中山大学计算机学院

人工智能

本科生实验报告

（2022 学年春季学期）

课程名称：Artificial Intelligence

教学班级	计科二班	专业（方向）	计算机科学与技术
学号	20337263	姓名	俞泽斌

一、 实验题目

编写程序，实现一阶逻辑归结算法，并用于求解给出的三个逻辑推理问题，要求输出按照如下格式：

1. (P(x),Q(g(x)))
2. (R(a),Q(z),¬P(a))
3. R[1a,2c]{X=a} (Q(g(a)),R(a),Q(z))

… …

“R” 表示归结步骤.

“1a” 表示第一个子句(1-th)中的第一个 (a-th)个原子公式，即 P(x).

“2c”表示第二个子句(1-th)中的第三个 (c-th)个原子公式，即¬P(a).

“1a”和“2c”是冲突的，所以应用最小合一{X = a}.

二、 实验内容

1. 算法原理

首先是关于归结原理，就是通过两个含有互斥表达式的析取通过归结来消去其中的互斥项，然后如果其中的互斥项是有实例化的形式的，那得到的新的析取范式应该是在实例化之后的析取范式，然后把新的范式加入整个表达式集合中，再进行下一次循环，直到最后的表达式集合为空

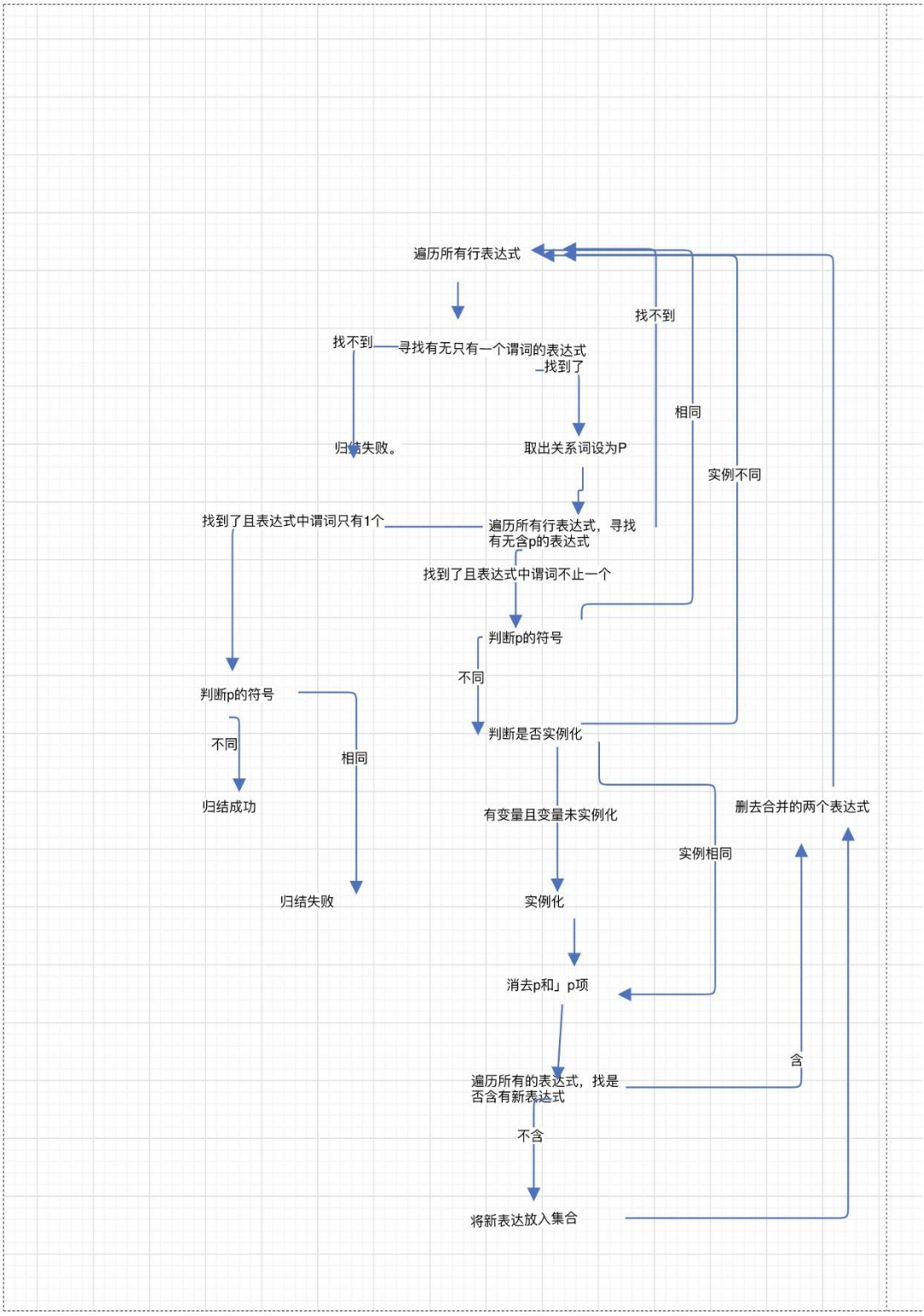
我这里使用的基本代码思想是通过对于只有一项的表达式来进行集中操作

首先是我通过一个 flag 信号来代码有没有归结成功，由于直接没有判断不成功后怎么终止，所以如果碰到不成功的归结案例可能会发生死循环，而归结成功后就不会再次进行输出，并且结束最外层的循环

然后通过一项的集合来操作，就是我一开始的找 clause_list 的 len 为 1 的时候，这时候一行只存在一个类，也就是一个谓词表达式，我先在所有的表达式中找到一个单个的无析取的表达式，取出他的关系字，然后在整个集合中找有没有谓词相同，但是前缀，也就是是这个表达式的取非形式的表达式，然后进行合并，如果需要实例化就实例化，如果两个都是实例化后的并且实例不一样就跳过这个表达式，如果得到的表达式与其中表达式集合中的某一项相同也跳过这一步操作，然后将实例化的表达式加入原来的表达式集合中，

同时将所有的经过归结的原表达式删去，一直到最后一项和合并后的实例化文件是相反的，那就最后输出一个空集，代表实验成功，标志位为 0，结束循环

2. 伪代码



```
    while flag:
        for i in range(len(clause_list)):
            if not flag:
                break
            if len(clause_list[i]) == 1:     # clause_list[i]只有一个谓词的子句，开始寻找后面的有相同的
                for j in range(0, len(clause_list)):     # clause_list[j]超过一个谓词的取或的子句
                    if i == j:
                        continue
                    # to_example 实例化
                    for k in range(len(clause_list[j])):  # 在子句 clause_list[j]中找相同的谓词，
                        do_time=0
                        if clause_list[i][0].get_weici() == clause_list[j][k].get_weici() and clause_list[i][0].fei() != clause_list[j][k].fei():
                            position = k  # clause_list[j][k]找到的谓词
                            do_time = do_time+1  # 找到了那就可以结束这一次 k 的循环了
                            for t in range(len(clause_list[j][k].element)-1):
