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	20230349	专业（方向）	计算机科学与技术（系统结构）
学号	22336290	姓名	张超

一、 实验题目

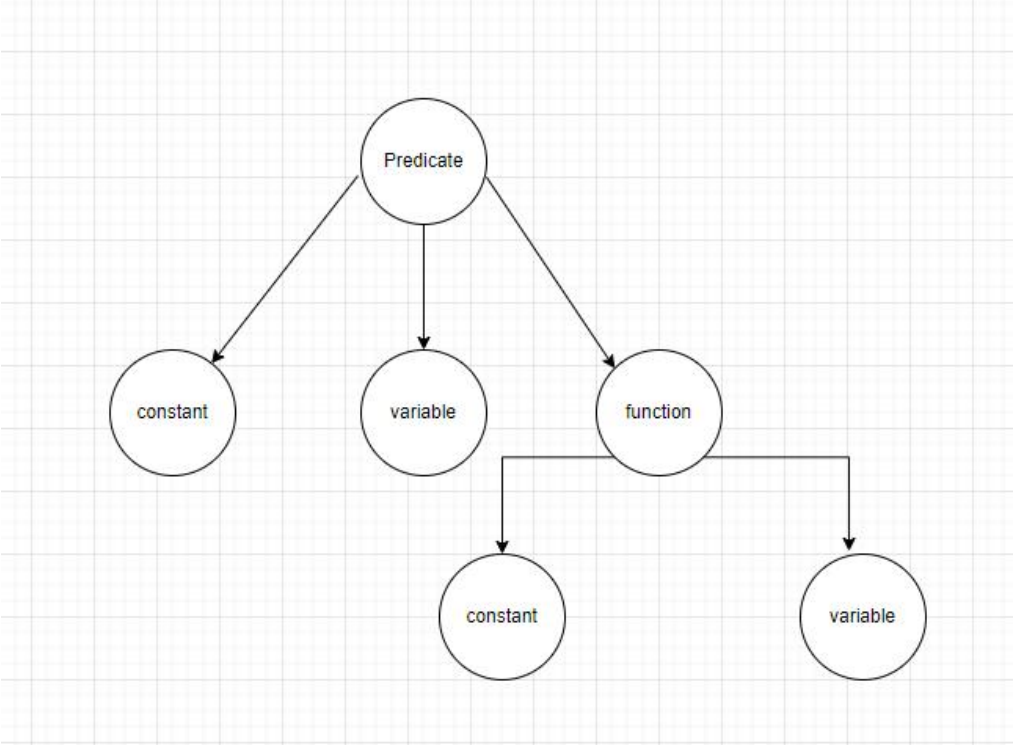
编写程序，实现一阶逻辑归结算法，并用于求解给出的两个逻辑推理问题。

二、 实验内容

1. 算法原理

一阶逻辑归结算法的原理，我把总共分成一下几步：

- **1.输入数据的处理**：如何处理输入数据其实涉及到的主要问题是用什么样的数据结构来存储输入的数据，这里我使用的树形结构，树形结构的好处是便于递归，而递归就可以函数的复合问题，通过树形结构我们可以更好的存储包含函数的项。我设计的数据结构如下图所示：

