**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 | **系统结构班** | 专业（方向） | **计算机专业** |
| 学号 | **21307358** | 姓名 | **曾慧蕾** |

# 归结原理实验：

# 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}.

# 实验内容

1. 算法原理

以鲁宾孙消解原理为基础，采用宽度优先搜索策略进行穷举，直到最后消解到子句集为空时，从尾部开始向上找，寻找到有最短路径的解题思路（相当于建了个树）

归结的一般过程（宽度优先策略）

设有子句集S={C1,C2,C3,C4}，则对此子句集归结的一般过程是：

1. S内任意子句两两逐一进行归结，得到一组归结式，称为第一级归结式，记为S1。

2. 把S与S1内的任意子句两两逐一进行归结，得到一组归结式，称为第二级归结式，记为S2。

3. S和S1内的子句与S2内的任意子句两两逐一进行归结，得到一组归结式，称为第三级归结式，记为S3。

4. 如此继续，直到出现了空子句或者不能再继续归结为止。

1. 伪代码

输入并处理子句集（包括删除多的空号和空格）

创建答案列表

While 子句集不为空{

For I in range（0，当前训练集长度）

确定j长度：若当前为第一次循环，则更新为i+1；否则更新为上一次最后的子句集长度

For j in range（当前训练集长度）

检查训练集i项和j项能否合并：

若能合并，则合并之后更新到答案集内，并将i，j一起合并到答案集中方便后续找到需要的项

不能合并就继续下一项

如果这一段合并的结果为空，则说明找到了正确答案，退出两重循环。

}

根据答案集中给出的i，j，从尾部开始向上查找，并更新合并后的i、j，插入新的答案集中

反向输出新的答案集，得到答案。

1. 关键代码展示（带注释）

对于输入数据的处理，包括清除空格和括号从而统一格式

def cutspace(ss): #清除项中间的空格

    n=len(ss)

    i=2

    while i<n:

        if ss[i]==" " and ss[i-2]!=")":

            # ss=ss.replace(ss[i],"")

            list1=list(ss)

            list1.pop(i)

            ss=''.join(list1)

            n-=1

        i+=1

    return ss

def cutsign(l): #在对输入数据进行分割后，处理尾部的逗号

    for i in range(0,len(l)):

        if l[i][-1]==',':

            ss=l[i][0:-1]

            l[i]=ss

    return l

这几个函数都是功能性函数，主要是提取谓词中的各个部分和统计数量和取反从而消解

def cuthead(a): #取谓词

    ans1=a[0:a.find("(")]

    return ans1

def get\_one\_case(eg): #取单例

    return eg[eg.find("(")+1:eg.find(")")]

def get\_front\_case(eg): #取双例的前部分

    return eg[eg.find("(")+1:eg.find(",")]

def get\_behind\_case(eg): #取单例的后部分

    return eg[eg.find(",")+1:eg.find(")")]

def isvar(f): #判断是否是变量

    if f in['x','y','z','u','v','w']: return True

    else: return False

def countvar(case): #统计变量数

    sum=0

    for i in range(0,len(case)):

        if isvar(case[i])==True:

            sum+=1

    return sum

def opposite(a): #取反

    ans1=""

    if a[0]!="¬":

        ans1="¬"+a

    else:

        ans1=a[1:len(a)]

    return ans1

合一部分

def unify(list1,list2,i,j,l): #合一

    re=judge(list1,list2)

    if len(re)==0:

        return l

    l.append([])

    tail1=dic[re[0]]

    tail2=dic[re[1]]

    if len(list1[0])>1: a=str(i)+str(tail1)

    else: a=str(i)

    if len(list2[0])>1: b=str(j)+str(tail2)

    else: b=str(j)

    ss=re[2]

    t=[]

    l1=list1[0][:]

    l2=list2[0][:]

    for m in range(0,len(l1)):

        l1[m]=l1[m].replace(ss[1],ss[3:len(ss)-1])

    for m in range(0,len(l2)):

        l2[m]=l2[m].replace(ss[1],ss[3:len(ss)-1])

    list1[0]=l1

    list2[0]=l2

    for n in range(0,len(list1[0])):

        if n!=re[0]:

            t.append(list1[0][n])

    for n in range(0,len(list2[0])):

        if n!=re[1] and t.count(list2[0][n])==0:

            t.append(list2[0][n])

    l[len(l)-1].append(t)

    l[len(l)-1].append(a)

