**人工智能上机实验报告**

|  |  |  |
| --- | --- | --- |
| 课程名称：人工智能 | 班级：计科2301 | 实验日期：2025-10-29 |
| 姓名（组长）： 贺祥宇 | 学号：2312190107 | 指导教师： 万庆 |
| 姓名： 樊炎琪 | 学号：2223040416 |
|  |  |
| 实验序号：一 |  | 实验成绩： |
| 一、实验名称  **知识表示与推理-Resolution Theorem Prover** | |
| 二、实验目的及要求  1、熟悉Knowledge representation/base的基本概念和表示法；  2、明确命题逻辑和一阶逻辑的运算规则和推理方法；  3、熟练使用Resolution算法自动化推理一阶逻辑问题。 | | |
| 三、实验环境  Python语言/C++语言 | | |
| 四、实验内容  设计并开发一个基于Two-Pointer resolution的逻辑推理程序，该程序能探索并解决至少以下两个课堂讨论过的一阶逻辑问题：   1. Howling Hounds    1. All hounds howl at night    2. Anyone who has any cats will not have any mice    3. Light sleepers do not have anything which howls at night    4. John has either a cat or a hound   Prove: If John is a light sleeper, then John does not have any mice   1. Drug dealer and customs official    1. The customs officials searched everyone who entered the country who was not a VIP    2. Some of the drug dealers entered the country, and they were only searched by drug dealers    3. No drug dealer was a VIP    4. Some of the customs officials were drug dealers   提示:  你需要至少考虑以下一些问题：   1. 如何将一阶逻辑中使用的逻辑符号在程序中表示出来？ 2. 如何设计unification算法？ 3. 如何实现resolution算法中的substitution功能？ 4. 如何设计算法的停机标准？   实验提交：  以小组为单位来完成，一般所有小组成员得分共享。由组长负责提交，将所有内容打包成.zip文件，要求：   1. 文件名：以组长的信息命名学号-班级-姓名.zip （5分）； 2. **报告中写清楚你小组所有成员的姓名和学号；** 3. .zip内包含：你全部的代码和程序（关键的算法与步骤需注释），实验报告.docx，README.txt等。其中README.txt描述你的代码运行环境和方法 （10分）； 4. .zip文件在2025年10月31号晚上8点前提交；   **迟交扣分：20分/天。** | | |
| 五、实验方法与算法设计  1、输入定义方式：  数据结构定义程序使用面向对象的方式定义逻辑元素：  Term ：术语基类（变量、常量、函数）  Variable ：变量（如 x, y）  Constant ：常量（如 j 表示 John）  Function ：函数项（如 Skolem 函数 f(x)）  Predicate ：谓词（如 HasHound(x)）  Literal ：文字（谓词或其否定）  Clause ：子句（文字的析取）  2、Clauses表示方式：  内部数据结构如下所示：  @dataclass(frozen=True)  class Clause:  literals: Tuple[Literal, ...] # 文字元组  @dataclass(frozen=True)  class Literal:  pred: Predicate # 谓词  negated: bool = False # 是否否定  示例表示：  子句 ¬HasHound(x) ∨ HasHowler(x) 表示为：  Literal1: Predicate("HasHound", (x,)), negated=True  Literal2: Predicate("HasHowler", (x,)), negated=False  Clause: 包含这两个Literal的元组  空子句 ⊥ 表示为包含空元组的Clause。  3、算法原理  核心算法流程如下：  1、初始化 ：加载所有初始子句到知识库  2、归结循环 ：对所有子句对尝试二元归结  3、变量标准化 ：归结前对第二个子句重命名变量避免冲突  4、合一匹配 ：找到互补文字并计算最广通代  5、生成归结式 ：合并两个子句，去除互补文字，应用替换  6、去重 ：避免生成重复子句  7、终止检查 ：生成空子句或无法生成新子句时停止  关键算法细节：  1、归结算法：  1. 对c2变量标准化  2. 遍历所有文字对寻找互补对  3. 合一匹配计算替换  4. 生成新子句并去重  2、合一算法 ：  1. 使用occur检查避免循环替换  2. 支持变量、常量、函数的合一  3. 基于替换应用的迭代算法  3、依赖追踪 ：  1. 记录每个子句的父节点和归结信息  2. 支持证明链回溯，检查是否使用否定目标  4、停机准则：  成功终止条件：  if resolvent.is\_empty(): # 生成空子句⊥  return True, "证明成功"  失败终止条件：  if not changed: # 当前循环未生成任何新子句  return False, "证明失败"  停机准则设有避免无限循环的机制：  1、子句去重 ：使用字符串表示检查子句是否已存在  2、有限生成 ：n个文字的初始子句最多生成2^n个子句  3、标准化避免冗余 ：变量重命名避免语法不同但逻辑相同的子句 | | |
