**斑马问题程序报告**

学号2112213姓名：冯思程

1. **问题重述**

首先，这次实验背景是基于python语言之上的逻辑编程。早期的人工智能就是以规则和逻辑推理作为主要研究方向，在逻辑编程的发展中发挥了重要的影响。可以说逻辑编程是人工智能发展的基石。

其次，对斑马问题进行简单的描述。有五个人分别住着不同颜色房子、养不同的宠物、有不同的工作、喝不同的饮料，而且这五个人的房子是在连着的一条线上。现在给出14个有关这些人和相关属性的正确命题，需要我们推断出这五人与其相关属性的正确匹配。

要用python3语言完成实验，能基本掌握逻辑编程的思想，了解逻辑编程与命令式编程的区别。能够依据给定的事实与规则编写代码，解决逻辑约束问题。

1. **设计思想**

首先，我约定每个独立的人是一个主体，其他相关的称为主体的属性，这里人的属性包括国籍、工作、饮料、宠物、房子颜色。

然后，我观察给定条件的性质，将14个条件进行一个划分，首先是确定的主体与其确定属性的条件，以及确定的同属于一个主体的属性的条件。有以下几条：

1.英国人住在红房子里

2西班牙人养了一条狗

3日本人是一个油漆工

4意大利人喝茶。

5摄影师养了一只蜗牛

6外交官住在黄房子里

7喜欢喝咖啡的人住在绿房子里

8小提琴家喜欢喝橘子汁

然后是表示实体在整体的确切属性的条件，有下面两个：

1挪威人住在左边的第一个房子里

2中间房子的人喜欢喝牛奶

最后是描述实体之间的不确定的关系的表达式

1绿房子在白房子的右边

2挪威人住在蓝房子旁边

3养狐狸的人所住的房子与医生的房子相邻

4养马的人所住的房子与外交官的房子相邻

然后是对不同种类的条件的编程实现：

首先是确定的主体与其确定属性的条件，以及确定的同属于一个主体的属性的条件。用membero函数进行约束，membero（item，coll）表示item是coll集合中的一个成员，在我们这次实验中，每个主体有五个属性，于是以英国人住在红房子里为例实现语句如下：

(membero, ('Britain', var(), var(), var(), 'red'), houses),

然后是表示主体在整体中的确切属性的条件，用eq函数，eq（a，b）表示ab相等。以挪威人住在左边的第一个房子里为例编程实现如下：

(eq, (('Norwegian', var(), var(), var(), var()) , var(), var(), var(), var()), houses),

最后是实现实体之间的不确定的关系的表达式。由于没有可以直接用的函数，所以我利用zip函数和conde函数def了两个新函数left函数和next函数用来表示实体房子属性的相邻关系。其中next是借助left来表达的，因为next关系就是a在b的left或者b在a的left。

函数定义如下：

def left(q, p, list):

return membero((q, p), zip(list, list[1:]))

def next(q, p, list):

return conde([left(q, p, list)], [left(p, q, list)])

以上是14个条件的逻辑编程实现，然后我注意到，斑马和矿泉水两个属性没有在上述的14个条件里面提到过，所以在最后要将的这两个变量添加进去。

以上是本次实验的主体逻辑编程的设计思想，然后进行solutions求解。

1. **代码内容**

from kanren import run, eq, membero, var, conde # kanren一个描述性Python逻辑编程系统

from kanren.core import lall # lall包用于定义规则

import time

def left(q, p, list):

return membero((q, p), zip(list, list[1:]))

def next(q, p, list): # p q 相邻意味着要么p在q左边，要么q在p左边

return conde([left(q, p, list)], [left(p, q, list)])

class Agent:

"""

推理智能体.

"""

def \_\_init\_\_(self):

"""

智能体初始化.

"""

self.units = var() # 单个unit变量指代一座房子的信息(国家，工作，饮料，宠物，颜色)

self.rules\_zebraproblem = None # 用lall包定义逻辑规则

self.solutions = None # 存储结果

def define\_rules(self):

"""

定义逻辑规则.

"""

self.rules\_zebraproblem = lall(

(eq, (var(), var(), var(), var(), var()), self.units), # self.units共包含五个unit成员，即每一个unit对应的var都指代一座房子(国家，工作，饮料，宠物，颜色)

# 各个unit房子又包含五个成员属性: (国家，工作，饮料，宠物，颜色)

(membero, ('英国人', var(), var(), var(), '红色'), self.units), # 英国人住在红房子里

(membero, ('西班牙人', var(), var(), '狗', var()), self.units), # 西班牙人养了一条狗

(membero, ('日本人', '油漆工', var(), var(), var()), self.units), # 日本人是一个油漆工

(membero, ('意大利人', var(), '茶', var(), var()), self.units), # 意大利人喝茶。

(membero, (var(), '摄影师', var(), '蜗牛', var()), self.units), # 摄影师养了一只蜗牛

(membero, (var(), '外交官', var(), var(), '黄色'), self.units), # 外交官住在黄房子里

(membero, (var(), var(), '咖啡', var(), '绿色'), self.units), # 喜欢喝咖啡的人住在绿房子里

(membero, (var(), '小提琴家', '橘子汁', var(), var()), self.units), # 小提琴家喜欢喝橘子汁

(eq, (('挪威人', var(), var(), var(), var()) , var(), var(), var(), var()), self.units), # 挪威人住在左边的第一个房子里