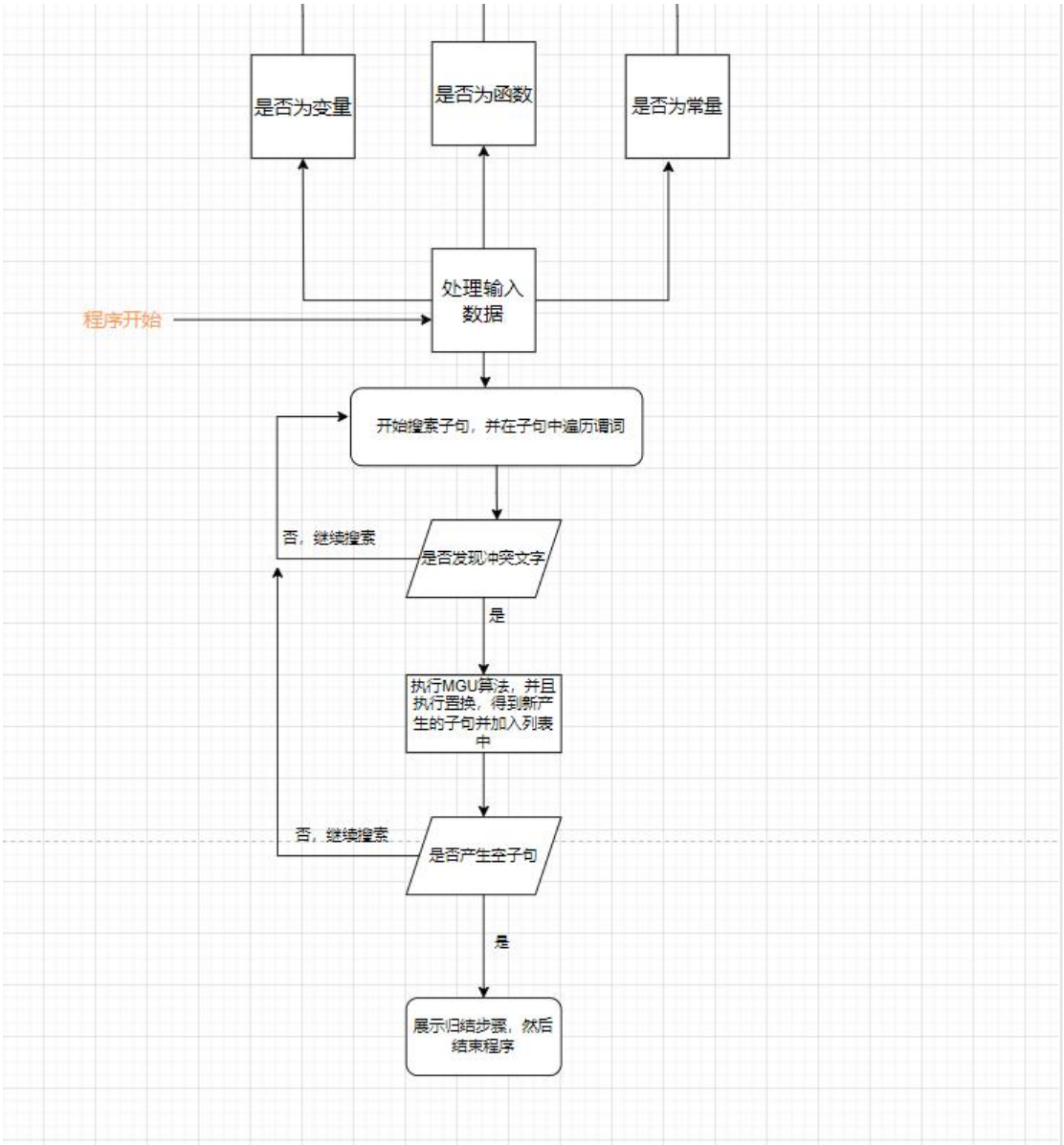


通过构建如图所示这样的一个树形结构对象（可以继续在 function 下添加参数，该图只是一个例子），就可以同时保存下谓词，变量，常量，参数。

- **2 搜索遍历子句**：关键点在于需要不遗漏的对任何两条子句进行匹配，测试能够归结。我的思路为：用一个列表存下每一条子句，遍历每一条字句（包括归结新产生的子句），对于每一条字句 i 再遍历下标在 i 之前的字句 j。如此就能不遗漏的对所有可能匹配的子句进行判断，即使是归结新产生的子句。
- **寻找冲突文字**：这一步骤比较简单，只需要判断谓词是否是否定关系。
- **MGU 算法**：我的 MGU 算法实现方法是，首先通过去掉冲突文字形成一个新的子句，然后遍历冲突谓词的参数，由于参数的类型不同而分情况处理，如参数是变量、常量类型或者变量、函数类型，都需要分情况处理然后加入差异集中，并去置换新产生的子句，通过不断遍历形成了逐项修补差异集的功能。遍历到最后便实现了 MGU 算法，同时也完成了置换。
- **展示归结过程**：由于归结过程产生了许多不需要的步骤，而这些步骤又不需要展示出来，所以当归结到空子句的时候，我找到产生空子句的两个子句，再通过这两个子句又找到四个子句，通过使用这种宽度优先搜索的策略就可以找到我们需要的子句

