index(hash_pair
[0])] = hash_pair[1]

    parent.append([clause1_index, i, clause2_index, j])
    assignment.append(hash_result if hash_result else [])
    newkb = list(map(list, set(map(tuple, (copyclause1 + copyclause
2)))))
    KB.append(newkb)
    if not newkb:
        return True
        # 能归结
    return False


def resolve(KB, assignment, parent):
    """
    对知识库中的子句进行归结。

    参数:
        KB (list): 知识库。
        assignment (list): 知识库中每个子句的变量分配列表。
        parent (list): 知识库中每个子句的父子句列表。
    """
    start_time = time.time()
    for clause1_index, clause1 in enumerate(KB):
        for clause2_index, clause2 in enumerate(KB):
            if clause1_index == clause2_index:
                continue
```

```
            for i, predicate1 in enumerate(clause1):
                for j, predicate2 in enumerate(clause2):
                    # 谓词相反
                    if (predicate1[0] == '¬' + predicate2[0] or predicate2[0] == '¬' + predicate1[0]) and len(predicate1) == len(predicate2):
                        if predicate1[1:] == predicate2[1:]:
                            if merge(KB, parent, assignment, clause1_index, i, clause2_index, j):
                                end_time = time.time()  # 记录结束时间
                                elapsed_time = end_time - start_time  # 计算时间差
                                if elapsed_time > 10:  # 如果时间超过10秒
                                    print("无法归结")
                                    exit()  # 终止整个程序
                                return
                        else:
                            hash_result = Judge(predicate1, predicate2)
                            if hash_result:
                                if merge(KB, parent, assignment, clause1_index, i, clause2_index, j, hash_result):
                                    end_time = time.time()  # 记录结束时间
                                    elapsed_time = end_time - start_time  # 计算时间差
                                    if elapsed_time > 10:  # 如果时间超过10秒
                                        print("无法归结")
                                        exit()  # 终止整个程序
                                    return


def pruning(n, KB, assignment, parent):
    """
    对归结树进行剪枝。

    参数:
        n (int): 原始知识库中的子句数量。
        KB (list): 知识库。
        assignment (list): 知识库中每个子句的变量分配列表。
        parent (list): 知识库中每个子句的父子句列表。

```

```
    返回:
        pruningkb (list): 剪枝后的知识库。
    """
    pruningkb = []
    q = queue.Queue()
    q.put(parent[-1])
    pruningkb.append([KB[-1], parent[-1], assignment[-1]])

    # 只有非知识库内的句子才会有变量替换
    while not q.empty():
        cur = q.get()
        # 大的先进队列(后推出来)，符合推理常理
        if cur[0] > cur[2]:
            if cur[0] >= n:
                pruningkb.append([KB[cur[0]], parent[cur[0]], assignment[cur[0]]])
                q.put(parent[cur[0]])
            if cur[2] >= n:
                pruningkb.append([KB[cur[2]], parent[cur[2]], assignment[cur[2]]])
                q.put(parent[cur[2]])
        else:
            if cur[2] >= n:
                pruningkb.append([KB[cur[2]], parent[cur[2]], assignment[cur[2]]])
                q.put(parent[cur[2]])
            if cur[0] >= n:
                pruningkb.append([KB[cur[0]], parent[cur[0]], assignment[cur[0]]])
                q.put(parent[cur[0]])
    return pruningkb


def labeling(n, pruningkb):
    """
    对剪枝后的知识库中的子句进行标号。

    参数:
        n (int): 原始知识库中的子句数量。
        pruningkb (list): 剪枝后的知识库。

    返回:
        newindex (dict): 一个字典，将旧的子句索引映射到新的子句索引。
    """
```

```
    newindex = {i: None for i in range(n)}
    seen_indexes = set()
    for item in pruningkb:
        indexes = item[1]
        # parent[0] and parent[2]
        for index in (indexes[0], indexes[2]):
            if index not in newindex and index not in seen_indexes:
                newindex[index] = None
                seen_indexes.add(index)
    newindex = sorted(newindex.keys())
    newindex = {x: newindex.index(x) + 1 for x in newindex}
    return newindex


def convert_to_string(lst):
    """
    将变量分配列表转换为字符串。

    参数:
        lst (list): 变量分配列表。

    返回:
        str: 变量分配的字符串表示。
    """
    # 初始化一个空字符串
    result = ""
    # 遍历列表中的元组
    for item in lst:
        # 将元组的第一个元素作为键，第二个元素作为值，拼接成字符串
        result += f"{item[0]}={item[1]},"
    # 去除最后一个逗号
    result = result.rstrip(",")
    # 返回结果字符串
    if result == "":
        return result
    return '(' + result + ')'


