|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 课程名称 | 人工智能基础 | | | | |
| 实验名称 | 基于产生式的专家识别系统 | | | | |
| 实验时间 | 12月13日 | | 实验地点 | 12教661-1 | |
| 学 院 | 信息科学与工程学院 | | 专 业 | 计算机科学与技术 | |
| 姓 名 | 占昌明 | 班 级 | 计科2202 | 学 号 | 202231222056 |
| 同组实验者  姓名 | 刘俊伸 卢鹏宇 | | | | |
| 实验成绩 |  | | 指导教师  （签字） |  | |
| **实验报告内容基本要求参考格式**  一、实验目的  二、实验环境  三、实验步骤/过程  四、实验结果  五、实验分析及反馈 | | | | | |
| 1. 实验目的   理解和掌握产生式知识表示方法及产生式系统的基本过程，能够利用编程技术建立一个基于产生式知识表示的简单的专家系统。（可采用前向推理）建立一个动物识别系统的规则库，用以识别虎、豹、斑马、长颈鹿、企鹅、鸵鸟、信天翁等7种动物 （至少7种动物，规则可增加）。   1. 实验环境   上机实验，IntelliJ IDEA 2023.2.1   1. 实验步骤/过程   1、实验分工：  占昌明：界面设计，用户信息提示。  刘俊伸：推理机设计，解释机  卢鹏宇：规则类设计，知识库，综合数据库设计 | | | | | |

|  |
| --- |
| 2、主要函数设计：  （1）规则类设计（主要部分）：用集合来存放一条规则的条件，同时设置结论变量，整型变量表示条件的个数，以及一个布尔值来判断是否为最终条件。下面为主要代码：  public class rule\_base {//规则类   private ArrayList<String> situ=new ArrayList<>();//用集合来存放一条规则的条件   private String clusion;//结论   boolean use; //是否被使用   int size;//条件的条数   boolean end; //判断该规则是否能得出最终结论   public rule\_base() {//构造函数，初始化为未被使用，且不是最终结论  use=false;  size=0;  end=false;  }  （2） 综合数据库设计（主要部分）：用集合来存放一条规则的条件，用一个String变量存放最终结论，用一个布尔值表示得出的结论是否为最终结论，同时使用一个String存放临时结果，用来存放非最终结论最后种类，比如鸟。最后设置一个集合存放已匹配到的规则序号。  public class rule\_temp {  //综合数据库，采用集合存放事实  private ArrayList<String> knowledge=new ArrayList<>();  //最终结论  private String result;  private boolean isend;//判断则出的结论是否为最终结论  private String tempResult;//临时结果，用来存放非最终结论最后种类，比如鸟。  private ArrayList<Integer> num=new ArrayList<>();//设置一个集合存放已匹配到的规则序号  public rule\_temp()//构造函数，初始化数据库  {  knowledge.add("有犀利牙齿");  knowledge.add("眼看前方");  knowledge.add("有爪");  knowledge.add("有毛发");  knowledge.add("有黄褐色");  knowledge.add("有黑色条纹");  isend=false;  } }  （3）知识库设计：用集合来存储所有的规则，之后设置默认构造函数，创建一个规则类，为该规则类添加条件，每次添加一条条件，相应的size增加1，最后设置规则的结论  //知识库 public class rule\_total {  //设置规格集合  private ArrayList<rule\_base> rules=new ArrayList<>();//用集合来存储所有的规则  //构造函数，设置默认规则。  public rule\_total()//构造函数，初始化知识库  {  rule\_base num1=new rule\_base();//创建一个规则类  num1.getSitu().add("有毛发");//为该规则类添加条件  num1.setSize(num1.getSize()+1);//每次添加一条条件，相应的size增加1，  num1.setClusion("哺乳动物");//设置规则的结论  rules.add(num1);  . . .  . . .  . . .  rule\_base num9=new rule\_base(); num9.getSitu().add("哺乳动物"); num9.setSize(num9.getSize()+1); num9.getSitu().add("食肉动物"); num9.setSize(num9.getSize()+1); num9.getSitu().add("有黄褐色"); num9.setSize(num9.getSize()+1); num9.getSitu().add("有暗斑点"); num9.setSize(num9.getSize()+1); num9.setClusion("豹"); num9.setEnd(true);//当能推出豹子的时候，设置该规则为最终结论，下面类似。 rules.add(num9);  （4）用户提示函数：根据已有结论进行用户提示，它会根据已有条件找到能推出最终结论最少的那条规则，然后提示该规则中没有的条件。