                                # 找到那个谓词有没有实例化，如果没有，进行实例化
                                if len(clause_list[j][k].element[t+1]) == 1 or len(clause_list[i][0].element[t+1]) == 1:
                                    variable.append()
                                    example.append()
                                elif clause_list[j][k].element[t+1] != clause_list[i][0].element[t+1]:
                                    position = -1  # 找到了谓词一样但是实例不一样
                                    break
                            if do_time!=0:
                               break

                    if position == -1:
                        continue  # not found
                    new_clause = []
                    for k in range(len(clause_list[j])):
```

```
30.                             if k != position:    # 把要合并的那一行的除了合并的谓词的其他谓词都提取出来到 new_clause 里，并且把里面的变量都实例化
31.                                 cloze.createlist(clause_list[j][k].element)
32.                                 to_example()
33.                                 new_clause.append(cloze)
34.                         clause_list.append(new_clause)
```

3. 关键代码展示（带注释）

```python
class Clause:
    #  单个谓词
    element = []

    def __init__(self):
        self.element = []

    def createstr(self, t_str):
        if t_str[0] == ',':
            t_str = t_str[1:]
        s = ""
        for i in range(len(t_str)):
            s += t_str[i]
            if t_str[i] == '(' or t_str[i] == ',' or t_str[i] == ')':
                self.element.append(s[0:-1])
                s = ""

    def createlist(self, list2):
        for i in range(len(list2)):
            self.element.append(list2[i])

    def fei(self):      # whether ~
        return self.element[0][0] == "¬"

    def get_weici(self):
        if self.fei():
            return self.element[0][1:]
        else:
            return self.element[0]
```