    l[len(l)-1].append(b)

    l[len(l)-1].append(ss)

    return l

判断能否合一，其中的两个实例由unify函数提供

def judge(list1,list2): #判断这两个结点有无可合一的部分

    list1\_index=-1

    list2\_index=-1

    flag1=0

    for i in range(0,len(list1[0])):

        head1=cuthead(list1[0][i])

        for j in range(0,len(list2[0])):

            head2=cuthead(list2[0][j])

            if head1==opposite(head2):

                list1\_index=i

                list2\_index=j

                flag1=1

                break

        if flag1==1:

            break

    if list1\_index==-1 and list2\_index==-1: return []

    else: #谓词可合一

        f1=str(list1[0][list1\_index])

        f2=str(list2[0][list2\_index])

        length=f1.count(",")

        if length==0:

            case1=get\_one\_case(f1)

            case2=get\_one\_case(f2)

            if case1==case2:

                return [list1\_index,list2\_index,"none"]

            elif isvar(case1)==True and isvar(case2)==False:

                return [list1\_index,list2\_index,"("+case1+"="+case2+")"]

            elif isvar(case1)==False and isvar(case2)==True:

                return [list1\_index,list2\_index,"("+case2+"="+case1+")"]

            else:

                return []

        elif length==1:

            case1=[get\_front\_case(f1),get\_behind\_case(f1)]

            if countvar(case1)>1: return []

            case2=[get\_front\_case(f2),get\_behind\_case(f2)]

            if countvar(case2)>1: return []

            ss=judgecase(case1,case2)

            if ss=="":return []

            else: return[list1\_index,list2\_index,ss]

对最后给出的答案集进行筛选（从尾部开始筛） 当筛查到了最初的训练集时则停止递归

def prepareans(l,node,ans,n):

    front=getnum(node[1])

    behind=getnum(node[2])

    if int(front)==-1 or int(behind)==-1:

        return

    tail=""

    if node[len(node)-1]=="none":

        tail=""

    else:

        tail=node[3]

    f=front

    front=gets(node[1],f)

    be=behind

    behind=gets(node[2],be)

    # t=str("R["+str(front)+","+str(behind)+"]"+tail+" = "+str(node[0]))

    t=[str(front),str(behind),tail,str(node[0])]

    ans.append(t)

    if int(f)>=n:prepareans(l,l[int(f)],ans,n)

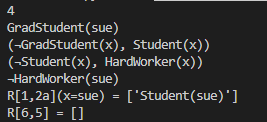
    if int(be)>=n:prepareans(l,l[int(be)],ans,n)

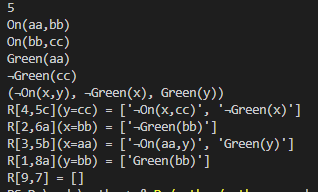
1. 创新点&优化（如果有）

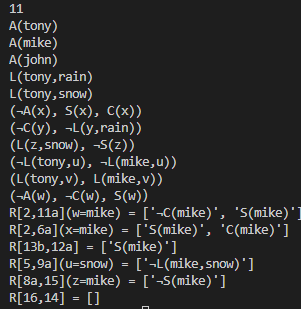
利用二叉树的思想，将归结过程构造成二叉树的形式，根节点为最终的归结结果，然后按层往上遍历，直到得到每一步的最终有效归结结果。

# 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）







2. 评测指标展示及分析（机器学习实验必须有此项，其它可分析运行时间等）

三个用例运行时间分别为：







**|-----------如有优化，请重复1，2，分析优化后的算法结果-----------------------|**

# 参考资料

**超算习堂-人工智能-课时3-课时5**

**主要参考了课件内一些搜索策略和鲁宾孙消解算法**

**合一算法参考了：**[合一算法的Python实现--人工智能\_索儿呀的博客-CSDN博客](https://blog.csdn.net/Zhangguohao666/article/details/105571115)

# 知识图谱实验

# 实验题目

编写程序，实现FOIL（First Order Inductive Learner）算法，对如下给定的知识图谱和目标谓词进行规则学习，并得到新的以目标谓词为关系的事实（用一阶逻辑表示）。

给定的知识图谱所有三元组的一阶逻辑如下：

Father(Jack,Dell)

Father(Dell,Stephen)

Grandfather(Jack,Stephen)

Father(Dell,Seth)

目标谓词

Grandfather(x,y)

