# 《人工智能》实验一报告

|  |  |  |  |
| --- | --- | --- | --- |
| 姓名 | 俞家宝 | 学号 | 2021117338 |
| 实验地点 | 院楼321实验室 | 实验日期 | 2023/10/16 |

1. **实验内容**

Implement unification algorithms in Prolog or Lisp or Java or C++.

**实验要求：**

**1.内容完备：方法简介、实验设置、实验结果及分析、代码；**

**2.格式规范、排版美观**

1. **实验原理**

合一算法的原理如下：

1. 合一算法首先检查两个表达式的根部是否相同，如果不同，则认为两个表达式无法统一，算法失败。

2. 如果两个表达式的根部符号相同，算法会检查它们的参数是否具有相同的数量。如果参数数量不同，则认为两个表达式无法统一，算法失败。

3. 如果两个表达式的根部符号和参数数量都相同，算法会逐个比较参数。对于每个参数，有以下几种情况：

- 如果参数是相同的常量或变量，则认为它们可以统一

- 如果参数是不同的常量或变量，则认为它们无法统一，算法失败。

4. 如果所有的参数都能成功统一，那么合一算法就会返回一个替代方案，将原始表达式中的变量替换为相应的常量或表达式，从而使得两个表达式相等或兼容。

1. **实验过程以及结果分析**

**使用函数：**

1. 主函数main()
2. 输入函数input()
3. 检查表达式函数checkLegality()
4. 格式转换函数exchangeFormat()
5. 执行合一置换函数doUnification(String[] legality1, String[] legality2)
6. 置换检查函数checkChange(String element1, String element2)
7. 置换函数doChange(String[] legality1, String[] legality2)
8. 元素类型判断函数getType(String element)
9. 表达式合法性判断judgeInputLegal(String legality)

实现代码：

*/\*\*  
 \* @author 俞家宝  
 \* @version 1.0  
 \* @className AiUnification  
 \* @dataTime 2023/10/16 18:15  
 \* 用java实现合一算法  
 \*/*  
import java.util.\*;  
import java.util.regex.Pattern;  
  
public class AiUnification {  
 */\*\*  
 \* LETTER\_CONSTANT : 常量集  
 \* minimalistUnification ： 合一式  
 \* f1，f2 ： 输入的两个表达式  
 \* lastLegality ： 最终置换后的结果  
 \*/* public static String *LETTER\_CONSTANT* = "abcdefghigklmnopqrstuvwxyz0123456789";  
 public static Map<String, String> *minimalistUnification* = new HashMap<>();  
 public static String *f1* = null, *f2* = null;  
 public static StringBuilder *lastLegality* = new StringBuilder();  
  
 */\*\*  
 \* 主函数  
 \*/* public static void main(String[] args) {  
 //对输入的表达式进行是否合法判断 ---> name(元素,元素,...)  
 while (!*input*()) {  
 System.*out*.println("输入不规范,请重新输入");  
 }  
 //如果输入的表达式不是一个函数或者元素的个数不匹配，则无法合一  
 if (!*checkLegality*()) System.*out*.println("无法进行合一，原因：函数符号不匹配或参数个数不匹配");  
 //如果两个表达式相对应的元素是不相桐的常量，则无法进行置换，无法合一  
 else if (!*exchangeFormat*()) System.*out*.println("无法进行合一，原因：变量约束冲突");  
 else {  
 System.*out*.println("最终合一表达式：" + *lastLegality*);  
 StringBuilder finalUnification;  
 finalUnification = new StringBuilder().append("{");  
 for (String key : *minimalistUnification*.keySet()) { finalUnification.append(key).append("/").append(*minimalistUnification*.get(key)).append(",");  
 }  
 finalUnification.deleteCharAt(finalUnification.length() - 1).append("}");  
 System.*out*.println("最一般合一式：" + finalUnification.toString());  
 }  
}