这是对于一个谓词的定义，其实就是一个二重列表，然后对于这个谓词的构

造，我采用了两种方式，一种是通过对于字符串的构造，这个比较复杂一点，就是排除所有符号的影响，得到所有的字母然后存储到列表里，一种是通过列表的构造，那就是一个传参的过程，比较简单，其实所有的构造就是一个分割的过程，一个 A（x，y）的字符串我用[A,x,y]的列表来进行存储，然后我就可以通过第一个元素来得到整个谓词的关系，当然也要区别正负

```
def typein(clause_list):
    #   clause_list = []
    i = 0
    # f = codecs.open('data.txt', mode='r')  # 打开txt文件，以"utf-8'编码读取
    #line = f.readline()  # 以行的形式进行读取文件
    #tline = line
    tline =input()
    while tline != "":
    #while input()!="":

        if tline[0] == '(':
            tline = tline[1:-1]
            # 除去最外层括号
        clause_list.append([])
        t_str = ""
        #tline.replace(" ","")
        for j in range(len(tline)):
            if tline[j] == " ":
                continue
            t_str += tline[j]
            if tline[j] == ')':  # 分割谓词公式
                cloze = cs.Clause()
                cloze.createstr(t_str)
                clause_list[i].append(cloze)
                t_str = ""  # 一个谓词结束后清空字符串
        #tline = f.readline()
        tline=input()
        i = i + 1
```

这是所有的输入操作，原来的想法是通过文件内读入的方式，后来发现可能是编译器编码的问题得到的数据有问题，就采用了直接输入的方式，通过直接输入然后一行一行的读取，每一行都读到 tline 这个字符串里面，然后这个字符串首先将外括号去掉，然后将里面的字符串通过上述的谓词的构造函数得到 cloze 这个谓词，然后整个谓词表加上得到的新的谓词类，i+1 得到最后的新的长度

```
def to_example(element, variable, example):
    for i in range(len(variable)):
        for j in range(1, len(element)):
            if element[j] == variable[i]:
                element[j] = example[i]
```

这里的代码主要的是关于实例化的方面，也是对于一个谓词类来操作的，当谓词类需要实例化的时候，就在每一个谓词类里面找到每一个元素，如果是相关的需要实例化的变量，就进行实例化

```
# to_example 实例化
for k in range(len(clause_list[j])):     # 在子句clause_list[j]中找相同的谓词,
    do_time=0
    if clause_list[i][0].get_weici() == clause_list[j][k].get_weici() and clause_list[i][0].fei() != clause_list[j][k].fei():
        position = k  # clause_list[j][k]找到的谓词
        do_time = do_time+1  # 找到了那就可以结束这一次k的循环了
        for t in range(len(clause_list[j][k].element)-1):
            # 找到那个谓词有没有实例化，如果没有，进行实例化
            if len(clause_list[j][k].element[t+1]) == 1 or len(clause_list[i][0].element[t+1]) == 1:
                variable.append(clause_list[j][k].element[t+1])
                example.append(clause_list[i][0].element[t+1])
            elif clause_list[j][k].element[t+1] != clause_list[i][0].element[t+1]:
                position = -1  # 找到了谓词一样但是实例不一样
                break
        if do_time!= 0 :
            break
```

这里是整一个实例化的代码文件，主要的目标是找到什么地方需要实例化，实例化的是什么变量，得到实例又是什么，这里的操作主要是通过我一开始找到的自己的基本的元素，也就是 clause_list[i]，然后在整一个 clause_list 里找有没有和他关系词相同但是是取非状态下的表达式，然后将位置放到 position 里面，然后在这个位置的谓词来进行判断，找里面的元素是否实例化，如果实例化了就将得到的谓词也进行相同的实例化操作，其实也就是将变量和实例上面 append 需要的元素，然后调用里面的函数

```
    flag = True
    while flag:
        for i in range(len(clause_list)):
            if not flag:
                break
            if len(clause_list[i]) == 1:      # clause_list[i]只有一个谓词的子句，开始寻找后面的有相同的
                for j in range(0, len(clause_list)):      # clause_list[j]超过一个谓词的取或的子句
                    variable = []
                    example = []
                    position = -1
                    if not flag:
                        break
                    if i == j:
                        continue
```

这是整一个代码的判断条件，也就是一个 flag 条件，然后我用的基谓词是通过找到一个长度为 1 的谓词来进行寻找的，其实也就是没有存在表达式的析取操作，也便于对所有的谓词表来进行检索，不会像取一些含有析取的表达式为基本表达式后要考虑取哪一个元素为基本元素的操作

```
if position == -1:
    continue  # not found
new_clause = []
for k in range(len(clause_list[j])):
    if k != position:    # 把要合并的那一行的除了合并的谓词的其他谓词都提取出来到new_clause里，并且把里面的变量都实例化
        cloze =cs.Clause()
        cloze.createlist(clause_list[j][k].element)
        to_example(cloze.element,variable,example)
        new_clause.append(cloze)

if len(new_clause) == 1:
    for s in range(len(clause_list)):
        if len(clause_list[s]) == 1 and new_clause[0].element == clause_list[s][0].element:
            position = -1
            break
if position == -1:
    continue
clause_list.append(new_clause)
```