要求的输入输出格式如下

输入格式

第1行：知识图谱的三元组数目𝑛𝑛

第2~(𝑛𝑛 + 1)行：三元组的一阶逻辑表示，例如L(David,Mike)

第𝑛𝑛 + 2行：目标谓词，即Grandfather(x,y)

输出格式

第1行：以目标谓词和前提约束谓词组成的规则推理，例如：

Mother（z,y）∧ Couple（x，z）→Father（x，y）

第2行及以后：每一行输出一条推理所得事实，用一阶逻辑表示，例如H (David, Ann)

# 实验内容

1、算法原理

主要采用一阶归纳推理规则，通过序贯覆盖实现规则推理。在给定目标谓词，FOIL算法从实例（正例、反例、背景知识样例）出发，不断测试所得推理规则是否还包含反例，一旦不包含，则学习结束，由此充分展示了“归纳学习”的能力。在学得推理规则后，利用背景知识再给推理规则中的变量赋予具体例子，经过“演绎”得到新的知识

2、伪代码

输入知识图谱所有三元组的一阶逻辑和目标谓词

根据输入创建训练集 得到最初的正反例数

While 正例没被包括完全或者仍包含反例：

初始化增益值标准

For I in range（0，训练集）：

得到第i项训练式的正例数

得到第i项训练式的反例数

若得到的正例数是0，则跳过 否则：

计算增益值

若增益值大于标准，则记录当前循环的训练式、正例数、反例数

遍历完一次训练集后，从中选择先前记录的训练式

更新答案集，已使用集，训练集

按照输出格式打印答案集

3、关键代码展示（带注释）

功能性函数 主要是对一阶谓词的拆分以及统计以及替换函数以及判断替换实例后两式是否相容

def get\_front\_case(eg): #取双例的前部分

    return eg[eg.find("(")+1:eg.find(",")]

def get\_behind\_case(eg): #取单例的后部分

    return eg[eg.find(",")+1:eg.find(")")]

def isvar(f): #判断是否是变量

    if f in['x','y','z','u','v','w']: return True

    else: return False

def get\_gainvalue(n\_t:int,n\_f:int,o\_t:int,o\_f:int): #计算增益值

    if n\_t==0: return -1

    front=math.log2(n\_t/(n\_t+n\_f))

    behind=math.log2(o\_t/(o\_t+o\_f))

    ans=o\_t\*(front-behind)

return ans

def meta(train,eg): #判断两个例子可判为正例或反例

    if train==eg:

        return True

    else:

        x1=get\_front\_case(eg)

        x2=get\_behind\_case(eg)

        if isvar(x1)==True:

            x3=get\_behind\_case(train)

            if x2==x3: return True

            else: return False

        elif isvar(x2)==True:

            x3=get\_front\_case(train)

            if x1==x3: return True

            else: return False

        else: return False

def replace\_f(tr\_e,bk,t\_replace): #算正反例时用到的替换函数

    x1=get\_front\_case(tr\_e)

    x1\_eg=get\_front\_case(bk)

    x2=get\_behind\_case(tr\_e)

    x2\_eg=get\_behind\_case(bk)

    t\_replace=t\_replace.replace(x1,x1\_eg)

    t\_replace=t\_replace.replace(x2,x2\_eg)

    return t\_replace

对于初始输入的一阶逻辑进行各种处理，包括初始化背景知识，得到训练集，得到拟加入前提约束谓词集

def get\_backgront\_knowledge(n:int): #最开始的输入

    bk=[]

    for i in range(n):

        a=input()

        bk.append(a)

    return bk

def get\_train\_knowledge(bk:list,goal:str): #初始化训练集合

    ans=[]

    j=0

    for j in range(0,len(bk)):

        if bk[j][:bk[j].find("(")]==goal[:goal.find("(")]:

            ans.append(bk[j])

            break

    g\_head="¬"+goal[:goal.find("(")] #头部取否 创造反例

    for i in range(0,len(bk)):

        if i!=j:

            t=g\_head+bk[i][bk[i].find("("):len(bk[i])]

            ans.append(t)

    return ans

def get\_train\_example(bk:list,goal:str): #得到前提约束谓词集

    have\_use=[]

    ans=[]

    goal\_head=goal[:goal.find("(")]

    for i in range(0,len(bk)):

        head=bk[i][:bk[i].find("(")]

        if head!=goal\_head:

            if len(have\_use)==0 or have\_use.count(head)==0:

                have\_use.append(head)

                for j in range(0,6):

                    t=str(head+tail[j])

                    ans.append(t)

    return ans

核心函数：得到正反例

def get\_m\_true(tr\_e,traink,bk,goal): #计算正例数

    if len(goal)==1:

        for i in range(0,len(bk)):

            if bk[i][:bk[i].find("(")]==tr\_e[:tr\_e.find("(")]:

                t\_replace=goal[0][:]

                t\_replace=replace\_f(tr\_e,bk[i],t\_replace)