| 六、实验结果  problem1：  Knowledge base clauses (CNF):  C1: ¬HasHound(x) ∨ HasHowler(x)  C2: ¬HasCat(x) ∨ ¬HasMouse(x)  C3: ¬L(x) ∨ ¬HasHowler(x)  C4: HasCat(j) ∨ HasHound(j)  C5: L(j) [NEGATED GOAL]  C6: HasMouse(j) [NEGATED GOAL]  Result: Refutation succeeded. Empty clause derived.  Proof chain (backtrace):  C2: ¬HasCat(x) ∨ ¬HasMouse(x)  C6: HasMouse(j) [NEGATED GOAL]  C10: resolved from C2(lit 2) & C6(lit 1) with {x→j} => ¬HasCat(j)  C1: ¬HasHound(x) ∨ HasHowler(x)  C4: HasCat(j) ∨ HasHound(j)  C8: resolved from C1(lit 1) & C4(lit 2) with {x→j} => HasHowler(j) ∨ HasCat(j)  C3: ¬L(x) ∨ ¬HasHowler(x)  C5: L(j) [NEGATED GOAL]  C11: resolved from C3(lit 1) & C5(lit 1) with {x→j} => ¬HasHowler(j)  C21: resolved from C8(lit 1) & C11(lit 1) with {} => HasCat(j)  C26: resolved from C10(lit 1) & C21(lit 1) with {} => ⊥  [OK] Empty clause uses at least one negated goal clause.  总计59步完成证明  problem2：  Knowledge base clauses (CNF):  C1: ¬EN(x) ∨ VIP(x) ∨ CO(f(x))  C2: ¬EN(x) ∨ VIP(x) ∨ S(f(x), x)  C3: DD(a)  C4: EN(a)  C5: ¬S(y, a) ∨ DD(y)  C6: ¬DD(x) ∨ ¬VIP(x)  C7: ¬CO(y) ∨ ¬DD(y) [NEGATED GOAL]  Result: Refutation succeeded. Empty clause derived.  Proof chain (backtrace):  C4: EN(a)  C1: ¬EN(x) ∨ VIP(x) ∨ CO(f(x))  C7: ¬CO(y) ∨ ¬DD(y) [NEGATED GOAL]  C10: resolved from C1(lit 3) & C7(lit 1) with {y\_231→f(x)} => ¬EN(x) ∨ VIP(x) ∨ ¬DD(f(x))  C3: DD(a)  C6: ¬DD(x) ∨ ¬VIP(x)  C14: resolved from C3(lit 1) & C6(lit 1) with {x\_239→a} => ¬VIP(a)  C52: resolved from C10(lit 2) & C14(lit 1) with {x→a} => ¬EN(a) ∨ ¬DD(f(a))  C2: ¬EN(x) ∨ VIP(x) ∨ S(f(x), x)  C5: ¬S(y, a) ∨ DD(y)  C12: resolved from C2(lit 3) & C5(lit 1) with {x→a, y\_234→f(x)} => ¬EN(a) ∨ VIP(a) ∨ DD(f(x))  C59: resolved from C12(lit 2) & C14(lit 1) with {} => ¬EN(a) ∨ DD(f(x))  C399: resolved from C52(lit 2) & C59(lit 2) with {x\_2225→a} => ¬EN(a)  C805: resolved from C4(lit 1) & C399(lit 1) with {} => ⊥  [OK] Empty clause uses at least one negated goal clause.  总计1302步完成证明 | | |
| 七、实验总结  算法优点:    1、完备性 ：一阶逻辑的归结是半可判定的  2、可靠性 ：只有逻辑蕴涵的结论才能被证明  3、依赖追踪 ：可回溯证明链，验证证明正确性  4、标准化处理 ：避免变量命名冲突  算法局限性：  1、组合复杂度 ：子句数量可能指数增长  2、启发式缺失 ：未实现文字选择等优化策略  3、效率问题 ：对于复杂问题可能无法在合理时间内终止  可改进之处：   1. 可以进一步优化性能：搜索策略优化、添加索引与过滤、优化子句管理； 2. 进一步完备算法：添加等式处理和高级归结策略 3. 增强输入输出：采取更友好的输入格式，并实现交互式证明环境 | | |