设计的主要思路是：先从所有的规则中找到条件中包含临时结论的规则，之后比较这些规则距离能推出最终结论还差多少条条件，找到缺少最少的那条规则，将其在知识库中的位置返回。  //提示，根据已有结论进行用户提示，它会根据已有条件找到能推出最终结论最少的那条规则，然后提示该规则中，没有的条件。  public int tip(rule\_total ruleTotal,rule\_temp ruleTemp) {  int num=0;//已有条件的个数  int max=999999;//缺少条件最少的那条规则的个数  int total=0;  int temp;  for(int i=0;i<ruleTotal.getRules().size();i++)//遍历所有的规则  {  num=0;  boolean haha=false;  for(int j=0;j<ruleTotal.getRules().get(i).size;j++)//对其中一条规则，遍历所有的条件，看能否找到包含临时结论的条件  {  if(ruleTotal.getRules().get(i).getSitu().get(j).equals(ruleTemp.getTempResult()))  {  haha=true;//如果找到，将其设置为true  break;//同时停止遍历  }  }  if(haha==true) {//如果能找到，对该规则所有的条件进行遍历，直至找到缺少条件最少的那条规则  if (ruleTotal.getRules().get(i).isEnd()) {//判断是否为最终结论  for (int j = 0; j < ruleTotal.getRules().get(i).getSitu().size(); j++) {//对该规则的每一条条件进行遍历  for (int k = 0; k < ruleTemp.getKnowledge().size(); k++) {//在数据库中寻找这条规则，  if (ruleTemp.getKnowledge().get(k).equals(ruleTotal.getRules().get(i).getSitu().get(j))) {  num++;//如果能找到num++，表示已有条件的个数  }  }  }  temp = ruleTotal.getRules().get(i).getSitu().size() - num;//缺少条件的个数  if (temp < max) {//与max比较，获取缺少条件最少的那条规则  max = temp;  total = i;//记录这条规则的位置。  }  }  }  }  return total;//返回缺少条件最少的那条规则在知识库的位置 }  （5）推理机设计：对每一条规则进行判定，如果某条规则的的一条条件在综合数据库找不到，则直接对下一条规则进行判定。如果有一条规则的所有条件都满足，则将该规则设置为已使用，将其结论纳入到综合数据库中，最后标记其规则的的位置。同时分两种情况：如果该规则为最终结论，则停止循环，并将结论赋值给数据库的最终结论，如果不是最终结论，将结论赋值给数据库的临时结论，并将结论添加到数据库当中，继续下一个循环。下面为详细代码：  //推理机 //原理：对每一条规则进行判定，如果某条规则的的一条条件在综合数据库找不到，则直接对下一条规则进行判定。 //如果有一条规则的所有条件都满足，则将该规则设置为已使用，将其结论纳入到综合数据库中，最后标记其条数的位置。 public void getResult(rule\_total ruleTotal,rule\_temp ruleTemp) {  String temp;  int i=0,j=0,k=0;  for( i=0;i<ruleTotal.getRules().size();i++)//设置循环，遍历所有的规则  {  if(ruleTotal.getRules().get(i).isUse())//如果该规则已被使用，则直接对下一条规则进行遍历  {  continue;  }  else  {  for(j=0;j<ruleTotal.getRules().get(i).getSize();j++)//如果该规则未被使用，对该规则的条件进行遍历，  //如果其中某一条条件无法再数据库中找到，则直接进行下一条规则的判定  {  for(k=0;k<ruleTemp.getKnowledge().size();k++)//从数据库中寻找与条件相匹配的数据  {  if(ruleTotal.getRules().get(i).getSitu().get(j).equals(ruleTemp.getKnowledge().get(k)))  {  break;  }  }  if(k>=ruleTemp.getKnowledge().size())//判断是否在数据库中找到该条件  {  break;  }    }  if(j>=ruleTotal.getRules().get(i).getSize())//判断该规则的全部条件是否被全部找到  {  ruleTotal.getRules().get(i).setUse(true);//将该规则设置为已使用  ruleTemp.getNum().add(i);//将该规则的在知识库的位置存放到数据库中   if(ruleTotal.getRules().get(i).isEnd()) //如果该规则得出的结论为最终结论，则不再进行下条规则的判定  {  ruleTemp.setResult(ruleTotal.getRules().get(i).getClusion());//将该规则的结论存放到数据库的结论当中  System.out.println(ruleTotal.getRules().get(i).getClusion());  break;   }  //该规则得出的结论不为最终结论，将该结论纳入到综合数据库中，并重新进行所有规格的判定，  //已经使用的规则不再进行判定。  