2. 伪代码

下图为程序的一个简单流程图：



3. 关键代码展示（带注释）

- 1.存储数据的数据结构：

```
    class Predicate:
        def __init__(self,name):
            self.name = name ;谓词的名字（字符串）
            self.arguments = [] #谓词的参数，包括了变量，常量，函数。参数以元组的类型存储，元组第一个元素为参数的名字，第二个元素表示参数的类型
        def __ne__(self, other: object) -> bool: #该函数定义了两个谓词不相等
            if self.name != other.name:
                return True
            else:
                for i in range(0,len(self.arguments)):
```

```
                    if self.arguments[i] != other.arguments[i]:
                        return True
            return False

    class function: #定义函数，函数中也有参数，包括了变量，常量，函数
        def __init__(self,name):
            self.name = name #函数的名字（字符串）
            self.arguments = []
```

- 遍历搜索子句和谓词：

```
def resolution(lines):
    length = len(lines) #lines 里面是所有子句，此种搜索方式可以搜索到后序加入的子句
    i = 0
    while i < length:
        for j in range(0,i): #搜索在第 i 句之前的子句
            for m in range(0,len(lines[i])):
                for n in range(0,len(lines[j])):
                    if find_clashing_predicate(lines[i][m],lines[j][n]):#找到冲突文字
                        MGU(lines[i][m],lines[j][n],j,i,n,m) #执行 MGU 算法
                        length = len(lines)
                        if success: #如果出现空子句，退出函数
                            return True
        i += 1
```

- MGU 算法：

```
def MGU(predicate,target,i,j,pre,tar):#predicate 和 target 是发生冲突的谓词
    flag = False
    add_line = []
    argu_cons = [] #差异集，用来存变量
    argu_vars = [] #差异集，用来存常量，函数在这里也被看作常量
    add_line = copy.deepcopy(lines[i][:pre]+lines[i][pre+1:]+lines[j][:tar]+lines[j][tar+1:]) #得到新子句但是还没发生置换
    for m in range(0,len(predicate.arguments)): #遍历谓词的参数
        if predicate.arguments[m] != target.arguments[m]:#下面为分类讨论，主要是为了区分变量，常量和函数。参数元组的第二位 0 代表常量，1 代表变量，2 代表函数
            if predicate.arguments[m][1] == 1 and target.arguments[m][1] == 0:
                argu_var = predicate.arguments[m]
                argu_con = target.arguments[m]
```

```
                argu_vars.append(argu_var)
                argu_cons.append(argu_con)
                flag = True
            elif predicate.arguments[m][1] == 0 and target.arguments[m][1]
== 1:
                argu_var = target.arguments[m]
                argu_con = predicate.arguments[m]
                argu_vars.append(argu_var)
                argu_cons.append(argu_con)
                flag = True
            elif predicate.arguments[m][1] == 1 and target.arguments[m][1]
== 1:
                argu_var = target.arguments[m]
                argu_con = predicate.arguments[m]
                argu_vars.append(argu_var)
                argu_cons.append(argu_con)
                flag = True
            elif predicate.arguments[m][1] == 2 and target.arguments[m][1]
== 1:
                argu_var = target.arguments[m]
                argu_con = predicate.arguments[m]
                argu_vars.append(argu_var)
                argu_cons.append(argu_con)
                flag = True
            elif predicate.arguments[m][1] == 1 and target.arguments[m][1]
== 2:
                argu_var = predicate.arguments[m]
                argu_con = target.arguments[m]
                argu_vars.append(argu_var)
                argu_cons.append(argu_con)
                flag = True
            elif predicate.arguments[m][1] == 2 and target.arguments[m][1]
== 2: #当两个参数都为函数的时候，情况特殊，需要递归的找到在函数内部的变量和对应
的常量
                argu_var,argu_con =
find_func_disagreement(predicate.arguments[m],target.arguments[m])
                argu_vars.append(argu_var)
                argu_cons.append(argu_con)
                flag = True
            else:
                return False
            if flag == True:
```

```
                add_line = substiution(argu_var,argu_con,add_line) #每次找到正确的参数，就置换新的子句，以此实现逐项修补差异集的效果。
    if no_repeat_line(add_line): #检查新的子句是否已经存在
        lines.append(add_line)
        add_line = delete_repeat_pre(add_line)
        cur_pos = len(lines) - 1
        record.append((add_line,i,j,pre,tar,argu_vars,argu_cons,cur_pos))
    if add_line == []: #如果新子句为空子句，则完成归结，将 success 置为 True
        cur_pos = len(lines)-1
        display_success((add_line,i,j,pre,tar,argu_vars,argu_cons,cur_pos))
        global success
        success = True
    return True
```

