## 实验二 产生式系统推理

**一 实验目的**

1 熟悉和掌握产生式系统的构成和运行机制；

2 掌握基于规则推理的基本方法和技术，掌握正确的正向推理和逆向推理方法；

3 熟悉在具体问题中如何实现正向推理和逆向推理的求解流程。

**二 实验预习内容**

1 基于产生式规则的机器推理

**三 实验内容**

以动物识别系统为例，用选定的编程语言建造规则库和综合数据库，并能对它们进行增加、删除和修改操作，开发能进行正确的正向推理或反向推理的推理机。

1 动物分类规则集（Ⅰ）

（1）若某动物有奶，则它是哺乳动物。

（2）若某动物有毛发，则它是哺乳动物。

（3）若某动物有羽毛，则它是鸟。

（4）若某动物会飞且生蛋，则它是鸟。

（5）若某动物是哺乳动物且有爪且有犬齿且目盯前方，则它是食肉动物。

（6）若某动物是哺乳动物且吃肉，则它是食肉动物。

（7）若某动物是哺乳动物且有蹄，则它是有蹄动物。

（8）若某动物是有蹄动物且反刍食物，则它是偶蹄动物。

（9）若某动物是食肉动物且黄褐色且有黑色条纹，则它是老虎。

（10）若某动物是食肉动物且黄褐色且有黑色斑点，则它是金钱豹。

（11）若某动物是有蹄动物且长腿且长脖子且黄褐色且有暗斑点，则它是长颈鹿。

（12）若某动物是有蹄动物且白色且有黑色条纹，则它是斑马。

（13）若某动物是鸟且不会飞且长腿且长脖子且黑白色，则它是驼鸟。

（14）若某动物是鸟且不会飞且会游泳且黑白色，则它是企鹅。

（15）若某动物是鸟且善飞且不怕风浪，则它是海燕。

2 问题描述

由上述动物识别规则组成规则库，推理机分别采用正向推理算法或反向推理算法，实现

对动物的查询。

如给出初始事实：

F1：某动物有毛发

F2：吃肉

F3：黄褐色

F4：有黑色条纹

目标条件为：该动物是什么？

下面是该规则集所形成的（部分）推理网络：

老虎

黄褐色

有黑色条纹

食肉动物

哺乳动物

有毛发

有奶

吃肉

有爪

有犬齿

目盯前方

金钱豹

有黑色斑点

长颈鹿

有蹄动物

有蹄

长腿

长脖子

有暗斑点

图7 动物识别系统部分推理网络

3 规则库扩充

在上述规则集(Ⅰ)基础上增加以下规则集(Ⅱ)：

（1）兔子：有毛发，有奶，善跳跃，唇裂；

（2）猫：有毛发，有奶，善捕鼠，脚有肉垫；

（3）犀牛：有毛发，有奶，鼻子上有角，褐色，皮糙肉后，皮糙肉厚，有蹄；

（4）熊猫：有毛发，有奶，黑眼圈，四肢短小；

（5）鹦鹉：鸟类，上嘴鹰钩，会模仿人说话；

（6）鸭子：鸟类，腿短，嘴扁平，善潜水游泳；

（7）鹰：鸟类，上嘴鹰钩，有爪，吃肉；

（8）鸭子：有羽毛，卵生，善游泳，嘴扁平，腿短；

（9）鹅：有羽毛，卵生，善潜水游泳，白色或黑色，颈长，嘴大，腿长，颈部有肉只凸起；

（10）鸦：有羽毛，卵生，黑色，嘴大；

（11）鹰：有羽毛，卵生，有爪，吃肉，上嘴鹰钩；

（12）鹦鹉：有羽毛，卵生，上嘴鹰钩，能模仿人说话；

（13）青蛙：卵生，生活在水中，生活在陆地，有皮肤呼吸，用肺呼吸，皮肤光滑，吃昆虫，会变色；

（14）蝾螈：卵生，生活在水中，生活在陆地，有皮肤呼吸，用肺呼吸，吃昆虫，皮肤粗糙，四肢扁，背部黑色；

（15）蟾蜍：卵生，生活在水中，生活在陆地，有皮肤呼吸，用肺呼吸，吃昆虫，皮肤粗糙；