| 附录：  problem1：  Resolution trace:  Step 1: resolve [C1:¬HasHound(x) ∨ HasHowler(x)] (lit 2) with [C3:¬L(x) ∨ ¬HasHowler(x)] (lit 2)  substitution: {x→x\_2}  => ¬HasHound(x\_2) ∨ ¬L(x\_2)  Step 2: resolve [C1:¬HasHound(x) ∨ HasHowler(x)] (lit 1) with [C4:HasCat(j) ∨ HasHound(j)] (lit 2)  substitution: {x→j}  => HasHowler(j) ∨ HasCat(j)  Step 3: resolve [C2:¬HasCat(x) ∨ ¬HasMouse(x)] (lit 1) with [C4:HasCat(j) ∨ HasHound(j)] (lit 1)  substitution: {x→j}  => ¬HasMouse(j) ∨ HasHound(j)  Step 4: resolve [C2:¬HasCat(x) ∨ ¬HasMouse(x)] (lit 2) with [C6:HasMouse(j)] (lit 1)  substitution: {x→j}  => ¬HasCat(j)  Step 5: resolve [C3:¬L(x) ∨ ¬HasHowler(x)] (lit 1) with [C5:L(j)] (lit 1)  substitution: {x→j}  => ¬HasHowler(j)  Step 6: resolve [C1:¬HasHound(x) ∨ HasHowler(x)] (lit 2) with [C3:¬L(x) ∨ ¬HasHowler(x)] (lit 2)  substitution: {x→x\_17}  => ¬HasHound(x\_17) ∨ ¬L(x\_17)  Step 7: resolve [C1:¬HasHound(x) ∨ HasHowler(x)] (lit 1) with [C4:HasCat(j) ∨ HasHound(j)] (lit 2)  substitution: {x→j}  => HasHowler(j) ∨ HasCat(j)  Step 8: resolve [C1:¬HasHound(x) ∨ HasHowler(x)] (lit 1) with [C9:¬HasMouse(j) ∨ HasHound(j)] (lit 2)  substitution: {x→j}  => HasHowler(j) ∨ ¬HasMouse(j)  Step 9: resolve [C1:¬HasHound(x) ∨ HasHowler(x)] (lit 2) with [C11:¬HasHowler(j)] (lit 1)  substitution: {x→j}  => ¬HasHound(j)  Step 10: resolve [C2:¬HasCat(x) ∨ ¬HasMouse(x)] (lit 1) with [C4:HasCat(j) ∨ HasHound(j)] (lit 1)  substitution: {x→j}  => ¬HasMouse(j) ∨ HasHound(j)  Step 11: resolve [C2:¬HasCat(x) ∨ ¬HasMouse(x)] (lit 2) with [C6:HasMouse(j)] (lit 1)  substitution: {x→j}  => ¬HasCat(j)  Step 12: resolve [C2:¬HasCat(x) ∨ ¬HasMouse(x)] (lit 1) with [C8:HasHowler(j) ∨ HasCat(j)] (lit 2)  substitution: {x→j}  => ¬HasMouse(j) ∨ HasHowler(j)  Step 13: resolve [C3:¬L(x) ∨ ¬HasHowler(x)] (lit 1) with [C5:L(j)] (lit 1)  substitution: {x→j}  => ¬HasHowler(j)  Step 14: resolve [C3:¬L(x) ∨ ¬HasHowler(x)] (lit 2) with [C8:HasHowler(j) ∨ HasCat(j)] (lit 1)  substitution: {x→j}  => ¬L(j) ∨ HasCat(j)  Step 15: resolve [C4:HasCat(j) ∨ HasHound(j)] (lit 2) with [C7:¬HasHound(x\_2) ∨ ¬L(x\_2)] (lit 1)  substitution: {x\_2\_45→j}  => HasCat(j) ∨ ¬L(j)  Step 16: resolve [C4:HasCat(j) ∨ HasHound(j)] (lit 1) with [C10:¬HasCat(j)] (lit 1)  substitution: {}  => HasHound(j)  Step 17: resolve [C5:L(j)] (lit 1) with [C7:¬HasHound(x\_2) ∨ ¬L(x\_2)] (lit 2)  substitution: {x\_2\_51→j}  => ¬HasHound(j)  Step 18: resolve [C6:HasMouse(j)] (lit 1) with [C9:¬HasMouse(j) ∨ HasHound(j)] (lit 1)  substitution: {}  => HasHound(j)  Step 19: resolve [C7:¬HasHound(x\_2) ∨ ¬L(x\_2)] (lit 1) with [C9:¬HasMouse(j) ∨ HasHound(j)] (lit 2)  substitution: {x\_2→j}  => ¬L(j) ∨ ¬HasMouse(j)  Step 20: resolve [C8:HasHowler(j) ∨ HasCat(j)] (lit 2) with [C10:¬HasCat(j)] (lit 1)  substitution: {}  => HasHowler(j)  Step 21: resolve [C8:HasHowler(j) ∨ HasCat(j)] (lit 1) with [C11:¬HasHowler(j)] (lit 1)  substitution: {}  => HasCat(j)  Step 22: resolve [C1:¬HasHound(x) ∨ HasHowler(x)] (lit 2) with [C3:¬L(x) ∨ ¬HasHowler(x)] (lit 2)  substitution: {x→x\_72}  => ¬HasHound(x\_72) ∨ ¬L(x\_72)  Step 23: resolve [C1:¬HasHound(x) ∨ HasHowler(x)] (lit 1) with [C4:HasCat(j) ∨ HasHound(j)] (lit 2)  substitution: {x→j}  => HasHowler(j) ∨ HasCat(j)  Step 24: resolve [C1:¬HasHound(x) ∨ HasHowler(x)] (lit 1) with [C9:¬HasMouse(j) ∨ HasHound(j)] (lit 2)  substitution: {x→j}  => HasHowler(j) ∨ ¬HasMouse(j)  Step 25: resolve [C1:¬HasHound(x) ∨ HasHowler(x)] (lit 2) with [C11:¬HasHowler(j)] (lit 1)  substitution: {x→j}  => ¬HasHound(j)  Step 26: resolve [C1:¬HasHound(x) ∨ HasHowler(x)] (lit 1) with [C18:HasHound(j)] (lit 1)  substitution: {x→j}  => HasHowler(j)  Step 27: resolve [C2:¬HasCat(x) ∨ ¬HasMouse(x)] (lit 1) with [C4:HasCat(j) ∨ HasHound(j)] (lit 1)  substitution: {x→j}  => ¬HasMouse(j) ∨ HasHound(j)  Step 28: resolve [C2:¬HasCat(x) ∨ ¬HasMouse(x)] (lit 2) with [C6:HasMouse(j)] (lit 1)  substitution: {x→j}  => ¬HasCat(j)  Step 29: resolve [C2:¬HasCat(x) ∨ ¬HasMouse(x)] (lit 1) with [C8:HasHowler(j) ∨ HasCat(j)] (lit 2)  substitution: {x→j}  => ¬HasMouse(j) ∨ HasHowler(j)  Step 30: resolve [C2:¬HasCat(x) ∨ ¬HasMouse(x)] (lit 1) with [C16:¬L(j) ∨ HasCat(j)] (lit 2)  substitution: {x→j}  => ¬HasMouse(j) ∨ ¬L(j)  Step 31: resolve [C2:¬HasCat(x) ∨ ¬HasMouse(x)] (lit 1) with [C17:HasCat(j) ∨ ¬L(j)] (lit 1)  substitution: {x→j}  => ¬HasMouse(j) ∨ ¬L(j)  Step 32: resolve [C2:¬HasCat(x) ∨ ¬HasMouse(x)] (lit 1) with [C21:HasCat(j)] (lit 1)  substitution: {x→j}  => ¬HasMouse(j)  Step 33: resolve [C3:¬L(x) ∨ ¬HasHowler(x)] (lit 1) with [C5:L(j)] (lit 1)  substitution: {x→j}  => ¬HasHowler(j)  Step 34: resolve [C3:¬L(x) ∨ ¬HasHowler(x)] (lit 2) with [C8:HasHowler(j) ∨ HasCat(j)] (lit 1)  substitution: {x→j}  => ¬L(j) ∨ HasCat(j)  Step 35: resolve [C3:¬L(x) ∨ ¬HasHowler(x)] (lit 2) with [C13:HasHowler(j) ∨ ¬HasMouse(j)] (lit 1)  substitution: {x→j}  => ¬L(j) ∨ ¬HasMouse(j)  Step 36: resolve [C3:¬L(x) ∨ ¬HasHowler(x)] (lit 2) with [C15:¬HasMouse(j) ∨ HasHowler(j)] (lit 2)  substitution: {x→j}  => ¬L(j) ∨ ¬HasMouse(j)  Step 37: resolve [C3:¬L(x) ∨ ¬HasHowler(x)] (lit 2) with [C20:HasHowler(j)] (lit 1)  substitution: {x→j}  => ¬L(j)  Step 38: resolve [C4:HasCat(j) ∨ HasHound(j)] (lit 2) with [C7:¬HasHound(x\_2) ∨ ¬L(x\_2)] (lit 1)  substitution: {x\_2\_130→j}  => HasCat(j) ∨ ¬L(j)  Step 39: resolve [C4:HasCat(j) ∨ HasHound(j)] (lit 1) with [C10:¬HasCat(j)] (lit 1)  substitution: {}  => HasHound(j)  Step 40: resolve [C4:HasCat(j) ∨ HasHound(j)] (lit 2) with [C12:¬HasHound(x\_17) ∨ ¬L(x\_17)] (lit 1)  substitution: {x\_17\_135→j}  => HasCat(j) ∨ ¬L(j)  Step 41: resolve [C4:HasCat(j) ∨ HasHound(j)] (lit 2) with [C14:¬HasHound(j)] (lit 1)  substitution: {}  => HasCat(j)  Step 42: resolve [C5:L(j)] (lit 1) with [C7:¬HasHound(x\_2) ∨ ¬L(x\_2)] (lit 2)  substitution: {x\_2\_146→j}  => ¬HasHound(j)  Step 43: resolve [C5:L(j)] (lit 1) with [C12:¬HasHound(x\_17) ∨ ¬L(x\_17)] (lit 2)  substitution: {x\_17\_151→j}  => ¬HasHound(j)  Step 44: resolve [C5:L(j)] (lit 1) with [C16:¬L(j) ∨ HasCat(j)] (lit 1)  substitution: {}  => HasCat(j)  Step 45: resolve [C5:L(j)] (lit 1) with [C17:HasCat(j) ∨ ¬L(j)] (lit 2)  substitution: {}  => HasCat(j)  Step 46: resolve [C5:L(j)] (lit 1) with [C19:¬L(j) ∨ ¬HasMouse(j)] (lit 1)  substitution: {}  => ¬HasMouse(j)  Step 47: resolve [C6:HasMouse(j)] (lit 1) with [C9:¬HasMouse(j) ∨ HasHound(j)] (lit 1)  substitution: {}  => HasHound(j)  Step 48: resolve [C6:HasMouse(j)] (lit 1) with [C13:HasHowler(j) ∨ ¬HasMouse(j)] (lit 2)  substitution: {}  => HasHowler(j)  Step 49: resolve [C6:HasMouse(j)] (lit 1) with [C15:¬HasMouse(j) ∨ HasHowler(j)] (lit 1)  substitution: {}  => HasHowler(j)  Step 50: resolve [C6:HasMouse(j)] (lit 1) with [C19:¬L(j) ∨ ¬HasMouse(j)] (lit 2)  substitution: {}  => ¬L(j)  Step 51: resolve [C7:¬HasHound(x\_2) ∨ ¬L(x\_2)] (lit 1) with [C9:¬HasMouse(j) ∨ HasHound(j)] (lit 2)  substitution: {x\_2→j}  => ¬L(j) ∨ ¬HasMouse(j)  Step 52: resolve [C7:¬HasHound(x\_2) ∨ ¬L(x\_2)] (lit 1) with [C18:HasHound(j)] (lit 1)  substitution: {x\_2→j}  => ¬L(j)  Step 53: resolve [C8:HasHowler(j) ∨ HasCat(j)] (lit 2) with [C10:¬HasCat(j)] (lit 1)  substitution: {}  => HasHowler(j)  Step 54: resolve [C8:HasHowler(j) ∨ HasCat(j)] (lit 1) with [C11:¬HasHowler(j)] (lit 1)  substitution: {}  => HasCat(j)  Step 55: resolve [C9:¬HasMouse(j) ∨ HasHound(j)] (lit 2) with [C12:¬HasHound(x\_17) ∨ ¬L(x\_17)] (lit 1)  substitution: {x\_17\_205→j}  => ¬HasMouse(j) ∨ ¬L(j)  Step 56: resolve [C9:¬HasMouse(j) ∨ HasHound(j)] (lit 2) with [C14:¬HasHound(j)] (lit 1)  substitution: {}  => ¬HasMouse(j)  Step 57: resolve [C10:¬HasCat(j)] (lit 1) with [C16:¬L(j) ∨ HasCat(j)] (lit 2)  substitution: {}  => ¬L(j)  Step 58: resolve [C10:¬HasCat(j)] (lit 1) with [C17:HasCat(j) ∨ ¬L(j)] (lit 1)  substitution: {}  => ¬L(j)  Step 59: resolve [C10:¬HasCat(j)] (lit 1) with [C21:HasCat(j)] (lit 1)  substitution: {}  => ⊥  Derived empty clause ⊥. Refutation complete.  problem2：  步骤过多，详见<https://kkgithub.com/kiraTheresa/AI-Lab1/blob/main/trace_problem2.txt> | | |
|  | | |