(eq, (var(), var(), (var(), var(), '牛奶', var(), var()), var(), var()), self.units), # 中间那个房子的人喜欢喝牛奶

(left, (var(), var(), var(), var(), '绿色'), (var(), var(), var(), var(), '白色'), self.units),# 绿房子在白房子的右边

(next, ('挪威人', var(), var(), var(), var()), (var(), var(), var(), var(), '蓝色'), self.units), # 挪威人住在蓝房子旁边。

(next, (var(), '医生', var(), var(), var()), (var(), var(), var(), '狐狸', var()), self.units), # 养狐狸的人所住的房子与医生的房子相邻  
(next, (var(), '外交官', var(), var(), var()), (var(), var(), var(), '马', var()), self.units),

# 示例：基于问题信息可以提炼出，有人养斑马，有人喜欢和矿泉水等信息

(membero,(var(), var(), var(), '斑马', var()), self.units),

(membero,(var(), var(), '矿泉水', var(), var()), self.units),

)

def solve(self):

"""

规则求解器(请勿修改此函数).

return: 斑马规则求解器给出的答案，共包含五条匹配信息，解唯一.

"""

self.define\_rules()

self.solutions = run(0, self.units, self.rules\_zebraproblem)

return self.solutions

agent = Agent()

solutions = agent.solve()

# 提取解释器的输出

output = [house for house in solutions[0] if '斑马' in house][0][4]

print ('\n{}房子里的人养斑马'.format(output))

output = [house for house in solutions[0] if '矿泉水' in house][0][4]

print ('{}房子里的人喜欢喝矿泉水'.format(output))

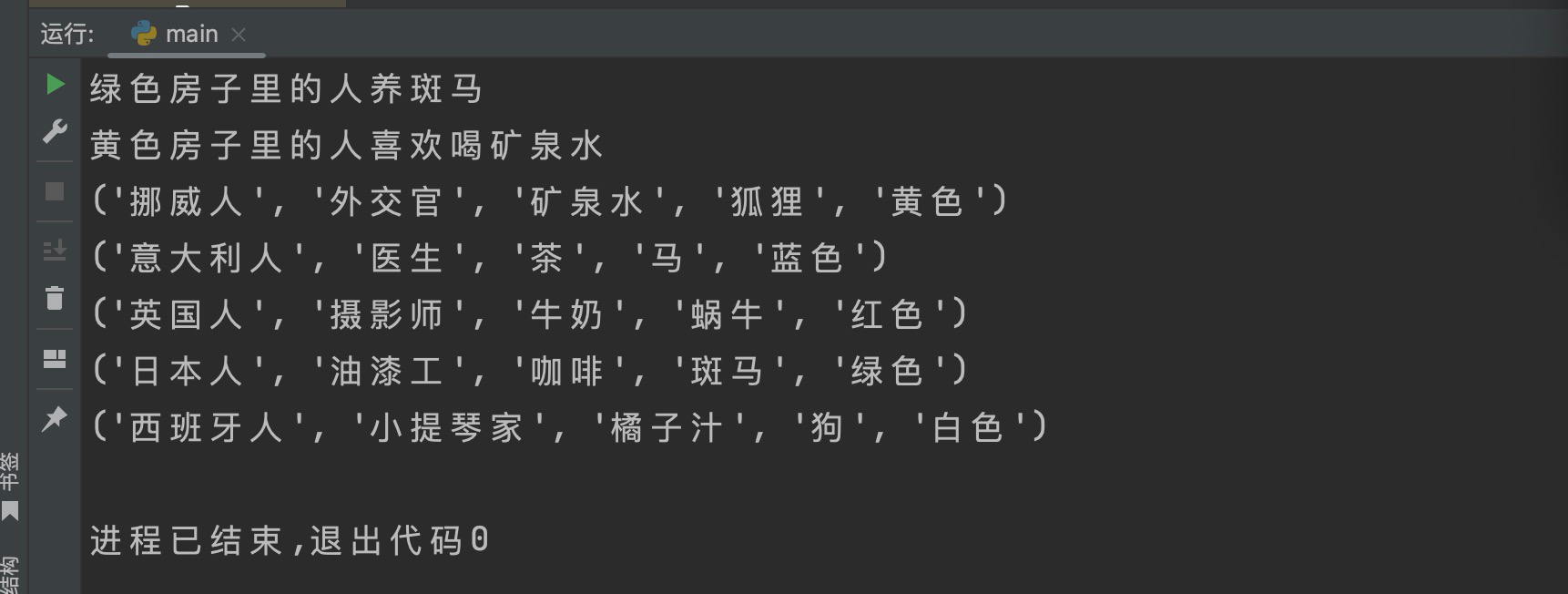
# 解释器的输出结果展示

for i in solutions[0]:

print(i)

1. **实验结果**

在pycharm上进行代码运行，运行结果如下：



1. **总结**

本次实验通过逻辑编程的方法成功解决了斑马问题，在改进方面我认为这次实验可以利用or-tools工具进行约束编程优化。

在本次实验的实现中遇到的主要困难是python的kanren包语法不熟练，以及对逻辑编程的思想不熟悉。但是老师已经在mo-lab环境下给出了整体编程框架，所以可以较快掌握。