（16）比目鱼：用鳃呼吸，身体有鳍，生活在海洋中，身体扁平，两眼在头部同侧；

（17）鲫鱼：用鳃呼吸，身体有鳍，生活在淡水中，身体扁平，头高尾部窄；

（18）蛇：生活在陆地，用肺呼吸，胎生，身体有鳞或甲，身体圆而细长，吃小动物；

（19）壁虎：生活在陆地，用肺呼吸，胎生，身体有鳞或甲，有四肢，尾巴细长易断，吃昆虫；

（20）乌龟：生活在陆地，用肺呼吸，胎生，身体有鳞或甲，身体圆而扁，有坚硬的壳；

（21）玳瑁：生活在陆地，用肺呼吸，胎生，身体有鳞或甲，壳为黄褐色，皮肤光滑，有黑斑；

（22）鳄鱼：生活在陆地，用肺呼吸，胎生，身体有鳞或甲，有四肢，善游泳，皮硬黑褐色。

要求在动物分类规则集(Ⅰ)的基础上添加上述22条知识，共构成29种动物的知识库系统，对原有动物分类系统进行扩充和修改。

**四 实验要求**

1 以产生式推理模式为基础，实现小型动物分类系统，推理方法可以采用正向推理或反向推理；

2 要求表示规则的语言必须能体现出规则前提和结论的对应关系，必须能体现出前提和结论中的逻辑关系；

3 要求能对规则库进行动态地增加、删除和修改操作；

4要求用界面显示要查询的初始事实、推理方法、推理中用到的规则和结论。

**五 实验检查要求**

1 界面显示要求

（1）有若干选择动物特征的选择列表；

（2）表现判断动物时，使用了哪些规则；

（3）显示规则的调用次序；

（4）显示最后的结果，包含动物能识别出来和动物不能识别出来两种情况；

（5）至少检查两个例子来检验推理系统的正确与全面性。

金钱豹：



老虎

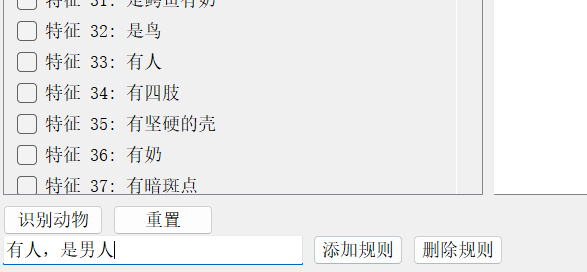


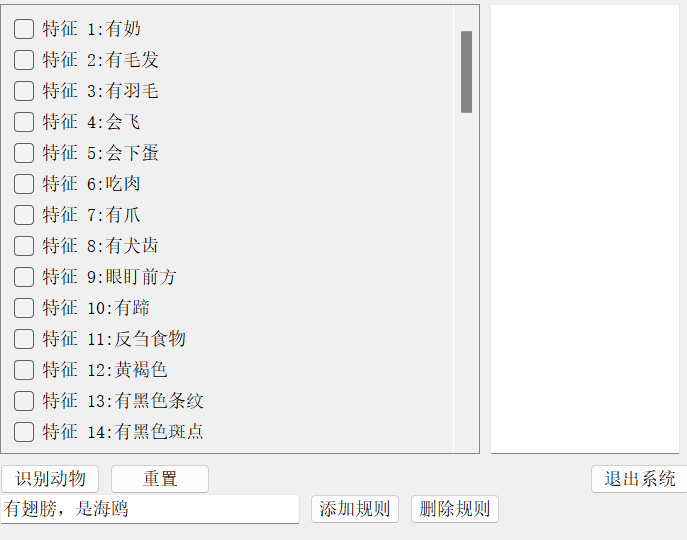
无法识别情况



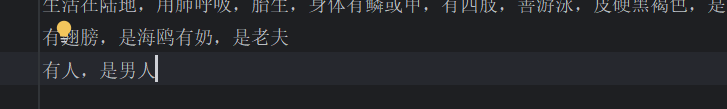
1. 增删

增加功能：

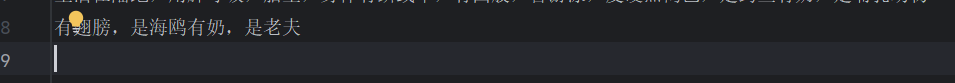


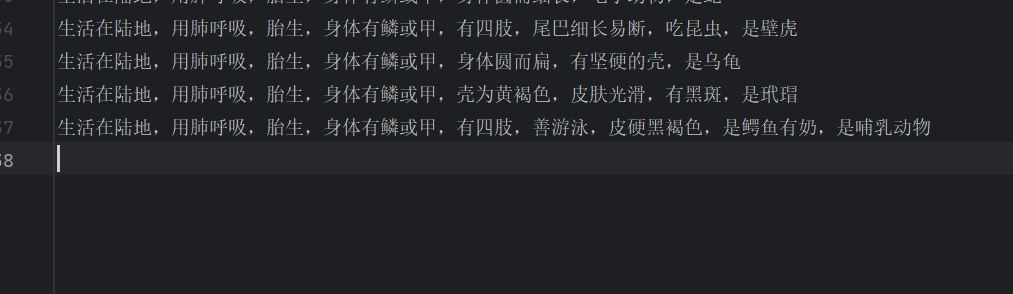


删除规则：



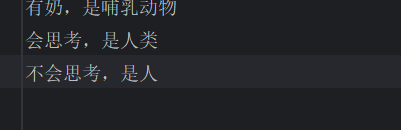






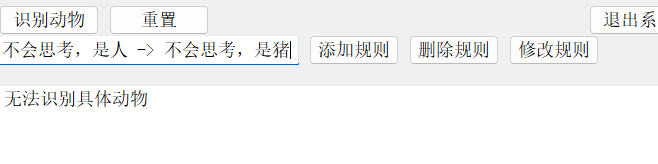
修改功能 （我们也可以通过先删除原先功能，再添加功能实现。）

修改这个规则判断





进行修改

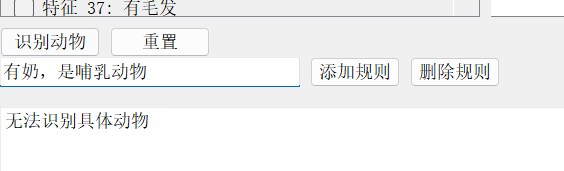


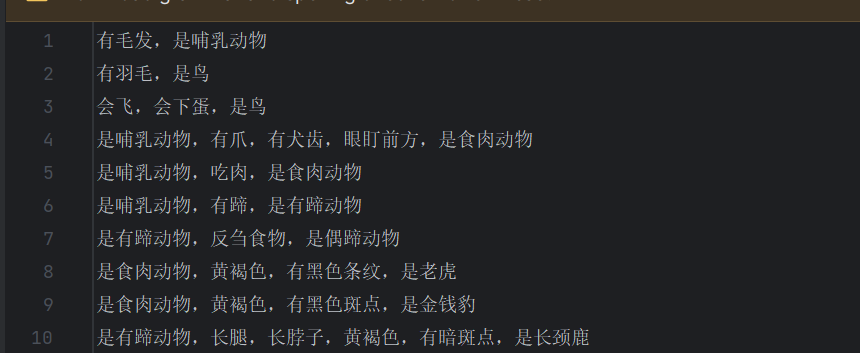
修改后：



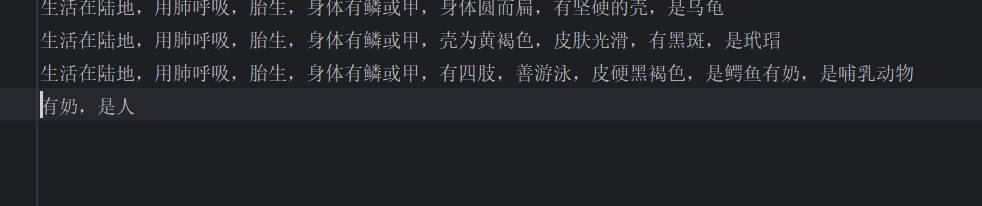
我们也可以通过先删除原先功能，再添加功能实现

先删除





然后再添加



2 代码要求

要求学生展示如何表征事实和特征的知识、所有规则的代码、控制策略的代码。

3 讲解要求

要求学生讲解采用的推理算法，规则的程序表示方法，规则库如何动态修改等，并说明程序运行效果。

1. **实验代码分析**

Find\_data（）函数

|  |
| --- |
