程序报告

学号：2212046　　姓名：王昱

一、问题重述

斑马问题：5 个不同国家且工作各不相同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，每个人都有自己养的不同宠物，喜欢喝不同的饮料。
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？

==========

对问题的理解：

- 斑马问题为我们提供了十四个条件，每个条件看作是一个约束或关系，通过这些约束可以找到唯一符合这些条件的一种情况，这种情况便是我们所要求解的情况。
- 问题中共有五个“房子”，每个房子有五种不同类型的数据。而每个房子可以视为一个逻辑变量，每个逻辑变量又含有五个逻辑变量，分别代表国家，工作，饮料，宠物，颜色。
- 通过问题中给出的这五组逻辑变量的关系，对所有情况进行剔除，最终得出唯一一种满足约束条件，符合逻辑关系的答案。

二、设计思想

kanren 是一个 Python 中的逻辑编程库，用于自动化推理和约束求解。 它允许用户使用关系和规则来表示问题，并使用不同的搜索算法来查找满足约束条件的解决方案。本次实验的斑马问题使用逻辑编程的方法解决，这种方法在一些特定的问题领域中非常有效，比如解谜题、推理问题等。通过定义规则和约束，我们可以使用逻辑编程来找到问题的解，而无需显式地编写具体的算法。

==========

设计步骤：

第一步：自定义需要用到函数 left,right,next。

- left 函数的构造思想很巧妙，通过切片[1:]来使其错位一个房子，然后用 zip 打包保存到 groups，使用 kanren 库的 membero 判断(x,y)是否包含在 groups 中，从而判断是否相邻。
- right 函数与 left 相反，调换参数位置即可。
- next 就是在左边或者在右边，通过 conda 来进行一个或的联系即可。

```
def left(x,y,units):
    groups=zip(units,units[1:])
def right(x, y, units):
    return left(y, x, units)
def next(x, y, units):
    return conde([left(x, y, units)], [right(x, y, units)])
```

==========

第二步：定义 Agent 类

- 进行智能体的初始化，将 units 设计为逻辑变量，units 包含五个逻辑变量，其中单个

unit 变量指代一座房子的信息（国家，工作，饮料，宠物，颜色）。之后将 rule_zebraproblem 与 solution 初始化为 none，分别用来定义约束的规则和储存结果。

```
def __init__(self):
    self.units = var()
    self.rules_zebraproblem = None
    self.solutions = None   # 存储结果
```

■ **定义规则函数，通过添加逻辑规则，使用 kanren 中的 lall 来进行逻辑约束。lall 函数接受一个由约束条件组成的列表作为参数，然后返回一个新的约束条件，该约束条件要求列表中的所有约束条件都成立。如果列表中任何一个约束条件不满足，那么整个约束条件将被视为不成立。**

本次实验使用了如下规则：

（1）eq 是相等关系，即前面的逻辑变量与后面的相等。下面的例子说么 units 是一个包含五个逻辑变量的逻辑变量。

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

（2）membero 函数检查一个元素是否存在于一个逻辑变量的可能取值集合中。下面的代码意思是 unit 中有一个西班牙人养狗。

```
(membero, ('西班牙人', var(), var(), '狗', var()), self.units)
```

（3）next,right,left 是房子的位置关系，是自定义的函数。下面的例子表示挪威人的房子在最左边。这种方法将含有挪威人的逻辑变量放到最左边，恰当的表示了挪威人房子的位置。

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

完整规则如下：

```
lall(
(eq, (var(), var(), var(), var(), var()), self.units),
(membero, (var(), var(), var(), '斑马', var()), self.units),
(membero, (var(), var(), '矿泉水', var(), var()), self.units),
(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),
(eq, (('挪威人', var(), var(), var(), var()), var(), var(), var(), var()), self.units),
(right, (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),
(membero, (var(), '外交官', var(), var(), '黄色'), self.units),
(eq, (var(), var(), (var(), var(), '牛奶', var(), var()), var(), var(), self.units)),
(membero, (var(), var(), '咖啡', var(), '绿色'), self.units),
(next, ('挪威人', var(), var(), var(), var()), (var(), var(), var(), var(), '蓝色'),
self.units),
(membero, (var(), '小提琴家', '橘子汁', var(), var()), self.units),
(next, (var(), var(), var(), '狐狸', var()), (var(), '医生', var(), var(), var()), self.units),
(next, (var(), var(), var(), '马', var()), (var(), '外交官', var(), var(), var()), self.units)
)
```

■ **定义 solve 函数**

solve 函数通过 run 函数对 units 用定义好的规则进行约束，从而得出 solutions 并且作

为函数的返回值。

```
self.solutions = run(0, self.units, self.rules_zebraproblem)
```

==========

第三步：调用 Agent 类输出结果

先定义一个 Agent 类的对象，调用 solve 函数得出返回值。

```
agent = Agent()
solutions = agent.solve()
```

提取解释器的结果，通过遍历 solutions 按照规定的格式输出题目要求的结果。

三、代码内容

主要的代码内容如下：

```
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
def left(x, y, units):
    groups = zip(units, units[1:])
    return membero((x, y), groups)
def right(x, y, units):
    return left(y, x, units)
def next(x, y, units):
    return conde([left(x, y, units)], [right(x, y, units)])
```

==========

```
class Agent:
    def __init__(self):
        self.units = var()
        self.rules_zebraproblem = None
        self.solutions = None  # 存储结果

    def define_rules(self):
        self.rules_zebraproblem = lall(
            (eq, (var(), var(), var(), var(), var()), self.units),
            (membero, (var(), var(), var(), '斑马', var()), self.units),
            (membero, (var(), var(), '矿泉水', var(), var()), self.units),
            (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),
            (eq, (('挪威人', var(), var(), var(), var()), var(), var(), var(), var()), self.units),
            (right, (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),
            (membero, (var(), '外交官', var(), var(), '黄色'), self.units),
            (eq, (var(), var(), (var(), var(), '牛奶', var(), var()), var(), var(), self.units)),
            (membero, (var(), var(), '咖啡', var(), '绿色'), self.units),
            (next, ('挪威人', var(), var(), var(), var()), (var(), var(), var(), var(), '蓝色'),
self.units),
```

```
                (membero, (var(), '小提琴家', '橘子汁', var(), var()), self.units),
                (next, (var(), var(), var(), '狐狸', var()), (var(), '医生', var(), var(), var()), self.units),
                (next, (var(), var(), var(), '马', var()), (var(), '外交官', var(), var(), var()), self.units)
            )

    def solve(self):
        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)
```

四、实验结果

输出结果：

```
D:\python2\pythonProject5\Scripts\python.exe D:\pythonProject5\main.py

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')
('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')

Process finished with exit code 0
```

==========

提交结果：

五、总结

- 本次实验收获很大，了解了 kanren 包的使用规则，学会了通过逻辑编程来解决一些较为复杂的逻辑推理问题。
- 定义规则的时候，如何表示房子的位置是我实验中遇到的困难，也是是重点。在经过思考后自定义 next,right,left 函数来表示位置关系。并且将含有某种信息的逻辑变量放到最左边来表示这个房子在最左边，这种方法十分巧妙，让我深有所思。
- 本次实验达到了我的目的预期，可以在较少的时间内正确地输出相应的结果。但也有许多不足之处，比如当规则较多的时候是否还能在较少的时间内输出，是否可以简化设计，通过自定义函数来提升程序运行的速度与性能。这些都是我思考以及改进的方向。