                #然后在训练集中查找正反例 这里只用看第一个训练式

                flag=meta(traink[0],t\_replace)

                if flag==True: return 1

        return 0

    elif len(goal)==2:

        for i in range(0,len(bk)):

            if bk[i][:bk[i].find("(")]==tr\_e[:tr\_e.find("(")]:

                t1=goal[1][:]

                t1=replace\_f(tr\_e,bk[i],t1)

                for j in range(0,len(bk)):

                    flag=meta(bk[j],t1)

                    if flag==True:

                        t2=goal[0][:]

                        t2=replace\_f(tr\_e,bk[i],t2)

                        flag1=meta(traink[0],t2)

                        if flag1==True:return 1

        return 0

    return 0

def get\_m\_false(tr\_e,traink,bk,goal): #计算反例数

    ans=0

    use=[]

    if len(goal)==1:

        for i in range(0,len(bk)):

            if bk[i][:bk[i].find("(")]==tr\_e[:tr\_e.find("(")]:

                t\_replace=goal[0][:]

                t\_replace=replace\_f(tr\_e,bk[i],t\_replace)

                t\_replace="¬"+t\_replace

                if use.count(t\_replace)==0:  #已经检查过的例子就不再使用

                    use.append(t\_replace)

                    #然后在训练集中查找正反例 这里只用看第一个训练式

                    for j in range(1,len(traink)):

                        flag=meta(traink[j],t\_replace)

                        if flag==True:

                            ans+=1

        return ans

    elif len(goal)==2:

        for i in range(0,len(bk)):

            if bk[i][:bk[i].find("(")]==tr\_e[:tr\_e.find("(")]:

                t1=goal[1][:]

                t1=replace\_f(tr\_e,bk[i],t1)

                for j in range(0,len(bk)):

                    flag=meta(bk[j],t1)

                    if flag==True:

                        t2=goal[0][:]

                        t2=replace\_f(tr\_e,bk[i],t2)

                        t2="¬"+t2

                        flag1=meta(traink[1],t2)

                        if flag1==True: return 1

        return 0

    return 0

每次更新答案集前要先对训练集进行删减

def cut\_traink(traink,new\_eg,goal): #更新训练集

    if len(traink)==2: return traink

    x1=get\_front\_case(new\_eg)

    y1=get\_behind\_case(new\_eg)

    x2=get\_front\_case(goal)

    y2=get\_behind\_case(goal)

    if x1==x2:

        eg=get\_front\_case(traink[0])

        n=len(traink)

        i=1

        while i<n:

            eg\_check=get\_front\_case(traink[i])

            if eg\_check!=eg:

                del traink[i]

                n-=1

                i-=1

            i+=1

    elif y1==y2:

        eg=get\_behind\_case(traink[0])

        n=len(traink)

        i=1

        while i<n:

            eg\_check=get\_behind\_case(traink[i])

            if eg\_check!=eg:

                del traink[i]

                n-=1

                i-=1

            i+=1

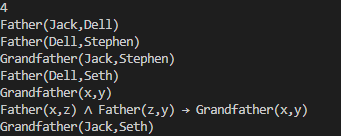
    return traink

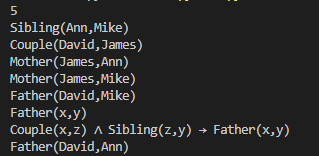
4、创新点&优化（如果有）

针对正例的搜索可以只对训练集第一项进行对比，所以我将正反例数的计算拆分开来从而减少了对整个训练集的遍历次数。

# 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）





2. 评测指标展示及分析（机器学习实验必须有此项，其它可分析运行时间等）

上面两例的运行时间分别为：





**|-----------如有优化，请重复1，2，分析优化后的算法结果-----------------------|**

# 参考资料

**超算习堂-人工智能-课时8-课时6**

**主要参考了课件内对训练集，背景知识，前提约束谓词集的划分以及正反例的计算方法**