else//如果不是最终结论  {    ruleTemp.setTempResult(ruleTotal.getRules().get(i).getClusion());//将该规则的结论存放到数据库临时结论当中  ruleTemp.getKnowledge().add(ruleTotal.getRules().get(i).getClusion());//并将该规则的结论存放到数据库的事实当中  i=-1;//回到第一条规则，重新进行规则的遍历  }  }  }  } }  （6）解释机：由于之前我们将所有已经匹配到的规则的位置存放到了数据库中的整形集合当中，因此我们把所有的位置遍历出来，然后将这些规则的条件和规则进行输出。函数从 ruleTemp 数据库的 num 集合中获取每个使用过的规则的序号。这些序号是之前推理过程中记录下来的。对于每个使用过的规则，函数通过序号在 ruleTotal 的 rules 集合中找到相应的规则对象。然后，它遍历这个规则对象的 situ 集合，这里面存储了规则的所有条件。对于每个条件，函数将其打印出来。在打印完所有条件后，它还会输出该规则的结论。  //解释机 //将使用过的规则全部进行输出 public void inputProcess(rule\_total ruleTotal,rule\_temp ruleTemp) {  for(int i=0;i<ruleTemp.getNum().size();i++)//遍历全部使用过的规则  {//由于之前我们将使用过的规则的序号全部存到了数据库中，所以直接遍历数据库中的num集合即可获取规则的位置  int c;  c=ruleTemp.getNum().get(i);  for(int j=0;j<ruleTotal.getRules().get(c).size;j++)//遍历每一条规则的所有条件  {   System.out.print(ruleTotal.getRules().get(c).getSitu().get(j)+" ");  }  System.out.print(ruleTotal.getRules().get(c).getClusion());//输出该规则的结论  System.out.println();   } }  （7）复位清空函数：将系统恢复到初始状态，重置规则的使用状态、清空综合数据库，并移除所有已得出的结论。重置规则使用状态：将知识库中所有规则的使用状态设置为未使用。这是为了让系统能够重新评估这些规则，而不是依赖之前的推理结果。清空综合数据库：移除综合数据库中的所有元素。综合数据库可能包含了在推理过程中收集的信息或中间结果，清空这些信息使得系统可以从头开始新的推理过程。移除结论：将已经得出的结论清空。这包括将数据库中的结论 (ruleTemp.setResult(null)) 和临时结论(ruleTemp.setTempResult(null)) 设置为 null。  //复位功能，将标记使用过的结论全部设置为未使用，将综合而数据库清空，将标记过的规则位置设置为清空，将已经得出的结论清空 public void updataUse(rule\_total ruleTotal,rule\_temp ruleTemp) {  for(int i=0;i<ruleTotal.getRules().size();i++)//遍历知识库中所有的规则，将其均设置值为未使用  {  if(ruleTotal.getRules().get(i).isUse())  ruleTotal.getRules().get(i).setUse(false);  }  ruleTemp.getKnowledge().removeAll(ruleTemp.getKnowledge());//将综合数据库清空  ruleTemp.getNum().removeAll(ruleTemp.getNum());  ruleTemp.setResult(null);//将数据库中的结论设置为null  ruleTemp.setTempResult(null);//将数据库中的临时结论设置为null； }  （8）添加规则：并进行适当的冲突消解：添加新的规则，并在添加前执行冲突消解以确保规则的一致性。通过比较新规则的结论和现有规则的结论，函数识别可能存在冲突的规则。如果找到结论相同的规则，函数会检查条件是否相互矛盾。它通过查找关键词（如 "有"、"无"、"可以"、"不可以"）并将其反转（例如，将 "有" 替换为 "无"），然后检查这些反转条件是否出现在现有规则中。如果发现新规则的条件与现有规则矛盾，函数打印一条消息并返回，不会添加新规则。如果新规则没有矛盾，它会被添加到规则库中。  //添加新的规格，并进行冲突消解  public void addRule(rule\_total ruleTotal)  {   rule\_base num16=new rule\_base();//创建新的规则类，存放新的规则  num16.getSitu().add("无奶");//设置条件  num16.setSize(num16.getSize()+1);   String clusion="哺乳动物";  num16.setClusion(clusion);//设置结论  String str;  //进行简易的冲突消解  //如果两条规则有相同的结论，则判断条件是否相互矛盾，主要以有无，可以不可以为关键词。  