**程序报告**

学号： 2212998 姓名：胡博浩

1. **问题重述**

**斑马问题：**

外交官、小提琴家、医生）的人分别住在一条街上的 5 所房子里，每所房子的颜色不同（红色、白色、蓝色、黄色、绿色），每个人都有自己养的不同宠物（狗、蜗牛、斑马、马、狐狸），喜欢喝不同的饮料（矿泉水、牛奶、茶、橘子汁、咖啡）。

根据以下提示，你能告诉我哪所房子里的人养斑马，哪所房子里的人喜欢喝矿泉水吗？

1. 英国人住在红色的房子里

2. 西班牙人养了一条狗

3. 日本人是一个油漆工

4. 意大利人喜欢喝茶

5. 挪威人住在左边的第一个房子里

6. 绿房子在白房子的右边

7. 摄影师养了一只蜗牛

8. 外交官住在黄房子里

9. 中间那个房子的人喜欢喝牛奶

10. 喜欢喝咖啡的人住在绿房子里

11. 挪威人住在蓝色的房子旁边

12. 小提琴家喜欢喝橘子汁

13. 养狐狸的人所住的房子与医生的房子相邻

14. 养马的人所住的房子与外交官的房子相邻

1. **设计思想**

（所采用的方法，有无对方法加以改进，该方法有哪些优化方向（参数调整，框架调整，或者指出方法的局限性和常见问题），伪代码，理论结果验证等… **思考题，非必填**）

====================================================================

本实验采用了演绎推理作为解决问题的方法，它适用于从一般到特殊的推理，而题目中要求的是通过一系列条件推断出特定的结论，因此演绎推理是比较适合的方法。在演绎推理这个问题中，设计思想主要涉及到使用逻辑编程库 kanren 进行推理，将条件形式化为表达式，加入约束到kanren的一个集合中，然后利用kanren内置的run求解即可得到问题的解。

需要注意的是，该方法有一定的逻辑编程的局限性，比如在处理大规模数据或复杂数值计算时可能效率不高，需要综合考虑问题的特点。

我们可以通过调整求解器的参数，比如迭代次数、搜索策略等，来提高求解效率或者准确性。

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

# 等价关系格式一: eq(var(), value) / eq(var(), var())

x = var() # 变量声明，kanren的推理基于变量var进行

z = var()

run(0, x, eq(x, z), eq(z, 3)) # 规则求解器，kanren的推理通过run函数进行

# 格式要求为: run(n, var(), rules,[rules, ...])

# 求解指定规则下符合的变量结果

# 等价关系格式二: (eq, var(), value) / (eq, var(), var())

x = var()

z = var()

run(0, x, (eq, x, z), (eq, z, 3))

# 属于关系格式 membero(var(), list / tuple)

x = var()

run(0, x, membero(x, (1, 2, 3)), # x is a member of (1, 2, 3) #x是（1,2,3）的成员之一

membero(x, (2, 3, 4))) # x is a member of (2, 3, 4) #x是（2,3,4）的成员之一

# 逻辑和关系格式 conde((rules, rules))

x = var()

run(0, x, conde((membero(x, (1, 2, 3)), membero(x, (2, 3, 4)))))

# 逻辑或关系格式 conde([rules], [rules]))

x = var()

run(0, x, conde([membero(x, (1, 2, 3))], [membero(x, (2, 3, 4))]))

# 调用lall包定义规则集合, lall(rules, [rules, ...])

x = var()

z = var()

rules = lall(

eq(x, z),

eq(z, 3)

)

run(0, x, rules)

# 调用lall包定义规则集合, lall(rules, [rules, ...])

x = var()

z = var()

rules = lall(

(eq, x, z),

(eq, z, 3)

)

run(0, x, rules)

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

###############################################################################

#### 可在此处定义自己所需要用到的自定义函数(可选)

####

#### 提示：定义左邻近规则left(), 定义右邻近规则right(),定义邻近规则next() ####

###############################################################################

# #

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)])

# #

###############################################################################

################# 非必要性工作 ##################

###############################################################################

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)

# 例如('英国人', '油漆工', '茶', '狗', '红色')即为正确格式，但不是本题答案

# 请基于给定的逻辑提示求解五条正确的答案

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)

##############################################################################

#### 请在以下区域中添加逻辑规则，感受逻辑约束问题 ####

#### 输出：五条房子匹配信息('英国人', '油漆工', '茶', '狗', '红色') ####

##############################################################################

#

(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。

(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

(left, # 绿房子在白房子的右边

(var(), var(), var(), var(), '绿色'),

(var(), var(), var(), var(), '白色'),

self.units),

(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

(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

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

(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。

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

(next, (var(), '医生', var(), var(), var()), # 养狐狸的人所住的房子与医生的房子相邻

(var(), var(), var(), '狐狸', var()), self.units),

(next, (var(), '外交官', var(), var(), var()), # 养马的人所住的房子与外交官的房子相邻

(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),

# #

##############################################################################

################# 完成后请记得提交作业 #################

##############################################################################

)

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**

在实验代码中填入相应的逻辑语句后点击运行，结果如下：

绿色房子里的人养斑马

黄色房子里的人喜欢喝矿泉水

('挪威人', '外交官', '矿泉水', '狐狸', '黄色')

('意大利人', '医生', '茶', '马', '蓝色')

('英国人', '摄影师', '牛奶', '蜗牛', '红色')

('日本人', '油漆工', '咖啡', '斑马', '绿色')

('西班牙人', '小提琴家', '橘子汁', '狗', '白色')

1. **总结**

（自评分析（是否达到目标预期，可能改进的方向，实现过程中遇到的困难，从哪些方面可以提升性能，模型的超参数和框架搜索是否合理等），**思考题，非必填**）

====================================================================

通过这次实验，我基本掌握了逻辑编程的思想，并且能够使用 Python 和 kanren 库解决逻辑约束问题。逻辑编程与命令式编程有着明显的区别，逻辑编程更侧重于描述问题的逻辑关系，而不是具体的计算步骤。通过演绎推理的方法，可以根据给定的事实和规则推断出特定的结论，这在解决一些复杂的逻辑问题时非常有用。

在实现过程中我对逻辑变量和规则一开始理解不够。我以后会着力加深对逻辑编程的理解，尝试解决更复杂的逻辑问题，学习更多关于kanren库和其他逻辑编程框架的知识，以及不断练习和深入理解逻辑编程的原理。