程序报告

学号： 2310422　　姓名：谢小珂

一、问题重述

斑马问题：5 个不同国家（英国、西班牙、日本、意大利、挪威）且工作各不相同（油漆工、摄影师、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，得出哪所房子里的人养斑马，哪所房子里的人喜欢喝矿泉水。
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

二、设计思想

·逻辑依据

演绎推理（Deductive Reasoning）是一种从一般到特殊的推理方法。在本题中，我们使用 Python 的逻辑编程库 kanren，将题目中的条件形式化为多个逻辑表达式，并将其作为约束加入 kanren 的集合中。随后，利用 kanren 内置的 run 函数进行求解，得到问题的解。
优化方法：对每条逻辑规则进行分析，去除冗余约束，降低复杂性；将相关规则适当合并，减少求解过程中的搜索深度和广度，提升效率；减少逻辑变量的数量，降低逻辑规则的复杂度，优化求解性能；引入异常处理机制，确保在不常见或异常情况下程序能够稳健运行。

·kanren 编程包的使用

```
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
```

・自定义函数：定义左邻近规则 left()、右邻近规则 right()和邻近规则 next()

```
#定义左邻近规则 left(), 定义右邻近规则 right(),定义邻近规则 next()
def right(x, y, list):
    return membero((y, x), zip(list, list[1:]))

def left(x,y,list):   #左边与右边逻辑相通
    return right(y,x,list)
def next(x, y, list):
    return conde( [left(x, y, list)],[right(x, y, list)])
```

・添加逻辑规则，感受逻辑约束问题

```
(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
(right, (var(), var(), var(), var(), '白色'),(var(), var(), var(), var(), '绿色'), self.units),# 绿房子在白房
子的右边
(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()), (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), # 养马的人所
住的房子与外交官的房子相邻
(membero, (var(), var(), var(), '斑马', var()), self.units),  # 有人养斑马
(membero, (var(), var(), '矿泉水', var(), var()), self.units)  # 有人喝水
```

三、代码内容

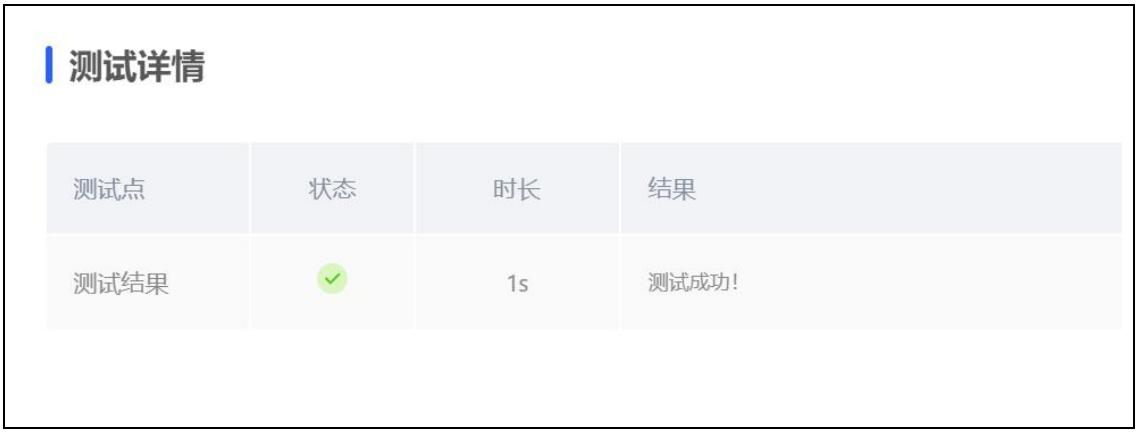
```
from kanren import run, eq, membero, var, conde
from kanren.core import lall   # lall 包用于定义规则
import time
```

```
def right(x, y, list):
    return membero((y, x), zip(list, list[1:]))
def left(x,y,list):
    return right(y,x,list)
def next(x, y, list):
    return conde( [left(x, y, list)],[right(x, y, list)])
class Agent:
    """
    推理智能体.
    """

    def __init__(self):
        """
        智能体初始化.
        """

        self.units = var()
        self.rules_zebraproblem = None
        self.solutions = None

    def define_rules(self):
        """
        定义逻辑规则.
        """

        self.rules_zebraproblem = lall(
            (eq, (var(), var(), var(), var(), var()), self.units),

            # 各个 unit 房子包含五个成员属性: (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。
            (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
            (right, (var(), var(), var(), var(), '白色'),(var(), var(), var(), var(), '绿色'),self.units),#
绿房子在白房子的右边
            (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()),(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),#
养马的人所住的房子与外交官的房子相邻
            (membero, (var(), var(), var(), '斑马', var()), self.units),   # 有人养斑马
            (membero, (var(), var(), '矿泉水', var(), var()), self.units)   # 有人喝水

    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))
```

四、实验结果

```
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')
```



五、总结

本次实验通过使用 Python 的逻辑编程库 kanren，成功解决了经典的斑马问题。实验的核心思想是利用演绎推理（Deductive Reasoning），将问题中的多个约束条件形式化为逻辑规则，并通过 kanren 的 run 函数进行求解。

1.实验中将提示条件逐一转化为逻辑规则，并通过 kanren 的 lall 函数将这些规则组合成一个完整的约束系统。每条规则都清晰地定义了国家、职业、饮料、宠物和房屋颜色之间的关系。

2. 为了处理房屋之间的相邻关系，实验定义了 left、right 和 next 函数，用于描述房屋之间的左右相邻关系。这些函数通过 membero 和 conde 实现，确保了逻辑推理的准确性。

4.在实验过程中，通过合并相关规则、减少冗余约束、优化变量数量等方式，提升了求解效率。同时，代码结构清晰，逻辑规则易于扩展和维护。

5. 本次实验不仅验证了逻辑编程在解决复杂约束问题中的有效性，还展示了如何通过形式化逻辑规则和优化求解过程，高效解决经典的逻辑推理问题。

6.逻辑规则的定义方式直接影响求解效率。通过优化和简化逻辑规则，可以减少求解时的计算量，提升求解速度。例如，某些规则可以通过合并或重构来减少搜索空间的大小，从而降低求解复杂度。在底层实现中，采用更高效的数据结构来存储和管理逻辑规则及变量，可能会减少内存占用并加快搜索速度。例如，利用哈希表可以快速查找和更新变量的约束条件，从而提升整体性能。

结论：

实验成功解决了斑马问题，验证了 kanren 在逻辑推理中的强大能力。通过合理的形式化规则和优化求解过程，能够高效处理复杂的约束满足问题。同时，实验过程中遇到的困难和改进方向也为我提供了宝贵的经验。未来，我将进一步优化逻辑规则的设计，探索更高效的数据结构和求解方法，以提升逻辑编程在实际应用中的性能和效率。