| def find\_data(process\_data\_list, rules\_applied=None):  # 初始化已应用规则列表，如果未提供则设置为空列表  if rules\_applied is None:  rules\_applied = []   # 初始化一个列表来存储过程中新发现的特征  new\_features = []  # 初始化一个标记，用于指示是否找到了最终结果  final\_result\_found = False   # 遍历所有规则。data\_process\_list 包含规则的前提条件，data\_result\_list 包含对应的结论  for epoch, data\_process in enumerate(data\_process\_list):  # 检查当前规则的所有前提条件是否都在已知特征列表中  if all(feature in process\_data\_list for feature in data\_process):  # 从结果列表中获取当前规则的结果部分  result = data\_result\_list[epoch].lstrip('是')   # 构建表示当前规则的字符串  applied\_rule = '，'.join(data\_process) + "，" + result   # 如果该规则尚未被应用，将其添加到已应用规则列表中  if applied\_rule not in rules\_applied:  rules\_applied.append(applied\_rule)   # 检查当前规则的结果是否是最终结果（在结果列表中）  if result in result\_list:  # 如果是最终结果，设置标记并退出循环  final\_result\_found = True  break  # 如果结果是一个新的特征，且尚未出现在已知特征列表中，将其添加到新特征列表中  elif result not in process\_data\_list and result not in new\_features:  new\_features.append(result)   # 检查是否找到最终结果或发现新特征  if final\_result\_found:  # 如果找到最终结果，返回 1 和已应用的规则列表  return 1, rules\_applied  elif new\_features:  # 如果发现新特征，将其添加到已知特征列表中，然后递归调用 find\_data 函数  for feature in new\_features:  process\_data\_list.append(feature)  return find\_data(process\_data\_list, rules\_applied)   # 如果既没有找到最终结果也没有发现新特征，返回 0 和已应用的规则列表  return 0, rules\_applied |

1. 函数接受两个参数 `process\_data\_list` 和 `rules\_applied`（可选参数，默认值为None）。