for(int i=0;i<ruleTotal.getRules().size();i++)  {  if(ruleTotal.getRules().get(i).getClusion().equals(clusion))  {  System.out.println("hello");  for(int j=0;j<num16.getSitu().size();j++)  {  str=num16.getSitu().get(j);  if(str.contains("无"))  {  str=str.replace("无","有");  }else  if(str.contains("有"))  {  str=str.replace("有","无");  }else  if(str.contains("可以"))  {  str=str.replace("可以","不可以");  }else  if(str.contains("不可以"))  {  str=str.replace("不可以","可以");  }  for(int k=0;k<ruleTotal.getRules().get(i).getSitu().size();k++)  {  if(str.equals(ruleTotal.getRules().get(i).getSitu().get(k)))  {  System.out.println("和已知规则矛盾");  return;  }  }  }  }  }  ruleTotal.getRules().add(num16);//在以上所有的条件均满足之后，将该规则类添加到知识库集合当中  } }  3、主界面设计：主界面分为背景，和8个按钮组成。  （1）背景类设计：  class BackgroundPanel extends JPanel {//创建一个背景类，该背景类继承Jpanel  private Image backgroundImage;//图像变量   public BackgroundPanel(String imagePath) {//构造函数  backgroundImage = new ImageIcon(imagePath).getImage();//为图像变量赋值图像地址  }   @Override  protected void paintComponent(Graphics g) {  super.paintComponent(g);//重写Jpanel的paintComponent函数  g.drawImage(backgroundImage, 0, 0, getWidth(), getHeight(), this);//绘制背景图像到绘画板当中  } }  （2）主界面设计（主要部分）：  rule\_total ruleTotal=new rule\_total();//创建知识库 rule\_temp ruleTemp=new rule\_temp();//创建数据库  operate operates=new operate();//创建操作函数对象 JFrame jf=new JFrame("动物识别专家系统");//创建图形界面 jf.setSize(450,300);//设置图形界面的大小 String imagePath = "C:\\Users\\30399\\Pictures\\Camera Roll\\下载4.jpg";//图像地址  // 创建自定义的背景面板 BackgroundPanel backgroundPanel = new BackgroundPanel(imagePath);//创建背景类   jf.setContentPane(backgroundPanel);//创建内容面板 jf.getContentPane().setPreferredSize(new Dimension(800, 600));//设置容器面板的大小  jf.setLayout(null);//随机布局 jf.setLocationRelativeTo(null);//居中显示 jf.setVisible(true);//可见 jf.setDefaultCloseOperation(EXIT\_ON\_CLOSE);//关闭窗口，结束程序。 jf.setResizable(false);//窗口大小不可变,使放大功能失效   //创建图形界面的8个操作按钮 JButton button1 = new JButton();//查看知识库 JButton button2 = new JButton();//修改知识库 JButton button3 = new JButton();//添加规则 JButton button4 = new JButton();//综合数据库 JButton button5 = new JButton();//推理 JButton button6 = new JButton();//解释 JButton button7 = new JButton();//复位 JButton button8 = new JButton();//退出  以下详细介绍以下每一按钮的设计：  （3）第一个按钮：查看知识库，将知识库中的所有数据输出到界面当中  //添加第一个按钮（查看知识库）的监听函数 button1.addActionListener(new ActionListener() {  public void actionPerformed(ActionEvent e) {  //创建一个新的窗口，并设置相应的属性  JFrame knowledgeBaseFrame = new JFrame("知识库窗口");  knowledgeBaseFrame.setSize(400, 300);  knowledgeBaseFrame.setLayout(null);  knowledgeBaseFrame.setLocationRelativeTo(jf); // 中心位置和主框架相同  knowledgeBaseFrame.setVisible(true);  knowledgeBaseFrame.setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE); // 点击关闭窗口仅仅只是关闭新窗口。   