这里是对于整个谓词表的更新操作了，当找到了需要归结的两个谓词表达式后，就将其中相反的谓词项消去，然后进行实例化操作，最后得到一个新的谓词类，然后将这个新的谓词类与其他的所有谓词进行比较，如果存在相同的谓词表达式就删去这个新的类，进入下一次循环，如果没有相同的谓词表达式就将谓词表达式里新增一个归结后的表达式，进入下一次循环

这里也有一个终止条件的判断，首先是发现整个谓词表中第一个元素与基元素相同，同时并不是相反关系，就需要终止然后报错

```
    print(len(clause_list), end=":   ")
    th = chr(position+97)
    if len(variable)==0:
        print("R[", i+1, ",", j+1, th, "]", end="=")
    else:
        print("R[", i+1, ",", j+1, th, "]", end="(")
    for m in range(len(variable)):
        if m == len(variable)-1:
            print(variable[m], "=", example[m], ") = ", end="")
        else:
            print(variable[m], "=", example[m], end=",")
    print("(",end="")
    for n in range(len(clause_list[len(clause_list)-1])):
        print(clause_list[len(clause_list)-1][n].element, end="")
    print(")")
    if len(new_clause) == 1:
        for n in range(len(clause_list)-1):  # clause_list[j]超过一个谓词的取或的子句
            if len(clause_list[n]) == 1 and new_clause[0].get_weici() == clause_list[n][0].get_weici() \
                    and new_clause[0].element[1:] == clause_list[n][0].element[1:] \
                    and new_clause[0].fei() != clause_list[n][0].fei():
                print(len(clause_list)+1, ": R[", n+1, ",", len(clause_list), "]() =  \t[]")
                flag = False
                break
```

这是整块的输出以及最后成功的终止条件，就是谓词表最后存在一个表达式只有一个谓词，同时这个谓词的关系字符与选中的新的谓词相同，并且是相反关系，那说明归结成功，并且输出一个空集

4. 创新点&优化（如果有）

```python
class Clause:
    #  单个谓词
    element = []

    def __init__(self):
        self.element = []

    def createstr(self, t_str):
        if t_str[0] == ',':
            t_str = t_str[1:]
        s = ""
        for i in range(len(t_str)):
            s += t_str[i]
            if t_str[i] == '(' or t_str[i] == ',' or t_str[i] == ')':
                self.element.append(s[0:-1])
                s = ""

    def createlist(self, list2):
        for i in range(len(list2)):
            self.element.append(list2[i])

    def fei(self):      # whether ~
        return self.element[0][0] == "¬"

    def get_weici(self):
        if self.fei():
            return self.element[0][1:]
        else:
            return self.element[0]
```

首先是关于将一个谓词进行封装成类，来方便之后的操作，避免使用一个三重列表而混乱，同时将所有的类的创建也就是构造函数写明，将其中的关系谓词以及是否为非的判断都进行类里的操作，来方便后续 if 里条件调用的快捷性

```
def typein(clause_list):
    #   clause_list = []
    i = 0
    # f = codecs.open('data.txt', mode='r')  # 打开txt文件，以"utf-8'编码读取
    #line = f.readline()  # 以行的形式进行读取文件
    #tline = line
    tline =input()
    while tline != "":
    #while input()!="":

        if tline[0] == '(':
            tline = tline[1:-1]
            # 除去最外层括号
        clause_list.append([])
        t_str = ""
        #tline.replace(" ","")
```

可以采用从文件中读入的形式，也可以直接输入，在输入之后首先判断输入数据有无外括号，然后将其中的空格取出，一开始是打算直接使用 replace 函数，后来发现出现了乱码，可能是编码问题

三、 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）

```
E:\study\程序设计\python\新建文件夹\python.exe E:/study/人工智能/lab2/lab2.py
----
Enter your question
Please enter twice after you type in
----
```