2. 首先，通过条件判断语句检查 `rules\_applied` 是否为None。如果是，将其初始化为空列表。

3. 接下来，初始化一个空列表 `new\_features` 用于存储在过程中新发现的特征。

4. 初始化一个布尔变量 `final\_result\_found`，用于指示是否找到了最终结果，初始值为False。

5. 使用 `enumerate` 函数遍历 `data\_process\_list` 列表，并在每个迭代步骤中获取索引和对应的元素。

6. 在循环中，使用 `all()` 函数和生成器表达式检查当前规则的所有前提条件是否都存在于 `process\_data\_list` 中。

7. 如果所有前提条件都满足，执行以下操作：

- 从 `data\_result\_list` 列表中获取当前规则的结果部分，并去除结果字符串开头的"是"。

- 构建表示当前规则的字符串 `applied\_rule`，将前提条件和结果用逗号连接起来。

- 如果该规则尚未被应用，将其添加到 `rules\_applied` 列表中。

- 检查当前规则的结果是否是最终结果（在 `result\_list` 中）。

- 如果是最终结果，将 `final\_result\_found` 设置为True，并退出循环。

- 如果结果是一个新的特征，并且尚未出现在 `process\_data\_list` 和 `new\_features` 列表中，将其添加到 `new\_features` 列表中。

8. 循环结束后，检查是否找到了最终结果或者发现了新特征。

- 如果找到最终结果，返回一个元组 `(1, rules\_applied)`，其中1表示找到最终结果，`rules\_applied` 是已应用的规则列表。

- 如果发现新特征，将新特征添加到 `process\_data\_list` 中，然后递归调用 `find\_data` 函数，传递更新后的 `process\_data\_list` 和 `rules\_applied`。

- 如果既没有找到最终结果，也没有发现新特征，返回一个元组 `(0, rules\_applied)`，其中0表示没有找到最终结果，`rules\_applied` 是已应用的规则列表。

函数通过遍历规则列表和结果列表，根据已知特征和规则的前提条件，逐步推导出新的特征并应用规则，直到找到最终结果或无法推导出新特征退出。

**七、实验总结**

1.通过这次实验，我对产生式系统的构成和运行机制有了更深入的理解。我学到了如何设计和实现一个基于规则的系统，这不仅涉及到规则的编写，还包括如何有效地管理这些规则（增加、删除和修改）。设计一个用户友好的界面来展示推理过程和结果，是我在这次实验中的一个新挑战。这不仅增强了我的编程技能，还提升了我在用户体验方面的认识。

2.在实验中，我实际体验了正向推理和逆向推理的应用，理解了它们的区别及适用场景。正向推理在确定事实基础上推导结论的场景中更为有效，而逆向推理则适用于目标驱动的情境。

3.动态规则管理的重要性：实验让我认识到，动态管理规则库对于保持系统的灵活性和适应性是至关重要的。通过实现增加、删除和修改规则的功能，我学会了如何更新和维护一个动态变化的知识库。