def restore_string(lst):
    """
    将谓词列表恢复为字符串。

    参数:
        lst (list): 谓词列表。
```

```
200
201         返回:
202             str: 谓词的字符串表示。
203         """
204         # 初始化一个空字符串
205         result = " "
206         # 遍历列表中的元素
207         for i, item in enumerate(lst):
208             # 如果是第一个元素，添加开头的字符串
209             if i == 0:
210                 result += item + '('
211             else:
212                 result += item + ','
213         # 返回结果字符串
214         return result[:-1] + ') '
215
216
217     def num_to_string(kb, line, num):
218         """
219         将数字转换为字符串。
220
221         参数:
222             kb (list): 知识库。
223             line (int): 知识库中的子句索引。
224             num (int): 要转换的数字。
225
226         返回:
227             str: 数字的字符串表示。
228         """
229         if len(kb[line]) == 1:
230             return ''
231         else:
232             return chr(num + 97)
233
234
235     def stdoutput(n, kb, pruningkb, newindex):
236         """
237         生成标准输出。
238
239         参数:
240             n (int): 原始知识库中的子句数量。
241             KB (list): 知识库。
242             pruningkb (list): 剪枝后的知识库。
243             newindex (dict): 一个字典，将旧的子句索引映射到新的子句索引。
```

```
244
245         返回:
246             output (list): 标准输出。
247         """
248         count = n
249         for i, j in enumerate(pruningkb):
250             if i == len(pruningkb) - 1:
251                 print(count + i + 1, f"R[{newindex[j[1][0]]},{newindex[j
    [1][2]]}] = []")
252             else:
253                 #
254                 print(count + i + 1,
255                       f"R[{newindex[j[1][0]]}{num_to_string(kb, j[1]
    [0], j[1][1])},{newindex[j[1][2]]}{num_to_string(kb, j[1][2], j[1]
    [3])}]{convert_to_string(j[2])} =",
256                       end='')
257             for k in range(len(j[0])):
258                 if k is not len(j[0]) - 1:
259                     print(restore_string(j[0][k]), end=',')
260                 else:
261                     print(restore_string(j[0][k]))
262
263
264     variable = ['x', 'y', 'z', 'u', 'v', 'w', 'xx', 'yy', 'zz']
265
266
267     def main():
268         """
269         程序的主函数。
270         """
271         filename = "alpineclub.txt"
272         KB = []
273         n = 0
274         # 打开文件
275         with open(filename, 'r', encoding="utf-8") as file:
276             # 打印知识库
277             # 使用计数器跳过第一行
278             line_count = 0
279             for i, line in enumerate(file):
280                 # 如果是第一行，则跳过
281                 if line_count == 0:
282                     n = int(line)
283                     # 获取个数n
284                     print(n)
```

```
                line_count += 1
                continue
            print(i, line.strip())

            # 使用正则表达式匹配谓词及其参数
            matches = (re.findall(r'¬?\w+\(\w+,*\w*\)', line))
            '''
            https://docs.python.org/zh-cn/3/library/re.html
            ¬?: 匹配零个或一个否定符号（¬）。?表示前面的元素可选。
            \w+: 匹配一个或多个字母、数字或下划线，表示谓词或函数名称。
            \(: 匹配左括号。
            \w+: 再次匹配一个或多个字母、数字或下划线，表示参数中的第一个元素。
            ,*: 匹配零个或多个逗号。
            \w*: 匹配零个或多个字母、数字或下划线，表示参数中的其余元素。
            \): 匹配右括号。
            '''
            # 将匹配结果添加到 KB 列表中
            KB.append(matches)
    # 记忆变量替换的列表assignment和记录父子句的列表Parent
    assignment = [[] for _ in range(n)]
    parent = [[] for _ in range(n)]
    sorted(KB)

    for i in range(len(KB)):
        for j in range(len(KB[i])):
            KB[i][j] = KB[i][j].replace('(', ",").replace(')', '').split(',')

    resolve(KB, assignment, parent)
    pruningkb = pruning(n, KB, assignment, parent)
    newindex = labeling(n, pruningkb)
    pruningkb = pruningkb[::-1]
    stdoutput(n, KB, pruningkb, newindex)


if __name__ == '__main__':
    main()
```

四、创新点&优化

(一) 在这两道题里，如果采用循环暴力求解，则会产生非常多无用的结果，所以我们最终要去掉无用的步骤，采用pruning函数以树状结构从后往前推得到最简的结果。

(二) 如果对原始数据进行sort ，尽量让同谓词的数据靠近，有助于减少最终的实现步数。

3 实验结果及分析

1. 实验结果展示示例（可图可表可文字，尽量可视化）

   a. alpineclub.txt

```
D:\Anaconda\envs\py311\python.exe "C:\Users\10559\Desktop\学习\大二\人工智能\AI by taoyzh\E2_22336216\code\E2-22336216.py"
11
1 A(tony)
2 A(mike)
3 A(john)
4 L(tony,rain)
5 L(tony,snow)
6 (¬A(x),S(x),C(x))
7 (¬C(y), ¬L(y,rain))
8 (L(z,snow), ¬S(z))
9 (¬L(tony,u), ¬L(mike,u))
10 (L(tony,v), L(mike,v))
11 (¬A(w), ¬C(w), S(w))
12 R[2,6a](x=mike) = S(mike) , C(mike)
13 R[2,11a](w=mike) = S(mike) , ¬C(mike)
14 R[5,9a](u=snow) = ¬L(mike,snow)
15 R[12b,13b] = S(mike)
16 R[14,8a](z=mike) = ¬S(mike)
17 R[15,16] = []