开始的 ui

```
['On', 'aa', 'bb']
['On', 'bb', 'cc']
['Green', 'aa']
['¬Green', 'cc']
['¬On', 'x', 'y']['¬Green', 'x']['Green', 'y']
6:   R[ 1 , 5 a ](x = aa,y = bb ) = (['¬Green', 'aa']['Green', 'bb'])
7:   R[ 2 , 5 a ](x = bb,y = cc ) = (['¬Green', 'bb']['Green', 'cc'])
8:   R[ 3 , 5 b ](x = aa ) = (['¬On', 'aa', 'y']['Green', 'y'])
9:   R[ 3 , 6 a ]=(['Green', 'bb'])
10:   R[ 4 , 5 c ](y = cc ) = (['¬On', 'x', 'cc']['¬Green', 'x'])
11:   R[ 4 , 7 b ]=(['¬Green', 'bb'])
12 : R[ 9 , 11 ]() =    []
```

第一项样例输入和输出，可以看到最后的输出是空集，说明归结成功，其中里面有一样的过程，然后中间是归结的每一步实例化过程以及实例化变量

```
Enter your question
Please enter twice after you type in
----
GradStudent(sue)
(¬GradStudent(x), Student(x))
(¬Student(x),HardWorker(x))
¬HardWorker(sue)

['GradStudent', 'sue']
['¬GradStudent', 'x']['Student', 'x']
['¬Student', 'x']['HardWorker', 'x']
['¬HardWorker', 'sue']
5:   R[ 1 , 2 a ](x = sue ) = (['Student', 'sue'])
6:   R[ 4 , 3 b ](x = sue ) = (['¬Student', 'sue'])
7 : R[ 5 , 6 ]() =      []
```

第二项样例输入和输出，可以看到最后集合为空，集合归结成功

```
['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']
12:   R[ 1 , 6 a ](x = tony ) = (['S', 'tony']['C', 'tony'])
13:   R[ 1 , 11 a ](w = tony ) = (['¬C', 'tony']['S', 'tony'])
14:   R[ 2 , 6 a ](x = mike ) = (['S', 'mike']['C', 'mike'])
15:   R[ 2 , 11 a ](w = mike ) = (['¬C', 'mike']['S', 'mike'])
```

```
26:   R[ 3 , 11 a ](w = john ) = (['¬C', 'john']['S', 'john'])
27:   R[ 18 , 6 c ](x = tony ) = (['¬A', 'tony']['S', 'tony'])
28:   R[ 18 , 12 b ]=(['S', 'tony'])
29:   R[ 20 , 8 a ](z = mike ) = (['¬S', 'mike'])
30:   R[ 1 , 6 a ](x = tony ) = (['S', 'tony']['C', 'tony'])
31:   R[ 1 , 11 a ](w = tony ) = (['¬C', 'tony']['S', 'tony'])
32:   R[ 2 , 6 a ](x = mike ) = (['S', 'mike']['C', 'mike'])
33:   R[ 2 , 11 a ](w = mike ) = (['¬C', 'mike']['S', 'mike'])
34:   R[ 3 , 6 a ](x = john ) = (['S', 'john']['C', 'john'])
35:   R[ 3 , 11 a ](w = john ) = (['¬C', 'john']['S', 'john'])
36:   R[ 18 , 6 c ](x = tony ) = (['¬A', 'tony']['S', 'tony'])
37:   R[ 29 , 6 b ](x = mike ) = (['¬A', 'mike']['C', 'mike'])
38:   R[ 29 , 11 c ](w = mike ) = (['¬A', 'mike']['¬C', 'mike'])
39:   R[ 29 , 14 a ]=(['C', 'mike'])
40:   R[ 29 , 15 b ]=(['¬C', 'mike'])
41 : R[ 39 , 40 ]() =   []
```

第三项样例输入和输出，因为归结步骤有点多，主要还是在找只有一个的操作，但是有些步骤会出现重复和不必要的合一，然后改动后将实例化后一样的整个句子跳过输出步骤，就稍微减去一部分，但似乎还是与 ppt 上的输出行数有差距，但是也可以看出是输出了空集，就也是成功归结

四、 参考资料

用 Python 实现命题逻辑归结推理系统--人工智能

https://blog.csdn.net/Zhangguohao666/article/details/105471307/?ops_request_misc=&request_id=&biz_id=102&utm_term=%E5%AE%9E%E7%8E%B0%E4%B8%80%E9%98%B6%E9%80%BB%E8%BE%91%E5%BD%92%E7%BB%93%E7%AE%97%E6%B3%95&utm_medium=distribute.pc_search_result.none-task-blog-2~blog~sobaiduweb~default-0-105471307.nonecase&spm=1018.2226.3001.4450

PS：可以自己设计报告模板，但是内容必须包括上述的几个部分，不需要写实验感想