2021-2022 年度春夏学期浙江大学人工智能本科专业《人工智能基础》课程实训技术报告

姓名：秦嘉俊　学号：3210106182

学院（系）专业：竺可桢学院，计算机科学与技术（图灵班）

1　问题概述

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，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

除了输出问题的答案外，还要输出正确的五条匹配信息。

2 算法描述

所谓“斑马难题”是一道基于已给条件，通过逻辑推断来判断的逻辑问题。[kanren] 是 Python 的一个逻辑编程包，凭借 kanren，我们可以将 Python 应用于逻辑编程。
其中我们会用到以下函数

- 等价关系表达式
  `eq(var(), value) / eq(var(), var())` 意为 `var` 与 `var` 等价或 `var` 与 `value` 等价
- 成员关系表达式
  `membero(var(), list / tuple)` 意为 `var` 是 `list/tuple` 的成员
- 逻辑和/或的目标构造函数
  `conde((rules, rules))` 意为两个规则的逻辑和,`conde([rules], [rules]))` 意为两个规则的逻辑或。
- 定义规则集合
  `lall(rules, [rules, ...])` 意为定义一系列规则

在这个问题中，我们利用 `lall` 包定义规则集合，而这里的规则就是题目中的约束条件（如英国人住在红色的房子里）具体来说，我们要做的就是将其全部翻译为逻辑表达式。

3 代码实现

3.1 翻译逻辑表达式

`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)，对应公式为：`(eq, (var(), var(), var(), var(), var()), self.units)`，将对应的逻辑规则约束填充进去即可。

```
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),
   # 英国人住在红色的房子里
   (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),
    # 意大利人喜欢喝茶
    (eq, (('挪威人', var(), var(), var(), var()), var(), var(), var(), var()),
        self.units),
    # 挪威人在左边第一个房子里
    (right, (var(), var(), var(), var(), '绿色'), (var(), var(), var(), var(),
        '白色'), self.units),
    # 绿房子在白房子右边
    (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),
    # 挪威人住在蓝色的房子旁边
    (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(), var(), var(), '狐狸', var()), (var(), '医生', var(), var(),
        var()), self.units),
    # 养狐狸的人所住的房子与医生的房子相邻
    (next, (var(), var(), var(), '马', var()), (var(), '外交官', var(), var(),
        var()), self.units)
    # 养马的人所住的房子与外交官的房子相邻
)
```

3.2 左（右）邻近规则

这里有一些规则比较特殊的邻近规则，以左邻近为例：

如果 p 在 q 的左边，那么 `(p,q)` 就应该是类似 `(list, list[1:])` 的形式。这里我们会用到 `zip()` 函数，`zip()`函数用于将可迭代的对象作为参数，将对象中对应的元素打包成一个个元组，然后返回由这些元组组成的列表。

而邻近函数 `next` 则可以直接利用 `left` 和 `right` 规则，并将他们用 `conde` 逻辑或

起来，即 p 或者在 q 左边，或者在 q 右边。

```
def left(p, q, list): # p is left of q
    return membero((p, q), zip(list, list[1:]))

def right(p, q, list):
    return left(q, p, list)

def next(p, q, list):
    return conde([left(p, q, list)], [right(p, q, list)])
```

3.3 输出结果

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

其中 `Agent` 是我们的推理智能体，`agent.solve()` 则是负责定义规则并通过 `run` 来实现逻辑推理并将答案返回。这里在推理出各个人的相关信息后，我们还要回答最开始的两个问题，即哪所房子里的人养斑马，哪所房子里的人喜欢喝矿泉水吗？因此我们枚举每个人判断他们的信息是否符合，如果符合就记录下来，然后输出答案。最后我们将每个人的信息依次输出即可。

4 算法实验结果与分析

对此题的答案，我们可以用人脑推理得到答案：(推理过程略)

挪威人	意大利人	英国人	西班牙人	日本人
外交官	医生	摄影师	小提琴家	油漆工
矿泉水	茶	牛奶	橘子汁	咖啡
狐狸	马	蜗牛	狗	斑马
黄色	蓝色	红色	白色	绿色

而程序得到的答案如下：

```
hobbitqia@LAPTOP-EBSFK1UV:~/ai$ python3 2.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')
```

图 1: 斑马问题答案

由此看出，程序的结果正确，算法符合预期。

5 算法进一步研究展望

本次算法实现中我们主要利用了 `karren` 这个逻辑编程包，但遗憾的是 `karren` 目前已经停止更新，相对来说比较过时。在网上的搜索中我发现有一个新的工具可以为我们所用：**Google OR-Tools**

> **Google Optimization Tools (a.k.a., OR-Tools)** [1] is an open-source, fast and portable software suite for solving combinatorial optimization problems. The suite contains:
>
> - A constraint programming solver;
> - Two linear programming solvers;
> - Wrappers around commercial and other open source solvers, including mixed integer solvers;
> - Bin packing and knapsack algorithms;
> - Algorithms for the Traveling Salesman Problem and Vehicle Routing Problem;
> - Graph algorithms (shortest paths, min cost flow, max flow, linear sum assignment).

而且我发现，在 **Google OR-Tools** 的 GitHub 仓库示例里恰好有斑马问题的样例程序，这为我们解决斑马问题提供了新的工具和思路，具体实现见仓库代码，这里就不再重复。

[1] 以下介绍摘自 **Google OR-Tools** 的GitHub 仓库介绍：

6 心得与收获

我们无论是八皇后问题，还是这次的斑马问题，都是使用的逻辑编程的方式。这次的斑马问题让我对其本质有了更深的理解。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，而非设置步骤来解决问题。过程是“事实 + 规则 = 结果”。而逻辑编程是基于逻辑学中的演绎推理来进行的。通过实验，我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以及规则编写代码，解决逻辑约束问题 (CLP)。

演绎推理（Deductive Reasoning）是由一般到特殊的推理方法。与“归纳法”相对。推论前提与结论之间的联系是必然的，是一种确实性推理。运用此法研究问题，首先要正确掌握作为指导思想或依据的一般原理、原则；其次要全面了解所要研究的课题、问题的实际情况和特殊性；然后才能推导出一般原理用于特定事物的结论。演绎推理的形式有三段论、假言推理和选言推理等。

在实际编程，比较难处理的是三个表示关系的条件。这里就需要引入两个自定义的函数：左邻近规则 `left()`, 和定义邻近规则 `next()`。这样就可以表达出甲与乙相邻或者甲在乙左边这样的条件。最后，如果缺乏某个条件，程序就会报错，不能得到正确的结果。本次实验主要学习了逻辑编程，这和一般的编程有不少区别，它设定答案须符合的规则来解决问题，而非设定步骤来解决问题，再次深化了“事实 + 规则 = 结果”这个法则。这对我有很大启发，期待课程的后续实验能给我更多收获！