进程已结束，退出代码为 0
```

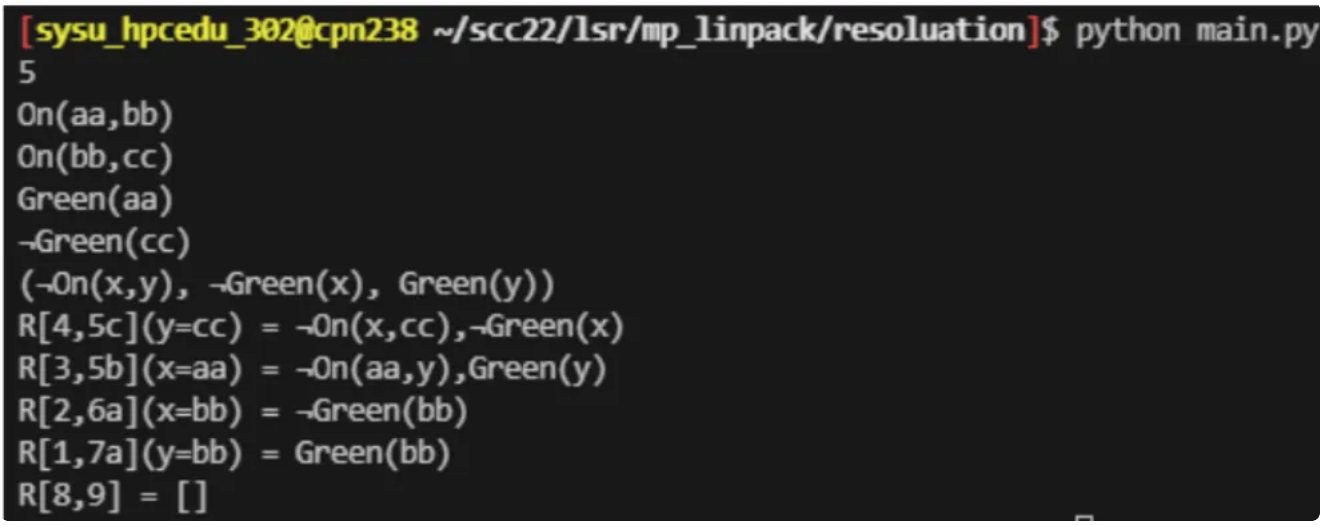
11 为子句数量，相较于PPT，截图里每一句都加了标号。

可见归结过程是正确的

   b. blockworld.txt

```
D:\Anaconda\envs\py311\python.exe "C:\Users\10559\Desktop\学习\大二\人工智能\AI by taoyzh\E2_22336216\code\E2-22336216.py"
5
1 On(aa,bb)
2 On(bb,cc)
3 Green(aa)
4 ¬Green(cc)
5 (¬On(x,y), ¬Green(x), Green(y))
6 R[1,5a](x=aa,y=bb) = Green(bb) , ¬Green(aa)
7 R[2,5a](x=bb,y=cc) = Green(cc) , ¬Green(bb)
8 R[3,6b] = Green(bb)
9 R[4,7a] = ¬Green(bb)
10 R[8,9] = []

进程已结束，退出代码为 0
```

此处和ppt上的不相同，但是可以看出也是正确的



2. 评测指标展示及分析

本程序的时间和空间复杂度取决于几个因素：

- 知识库的规模（KB）：知识库中句子的数量和每个句子中谓词的数量会直接影响算法的复杂度。更大规模的知识库会导致更多的归结操作和更长的剪枝过程。
- 归结操作：resolve 函数中的归结操作会对知识库中的每对句子进行配对和比较。在每次归结操作中，都要遍历句子中的每个谓词，因此其时间复杂度取决于知识库的规模。
- 剪枝操作（pruning）：剪枝操作使用了二叉树结构的层序遍历方法，对归结操作的历史记录进行剪枝。这涉及到队列的操作，因此其时间复杂度为 O(m)，其中 m 是历史记录的长度。
- 变量替换和重新标号（Judge、labeling）：在合一操作中涉及变量替换和重新标号，这通常是线性复杂度的操作，因为它们需要遍历句子中的每个谓词。因此，其时间复杂度也取决于知识库的规模。

基于以上因素，我们可以对程序的时间和空间复杂度做出以下估计：

时间复杂度：主要由归结操作和剪枝操作决定，通常为 O(n^2 * m)，其中 n 是知识库中句子的数量，m 是知识库中每个句子的谓词数量的平均值。

空间复杂度：主要由知识库的规模和归结过程中使用的额外数据结构（如 parent 和 assignment 列表）决定，通常为 O(n * m)，其中 n 是知识库中句子的数量，m 是知识库中每个句子的谓词数量的平均值。

4 思考题

无

5 参考资料

合一算法的Python实现--人工智能_unify算法-CSDN博客

3置换与合一.5归结原理.人工智能复习笔记_置换与合一概念-CSDN博客

【AI学习笔记】置换、合一、归结原理_合一置换-CSDN博客

正则表达式 – 语法 | 菜鸟教程