4. 创新点&优化（如果有）

1. 数据结构的设计：通过设计这种树形结构可以方便的存下谓词和各种参数，特别是函数这种参数，树形结构也非常适合递归，方便对函数这种递归的项进行操作。大大的减少了代码复杂度，提高了可读性，也使得程序更加完备，可以处理包含函数的归结推理。

2. MGU 算法：MGU 算法有一个重要步骤是逐项修补差异集，再发生置换。我的算法为直接对新子句产生置换，实现逐项修补差异集的效果。

3. 展示归结步骤：这里使用了宽度优先搜索的思想，通过维护一个队列，不断加入旧子句来追溯到归结的第一个步骤。

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）

测试 1:

```
(ai_sysu) xxww@myvb:~/ai_sysu$ python -u "/home/xxww/ai_sysu/lab2/resolution.py"
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))
R[2a,6a](x=mike) = (S(mike),C(mike))
R[5a,9a](u=snow) = (¬L(mike,snow))
R[8b,12a](z=mike) = (L(mike,snow),C(mike))
R[2a,11a](w=mike) = (¬C(mike),S(mike))
R[8a,13a](z=mike) = (¬S(mike))
R[13a,14a] = (C(mike))
R[15b,16a] = (¬C(mike))
R[17a,18a] = ()
```

测试 2：

```
(ai_sysu) xxww@myvb:~/ai_sysu$ python -u "/home/xxww/ai_sysu/lab2/resolution.py"
5
On(aa,bb)
On(bb,cc)
Green(aa)
¬Green(cc)
(¬On(x,y), ¬Green(x), Green(y))
R[1a,5a](x=aa,y=bb) = (¬Green(aa),Green(bb))
R[2a,5a](x=bb,y=cc) = (¬Green(bb),Green(cc))
R[3a,6a] = (Green(bb))
R[4a,7b] = (¬Green(bb))
R[8a,9a] = ()
```

测试 3：

```
(ai_sysu) xxww@myvb:~/ai_sysu$ python -u "/home/xxww/ai_sysu/lab2/resolution.py"
5
I(bb)
U(aa,bb)
¬F(u)
(¬I(y),¬U(x,y), F(f(z)))
(¬I(v), ¬U(w,v),E(w,f(w)))
R[1a,4a](y=bb) = (¬U(x,bb),F(f(z)))
R[2a,6a](x=aa) = (F(f(z)))
R[3a,7a](u=f(z)) = ()
```

测试 4：

```
(ai_sysu) xxww@myvb:~/ai_sysu$ python -u "/home/xxww/ai_sysu/lab2/resolution.py"
4
¬P(aa)
(P(z), ¬Q(f(z),f(u)))
(Q(x, f(g(y))), R(s))
¬R(t)
R[1a,2a](z=aa) = (¬Q(f(aa),f(u)))
R[3b,4a](s=t) = (Q(x,f(g(y))))
R[5a,6a](x=f(aa),u=g(y)) = ()
```

通过手算检验，四个测试都产生正确结果

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）

从上面的实验结果可以看出，程序展示的归结步骤并不复杂，有些测试比实验指导书给出的步骤更加简洁，可知该程序的搜索规则合理有效并且全面。

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

四、 思考题

无

五、 参考资料

实验指导书，课程 ppt

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