JTextArea textArea = new JTextArea();  textArea.setLineWrap(true); // 会自动换行  textArea.setWrapStyleWord(true); // 在单词边界换行  textArea.setEditable(false); // 设置为不可编辑  JScrollPane scrollPane = new JScrollPane(textArea);//设置滚动条，并将文本域添加到里面  scrollPane.setBounds(10, 10, 360, 180); // 设置其位置，大小  knowledgeBaseFrame.add(scrollPane); // 将滚动条添加到新窗口当中  //将所有的规则输出在文本域当中  for(int i=0;i<ruleTotal.getRules().size();i++)//遍历知识库，并将知识库的规则进行输出  {  String str;  textArea.append("第"+(i+1)+"条 "+"条件: ");  for(int j=0;j<ruleTotal.getRules().get(i).getSitu().size();j++)  {  str=ruleTotal.getRules().get(i).getSitu().get(j);  textArea.append(str+" ");  }  str=ruleTotal.getRules().get(i).getClusion();  textArea.append(" 结论: "+str+"\n");  }   } });  （4）第二个按钮：修改删除某条规则 //修改，删除某条规则。 button2.addActionListener(new ActionListener() {  @Override  public void actionPerformed(ActionEvent e) {  ...  //删除按钮，根据用户输入的序号，删除该位置的规则，并重新进行打印知识库 buttonDel.addActionListener(new ActionListener() {  @Override  public void actionPerformed(ActionEvent e) {    int num=Integer.parseInt(textField2.getText());  if(ruleTotal.getRules().size()>=num)  {  ruleTotal.getRules().remove(num-1);//删除该规则  System.out.println("hello");  }    textArea.setText("");  ...  } });  ...  //修改按钮，可以修改某条规则的条件，或者添加条件 button9.addActionListener(new ActionListener() {  @Override  public void actionPerformed(ActionEvent e) {  String str,str2,str3;  int num=Integer.parseInt(textField2.getText());  str=textField3.getText();  str2=textField4.getText();  str3=textField5.getText();  if(str.isEmpty()&&str3.isEmpty())  {  }  else  if(str2.isEmpty())//  {  //如果修改后的内容框为空，但要修改内容的文本框不为空，则默认移除该条件  for(int i=0;i<ruleTotal.getRules().get(num-1).getSitu().size();i++)  {  if(ruleTotal.getRules().get(num-1).getSitu().get(i).equals(str))  {  ruleTotal.getRules().get(num-1).getSitu().remove(i);  break;   }  }  //如果移除后该规则的条件少于1，则默认移除该规则  if(ruleTotal.getRules().get(num-1).getSitu().size()==0)  {  ruleTotal.getRules().remove(num-1);  }  }  else  {   //若二者接不为空，则修改该条件的内容  for(int i=0;i<ruleTotal.getRules().get(num-1).getSitu().size();i++)  {  if(ruleTotal.getRules().get(num-1).getSitu().get(i).equals(str))//首先找到该条件，然后  {  ruleTotal.getRules().get(num-1).getSitu().set(i,str2);//修改该条件的内容  break;   }  }  }  if(str3!=null)//若果添加内容一栏不为空，则将该框内数据添加为一条新的条件  {  ruleTotal.getRules().get(num-1).getSitu().add(str3);  }  （5）第三个按钮：添加新的规则，可以让用户输入规则的条件，结论，以及是否为最终结论。