*/\*\*  
 \* 分别输入两个表达式，并调用judgeInputLegal()进行合法判断，不合法则返回false  
 \*/* public static boolean input() {  
 Scanner in = new Scanner(System.*in*);  
 System.*out*.println("input Legality1:f(?,?,?,...) or g(?,?,...) or name(?,?,...) or ... in String");  
 *f1* = in.next();  
 if (!*judgeInputLegal*(*f1*)) return false;  
 System.*out*.println("input Legality2:f(?,?,?,...) or g(?,?,...) or name(?,?,...) or ... in String");  
 *f2* = in.next();  
 return *judgeInputLegal*(*f2*);  
 }  
 */\*\*  
 \* 对表达式的元素个数进行比较，元素个数不相等则无法合一，返回false  
 \* 原理：元素以","分隔，所以统计","的个数判断元素的个数  
 \*/* public static boolean checkLegality() {  
 if (*f1*.charAt(0) != *f2*.charAt(0)) return false;  
 int count1 = *f1*.length() - *f1*.replace(",", "").length();  
 int count2 = *f2*.length() - *f2*.replace(",", "").length();  
 return count1 == count2;  
 }  
 */\*\*  
 \* 将表达式转换成只有元素的字符数组列表  
 \* 如果最后一个数组地址内容为空，则证明输入的表达式依旧不合法，所以返回false  
 \* 调用doUnification()开始进行合一置换  
 \*/* public static boolean exchangeFormat() {  
 *lastLegality*.append(*f1*.split("\\(")[0]).append("(");  
 *f1* = *f1*.split("\\(")[1];  
 *f2* = *f2*.split("\\(")[1];  
 *f1* = *f1*.replace(")", "");  
 *f2* = *f2*.replace(")", "");  
 String[] legality1 = *f1*.split(",");  
 String[] legality2 = *f2*.split(",");  
 if (legality1[legality1.length - 1].isEmpty() || legality2[legality2.length - 1].isEmpty()) return false;  
 return *doUnification*(legality1, legality2); }  
 */\*\*  
 \* @param legality1 表达式1的元素数组  
 \* @param legality2 表达式2的元素数组  
 \* 对元素个数进行遍历：对相对应的元素调用checkChange()进行比较  
 \* 如果可以进行置换,调用doChange()进行置换  
 \* 对置换后的元素进行遍历得到置换后的两个表达式，进行比较是否相同，相同则成功合一，返回true  
 \*/* public static boolean doUnification(String[] legality1, String[] legality2) {  
 int index = legality1.length;  
 for (int i = 0; i < index; i++) {  
 if (!*checkChange*(legality1[i], legality2[i])) return false;   
 *doChange*(legality1, legality2);  
 }  
 for (int i = 0; i < index; i++) {  
 *lastLegality*.append(legality1[i]).append(",");  
 }  
 *lastLegality*.deleteCharAt(*lastLegality*.length() - 1).append(")");  
 return true;  
 }  
 */\*\*  
 \* @param element1 表达式1的元素  
 \* @param element2 表达式2的元素  
 \* 如果两个元素相等，则不用进行置换就可以合一，返回true  
 \* 如果两个元素不等且都是常量，则不能合一，返回false  
 \* 如果两个元素一个是变量，一个是常量 ---> 则可以进行置换，将合一式存入HashMap中用变量作为Key，用常量作为Value，用Value替换Key  
 \* 如果两个元素都是变量，则用表达式2的元素置换表达式1的元素  
 \*/* public static boolean checkChange(String element1, String element2) {  
 if (element1.equals(element2)) return true;  
 else {  
 if (*getType*(element1).equals("constant") && *getType*(element2).equals("constant")) return false;   
 else {  
 if (*getType*(element1).equals("constant") && !*getType*(element2).equals("constant"))  
 *minimalistUnification*.put(element2, element1);  
 else if (*getType*(element2).equals("constant") && !*getType*(element1).equals("constant"))  
 *minimalistUnification*.put(element1, element2);  
 else if (!*getType*(element2).equals("constant") && !*getType*(element2).equals("constant"))  
 *minimalistUnification*.put(element1, element2);  
 return true; }}}

*/\*\*  
 \* @param legality1 表达式1的元素列表  
 \* @param legality2 表达式2的元素列表  
 \* 进行置换：遍历元素个数，并在HashMap中查找是否包含对应的Key，并用对应Key的Value进行替换  
 \*/* public static void doChange(String[] legality1, String[] legality2) {  
 int index = legality1.length;  
 for (int i = 0; i < index; i++) {  
 if (*minimalistUnification*.containsKey(legality1[i]))  
 legality1[i] = *minimalistUnification*.get(legality1[i]);  
 if (*minimalistUnification*.containsKey(legality2[i]))  
 legality2[i] = *minimalistUnification*.get(legality2[i]);  
 }  
 }  
  
 */\*\*  
 \* @param element 元素  
 \* 对元素类型进行判断，如果元素存在于常量集中，则是常量，返回constant；否则一致定义为变量返回variable  
 \*/* public static String getType(String element) {  
 if (*LETTER\_CONSTANT*.contains(element))  
 return "constant";  
 else  
 return "variable";  
 }

*/\*\*  
 \* @param legality 表达式  
 \* 对表达式进行合法性判断  
 \* 用正则表达式 --> 以字母或\_或$开头的包含任意字母或\_或$加上()，并且()中包含任意不是()的字符串  
 \* 如果满足正则，则返回true证明表达式合法；否则不合法返回false  
 \*/* public static boolean judgeInputLegal(String legality) {  
 String judge = "^([a-zA-Z\_$][a-zA-Z\\d\_$]\*)\\s\*\\(([^()]\*?)\\)\\s\*$";  
 Pattern regex = Pattern.*compile*(judge);  
 return regex.matcher(legality).matches();  
 }  
}

**运行结果截图：**

输入不合法：

文本

描述已自动生成

成功合一：

文本

描述已自动生成

文本

描述已自动生成

文本

描述已自动生成

无法合一：

文本

描述已自动生成

文本

描述已自动生成

文本

描述已自动生成

1. **实验总结**

使用java实操初步实现了合一算法的实现，进一步了解并且理解了合一算法的执行流程，其中遇到了许多困难，比如如何进行置换判断，如何判断输入的合法性等等，将困难攻克之后个人得到了很大的提升。