斑马问题程序报告

学号：2112514　　姓名：辛浩然

1、 **问题重述**

（简单描述对问题的理解，从问题中抓住主干，必填）

==========

斑马问题实质上是存在一些信息，如五个国家、五种颜色、五种宠物等，并给定相关信息的零碎的约束**条件**，如“西班牙人养了一条狗”。解决斑马问题，需要**综合依据**给定的这些**约束条件**，进行**逻辑推理**得到房子与这些方面信息的匹配结果。

2、 **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**）

==========

通过编程解决该问题，可以用到逻辑编程的方法。逻辑编程是一种编程典范，它设置答案**须匹配的规则**来解决问题。过程是：**事实+规则=结果**。事实就是一些已知输入，规则是应该遵循的限定条件，输入和输出应该遵循一定的规则，然后**找出所有满足这些规则的输出**，便是结果。对于该问题，将题目给定的约束条件设定为规则，然后找出满足规则的信息匹配结果。

使用 python 语言，可以借助 kanren 逻辑编程包。

3、 **代码内容**

（能体现解题思路的主要代码，有多个文件或模块可用多个"===="隔开，必填）

==========

（1）**实验完整代码**

```
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(m, n, list):
    return membero((m, n), zip(list, list[1:]))


def next(m, n, list):
    return conde([left(m, n, list)], [left(n, m, 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(
            # self.units 共包含五个 unit 成员，即每一个 unit 对应一座房子(国家，工作，饮料，宠物，颜色)
            (eq, (var(), var(), var(), var(), var()), self.units),
            # 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),
            (membero, ('西班牙人', var(), var(), '狗', var()), self.units),
            (membero, ('日本人', '油漆工', var(), var(), var()), self.units),
            (membero, ('意大利人', var(), '茶', var(), var()), self.units),
            (membero, (var(), '外交官', var(), var(), '黄色'), self.units),
            (membero, (var(), '摄影师', var(), '蜗牛', var()), self.units),
            (membero, (var(), var(), '咖啡', var(), '绿色'), self.units),
            (membero, (var(), '小提琴家', '橘子汁', var(), var()), self.units),

            (eq, (var(), var(), (var(), var(), '牛奶',
             var(), var()), var(), var()), self.units),
            (eq, (('挪威人', var(), var(), var(), var()),
             var(), var(), var(), var()), self.units),

             (left, (var(), var(), var(), var(), '白色'),
             (var(), var(), var(), var(), '绿色'), self.units),
            (next, ('挪威人', var(), var(), var(), var()),
             (var(), var(), var(), var(), '蓝色'), self.units),
            (next, (var(), '医生', var(), var(), var()),
             (var(), var(), var(), '狐狸', var()), self.units),
            (next, (var(), '外交官', var(), var(), var()),
             (var(), var(), var(), '马', var()), self.units),

            (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
```

（2）逻辑规则定义具体代码分析

```
 (eq, (var(), var(), var(), var(), var()), self.units),

# self.units 共包含五个 unit 成员，即每一个 unit 对应一座房子，每座房子包含五个属性(国家，工作，饮料，宠物，颜色)
# 这里用到了等价关系表达式 eq 语句。格式为(eq,var(),var())

# 然后定义规则：
# 第一类规则，是简单的描述不同方面的信息叠加：
# 利用成员关系表达式 membero 语句，同时具有某些特征的房子是 self.units 的成员之一。如语句(membero, ('英国人', var(), var(), var(), '红色'), self.units),表示有一栋房子的是红色且主人是英国人。
(membero, ('英国人', var(), var(), var(), '红色'), self.units),
(membero, ('西班牙人', var(), var(), '狗', var()), self.units),
(membero, ('日本人', '油漆工', var(), var(), var()), self.units),
(membero, ('意大利人', var(), '茶', var(), var()), self.units),
(membero, (var(), '外交官', var(), var(), '黄色'), self.units),
(membero, (var(), '摄影师', var(), '蜗牛', var()), self.units),
(membero, (var(), var(), '咖啡', var(), '绿色'), self.units),
(membero, (var(), '小提琴家', '橘子汁', var(), var()), self.units),

# 第二类规则，所给信息包含房子的绝对位置关系：
# 利用等价关系表达式 eq 语句，self.units 与这样描述的五个 unit 成员等价，其中的某个 unit 成员具有一些特征。
# 比如，中间那个房子的人喜欢喝牛奶  喝牛奶的这个房子在所有的 unit 成员中居中，self.units 与五个 unit 成员等价，其中最中间的 unit 成员的主人爱喝牛奶。
(eq, (var(), var(), (var(), var(), '牛奶',var(), var()), var(), var()),
self.units),
# 挪威人住在左边的第一个房子里
 (eq, (('挪威人', var(), var(), var(), var()), var(), var(), var(), var()),
self.units),

# 第三类规则，所给信息包含房子的相对位置关系：
# 此时需要自己定义函数，关于相对位置关系的函数

# left 函数表示 m 在 n 的左侧。具体实现：完整列表与去除第一个元素的列表打包为一个个元组，这些元组即为一个列表中左右相邻的两个元素对，(m, n)是其中一个。

def left(m, n, list):
    return membero((m, n), zip(list, list[1:]))

# next 函数表示 m 与 n 相邻。具体实现：借助 left 函数与逻辑或关系格式，m 与 n 相邻包括 m 在 n 左侧和 n 在 m 左侧两种情况。

def next(m, n, list):
    return conde([left(m, n, list)], [left(n, m, list)])
```

```
# 绿房子在白房子的右边
(left, (var(), var(), var(), var(), '白色'), (var(), var(), var(), var(), '绿色
'), self.units),
# 挪威人住在蓝色的房子旁边
 (next, ('挪威人', var(), var(), var(), var()), (var(), var(), var(), var(), '
蓝色'), self.units),
# 养狐狸的人与医生房子相邻
(next, (var(), '医生', var(), var(), var()), (var(), var(), var(), '狐狸', var()),
self.units),
# 养马的人与外交官房子相邻
(next, (var(), '外交官', var(), var(), var()), (var(), var(), var(), '马', var()),
self.units),

# 在上述规则定义结束后，还存在未提到的元素，对其进行单独声明
# 利用 membero 语句，如 self.units 有一个 unit 成员喜欢喝矿泉水。
(membero, (var(), var(), var(), '斑马', var()), self.units),
(membero, (var(), var(), '矿泉水', var(), var()), self.units)
```

4、 **实验结果**

（实验结果，必填）

==========

得到了匹配结果：

('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')
('西班牙人', '小提琴家', '橘子汁', '狗', '白色')
('日本人', '油漆工', '咖啡', '斑马', '绿色')

5、 **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**）

==========

在该实验中，第一次接触到逻辑编程。我感受到逻辑编程与以往编程的极大的不同，它是设置答案须匹配的规则来解决问题。通过实践，初步掌握了逻辑编程的基本思想与方法，熟悉了一些常用语句。