同时在添加之前对条件进行适当的冲突消解。  //添加第三个按钮（添加知识库）的监听函数 button3.addActionListener(new ActionListener() {  //同样是创建一个新的窗口，此后的所有的监听函数均会有一个新的窗口，便不再赘述  @Override  public void actionPerformed(ActionEvent e) {  ...  //添加新的功能按钮，包括添加条件的文本域，添加结论的文本框，以及单选按钮，可以选择是否为最终结论 JLabel label6 = new JLabel(); JTextField textField6 = new JTextField(); JButton button10 = new JButton(); JLabel label5 = new JLabel(); JTextArea textArea2=new JTextArea();  ...  //button10表示确认创建按钮，添加其监听函数 button10.addActionListener(new ActionListener() {  @Override  public void actionPerformed(ActionEvent e) {  String str1=textArea2.getText();//首先获取图形界面的数据，获取条件  System.out.println(str1);  String[] strs=str1.split("\n");  rule\_base num16=new rule\_base();  for(int i=0;i<strs.length;i++)  {  num16.getSitu().add(strs[i]);  num16.setSize(num16.getSize()+1);  }  String str2=textField6.getText();//获取结论  num16.setClusion(str2);  if(yesButton.isSelected())//获取是否为最终结论  {  num16.setEnd(true);  }  if(noButton.isSelected())  {  num16.setEnd(false);  }  ...  //进行适当冲突消解 String str; for(int i=0;i<ruleTotal.getRules().size();i++) {  if(ruleTotal.getRules().get(i).getClusion().equals(str2))  {  System.out.println("hello");  for(int j=0;j<num16.getSitu().size();j++)  {  str=num16.getSitu().get(j);  if(str.contains("无"))  {  str=str.replace("无","有");  }else  if(str.contains("有"))  {  str=str.replace("有","无");  }else  if(str.contains("可以"))  {  str=str.replace("可以","不可以");  }else  if(str.contains("不可以"))  {  str=str.replace("不可以","可以");  }  for(int k=0;k<ruleTotal.getRules().get(i).getSitu().size();k++)  {  if(str.equals(ruleTotal.getRules().get(i).getSitu().get(k)))  {  JOptionPane.showMessageDialog(null,"和已知结论矛盾","error",0);  return;  }  }  }  } } ruleTotal.getRules().add(num16);//将新的规则添加到知识库当中 JOptionPane.showMessageDialog(null,"Success","tip",1);  （6）第四个按钮：输入已知事实到综合数据库当中  //添加数据库，根据用户的输入，添加到数据库当中 button4.addActionListener(new ActionListener() {  @Override  public void actionPerformed(ActionEvent e) {  ...  ...  button11.addActionListener(new ActionListener() {  @Override  public void actionPerformed(ActionEvent e) {  String str=textArea3.getText();  String[] strs=str.split("\n");  for(int i=0;i<strs.length;i++)  {  ruleTemp.getKnowledge().add(strs[i]);  }  JOptionPane.showMessageDialog(null,"事实添加成功！","提示",1);  knowledgeBaseFrame.dispose();    }  });    } });  （7）第五个按钮：推理，补充事实，用户提示。  //添加第五个按钮（推理）的监听函数 //此部分包含推理，补充事实以及根据已有线索进行提示等功能 button5.addActionListener(new ActionListener() {  @Override  public void actionPerformed(ActionEvent e) {  //主要算法部分，判断是否得出最终结论，如果得出，直接打印相应的结论， if(ruleTemp.getResult()!=null) {  textField7.setText(ruleTemp.getResult()); } else//如果不是，会打印出目前能推出的最近的物种，比如食肉动物，并开始为用户提供线索提示 {  if(ruleTemp.getTempResult()==null)//判断是否有得出结论(非最终结论)  textField7.setText("啥也没有");//如果没有，输出空  else  {  //如果不是，则会输出最近的物种  textField7.setText("无法推出最终结论，但可以推出它的种类是："+ruleTemp.getTempResult());  textArea11.append("你可以根据以下提示进行事实补充："+"\n");  int num=operates.tip(ruleTotal,ruleTemp);//同时对用户进行提示  //将提示的所有内容输出到界面道中  for(int i=0;i<ruleTotal.getRules().get(num).getSitu().size();i++)  {  int j=0;  for(j=0;j<ruleTemp.getKnowledge().size();j++)  {  if(ruleTemp.getKnowledge().get(j).equals(ruleTotal.getRules().get(num).getSitu().get(i)))  {  break;   }  }  if(j>=ruleTemp.getKnowledge().size())  {  textArea11.append("是否"+ruleTotal.getRules().get(num).getSitu().get(i)+"\n");  }    }   } }  （8）第六个按钮：解释结论是如何得出，使用上面的解释函数即可。  //添加第六个按钮（解释机）的监听函数 button6.addActionListener(new ActionListener() {  @Override  public void actionPerformed(ActionEvent e) {  ...  ...  //将使用过的规则进行打印，输出到图形界面  for(int i=0;i<ruleTemp.getNum().size();i++)//遍历所有使用过的规则  {  int c;  c=ruleTemp.getNum().get(i);  for(int j=0;j<ruleTotal.getRules().get(c).size;j++)//遍历该规则的所有条件  {  textArea4.append(ruleTotal.getRules().get(c).getSitu().get(j)+" ");//输出到图形界面  }  textArea4.append(ruleTotal.getRules().get(c).getClusion());//输出结论  textArea4.append("\n");   }  } });  （9）第七个按钮：使用复位函数即可。  //为button添加监听器，这个监听器在new的时候随即创建，创建之时并重写actionPerformed函数 //复位按钮，将知识库和数据库恢复到最初的状态 button7.addActionListener(new ActionListener() {  @Override  public void actionPerformed(ActionEvent e) {  operates.updataUse(ruleTotal,ruleTemp);     } });  （10）第八个按钮：退出程序。  //退出函数，结束程序，并退出 button8.addActionListener(new ActionListener() {  @Override  public void actionPerformed(ActionEvent e) {  System.exit(0);     } });  四、实验结果  1、主界面：    图一 主界面  2、展示知识库的内容    图二 知识库内容展示  3、修改知识库    图三 修改知识库  4、删除有羽毛->会飞这条规则    图四 删除某条规则  5、添加规则    图五 添加新规则  6、添加成功：    图六 添加规则成功  7、添加综合数据库：    图七 添加事实  8、默认推理：    图八 默认推理  9、线索不够的推理结果：    图九 线索不够时的推理结果  10、根据提示，补充事实：    图十 补充事实  11、解释推理：    图十一 解释  五、实验分析及反馈  通过这次实验，我对产生式知识表示和专家系统的理论有了更深刻的理解。通过将理论知识应用于实际的编程实践中，我不仅学习了如何设计和实现规则，还学习了如何通过编程解决实际问题。这对于提升我的编程能力和理解复杂系统的能力都非常有帮助。不仅如此，这个实验使我对Java编程语言的理解更加深入。在实现专家系统的过程中，我更加熟悉了Java的数据结构、控制流程和对象导向编程的概念。这些技能对于我未来在软件开发和系统设计方面的学习和工作将非常有用。我了解到，尽管专家系统在某些方面非常强大，但它们也有局限性，特别是在处理模糊、不确定或复杂交互的特征时。这说明，在设计系统时需要综合考虑多种因素，包括系统的灵活性和适应性。  总的来说，这次实验不仅提升了我的编程技能和解决问题的能力，也激发了我对人工智能更深层次的兴趣和好奇心。它让我意识到，理论知识和实践应用之间的桥梁是如此重要，希望未来在这个充满挑战和机遇的领域继续学习和成长。 |

|  |
| --- |
|  |

|  |
| --- |
|  |