苏州大学实验报告

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 院系 | 计算机学院 | | 年级专业 | | 21计科 | | 姓名 | 方浩楠 | 学号 | 2127405048 |
| 课程名称 | | 人工智能与知识工程 | | | | | | | 成绩 |  |
| 指导教师 | | 杨壮 | | 同组实验者 | | 无 | | 实验日期 | 2023.10.10 | |

|  |  |
| --- | --- |
| 实 验 名 称 | 鲁宾孙归结原理实现对命题逻辑及谓词逻辑归结推理系统 |

1. 实验目的

熟悉并理解鲁滨逊归结原理的基本概念和方法。

编程实现命题逻辑和谓词逻辑的归结证明。

通过编程实践，加深对人工智能逻辑推理部分的理解和应用。

1. 实验内容

编写 Python 程序，实现命题逻辑的归结原理。主要包括文字和子句的表示，以及单个归结和多个归结的方法。

编写 Python 程序，实现谓词逻辑的归结证明。通过给定的谓词、变量和公式，执行归结证明，以确定是否可以从给定的公式推导出结论公式。

1. 实验步骤和结果

两个python文件中每个类,以及每个函数的作用,接口和返回值写在了**readme.md**中

1. 命题逻辑归结原理 (propositional\_resolution.py)

该文件主要实现了命题逻辑的归结原理。主要类和函数包括 Literal 类, Clause 类, resolve 函数和 resolve\_multiple 函数。

1.1 Literal 类

作用:

表示命题逻辑中的文字。

提供文字的字符串表示和否定操作。

Docstring:

*"""  
表示命题逻辑中的文字.  
  
Attributes:  
 name (str): 文字的名称.  
 negated (bool): 文字是否被否定.  
"""*

实现方式:

\_\_init\_\_ 方法初始化一个 Literal 对象，接受文字名称和否定标志作为参数。

Docstring:

*"""  
初始化一个Literal对象。  
  
Args:  
 name (str): 文字的名称。  
 negated (bool, optional): 表示文字是否被否定。默认为False。  
"""*

\_\_repr\_\_ 方法返回文字的字符串表示，包括是否有否定符号。

Docstring:

*"""返回此文字的字符串表示形式."""*

\_\_neg\_\_ 方法返回文字的否定，创建一个新的 Literal 对象，其否定标志与原文字相反。

Docstring:

*"""  
返回这个文字的否定.  
  
Returns:  
 Literal: 这个文字的否定.  
"""*

1.2 Clause 类

作用:

表示命题逻辑中的子句。

存储子句中的文字集合。

Docstring:

*"""  
表示命题逻辑中的子句.  
  
Attributes:  
 literals (set[Literal]): 子句中的文字集合.  
"""*

实现方式:

\_\_init\_\_ 方法初始化一个 Clause 对象，接受一个文字列表作为参数，并将其转换为集合存储。

1.3 resolve 函数

作用:

对两个子句应用归结规则，尝试得到一个新的子句。

Docstring:

*"""  
对两个子句应用归结规则，尝试得到一个新的子句.  
  
Args:  
 clause1 (Clause): 第一个子句.  
 clause2 (Clause): 第二个子句.  
  
Returns:  
 Clause: 如果可以应用归结规则得到一个新的子句，则返回新的子句;否则返回None.  
"""*

实现方式:

遍历两个子句中的所有文字，寻找具有相同名称但否定标志相反的文字对。

如果找到这样的文字对，合并两个子句的文字集合，并去除找到的文字对，得到一个新的子句。

1.4 resolve\_multiple 函数

作用:

对多个子句应用归结规则，尝试得到一个新的子句。

Docstring:  
*"""  
对多个子句应用归结规则，尝试得到一个新的子句.  
  
Args:  
 \*clauses (Clause): 多个子句.  
  
Returns:  
 Clause: 如果可以应用归结规则得到一个新的子句，则返回新的子句;否则返回None.  
"""*

实现方式:

初始化一个答案子句为输入子句列表中的第一个子句。

遍历所有输入子句，逐一与答案子句应用 resolve 函数进行归结，更新答案子句。

示例代码执行结果:

{R}

通过示例代码，成功地应用了 resolve\_multiple 函数对多个子句进行归结，得到了一个新的子句，并输出了该子句中的文字集合。

2. 谓词逻辑归结证明 (predicate\_resolution.py)

该文件主要实现了谓词逻辑的归结证明。核心函数为 predicate\_resolution 函数。

2.1 predicate\_resolution 函数

作用:

接受一组谓词、变量、给定的公式和一个结论公式，执行归结证明，以确定是否可以从给定的公式推导出结论公式。

Docstring:

*"""  
执行给定谓词、变量和公式的归结证明。  
  
该函数接受一组谓词、变量和公式，执行归结证明，然后返回证明的结果。  
  
Args:  
 predicates (str): 空格分隔的谓词名字符串。  
 variables (str): 空格分隔的变量名字符串。  
 givens (list): 给定的公式列表。  
 conclusion (str): 要证明的结论公式。  
  
Returns:  
 bool: 如果给定的公式能够推导出结论，则返回 True；否则返回 False。  
"""*

实现方式:

创建谓词和变量字典，以便将字符串公式转换为 pyprover 库可以处理的形式。

定义 parse\_formula 内部函数，将字符串公式转换为 pyprover 库可以处理的表达式。

对所有给定的公式和结论公式应用 parse\_formula 函数。

调用 pyprover 库的 proves 函数，执行归结证明。

示例代码执行结果:

通过示例代码，成功地应用了 predicate\_resolution 函数进行谓词逻辑的归结证明，并输出了证明结果，验证了实现的正确性。

1. 实验总结

通过本次实验，我对鲁滨逊归结原理有了更深的理解，特别是其在命题逻辑和谓词逻辑中的应用。

实验中的编程实践帮助我理解了如何将理论知识应用于实际问题，也让我熟悉了在 Python 中实现逻辑推理的方法。

我也学会了如何使用 pyprover 这个强大的库来辅助实现谓词逻辑的归结证明，提高了我的编程和解决问题的能力。

附录:源代码:

propositional\_resolution.py

*"""  
该模块实现了命题逻辑的归结原理。  
  
包括：  
- Literal类：表示命题逻辑中的文字。  
- Clause类：表示命题逻辑中的子句，包括单个归结和多个归结的方法。  
  
通过归结原理，我们可以检查一组逻辑公式是否自洽，或从一组公式中推导出新的公式。  
函数的使用示例包含在模块的 `\_\_main\_\_` 块中  
  
作者: 水告木南  
创建日期: 2023-10-17  
"""*class Literal:  
 *"""  
 表示命题逻辑中的文字.  
  
 Attributes:  
 name (str): 文字的名称.  
 negated (bool): 文字是否被否定.  
 """* def \_\_init\_\_(self, name: str, negated: bool = False):  
 *"""  
 初始化一个Literal对象。  
  
 Args:  
 name (str): 文字的名称。  
 negated (bool, optional): 表示文字是否被否定。默认为False。  
 """* self.name = name  
 self.negated = negated  
  
 def \_\_repr\_\_(self) -> str:  
 *"""返回此文字的字符串表示形式."""* return f"{'~' if self.negated else ''}{self.name}"  
  
 def \_\_neg\_\_(self) -> 'Literal':  
 *"""  
 返回这个文字的否定.  
  
 Returns:  
 Literal: 这个文字的否定.  
 """* return Literal(self.name, not self.negated)  
  
  
class Clause:  
 *"""  
 表示命题逻辑中的子句.  
  
 Attributes:  
 literals (set[Literal]): 子句中的文字集合.  
 """* def \_\_init\_\_(self, literals: list) -> None:  
 self.literals = set(literals)  
  
  
def resolve(clause1: Clause, clause2: Clause) -> Clause:  
 *"""  
 对两个子句应用归结规则，尝试得到一个新的子句.  
  
 Args:  
 clause1 (Clause): 第一个子句.  
 clause2 (Clause): 第二个子句.  
  
 Returns:  
 Clause: 如果可以应用归结规则得到一个新的子句，则返回新的子句;否则返回None.  
 """* new\_clause = None  
 for l1 in clause1.literals:  
 for l2 in clause2.literals:  
 if l1.name == l2.name and l1.negated != l2.negated:  
 new\_literals = (clause1.literals | clause2.literals) - {l1, l2}  
 new\_clause = Clause(new\_literals)  
 return new\_clause  
  
  
def resolve\_multiple(\*clauses: Clause) -> Clause:  
 *"""  
 对多个子句应用归结规则，尝试得到一个新的子句.  
  
 Args:  
 \*clauses (Clause): 多个子句.  
  
 Returns:  
 Clause: 如果可以应用归结规则得到一个新的子句，则返回新的子句;否则返回None.  
 """* ans\_clause = clauses[0]  
 clause1 = clauses[0]  
 for clause2 in clauses:  
 if clause1 != clause2:  
 ans\_clause = resolve(ans\_clause, clause2)  
 return ans\_clause  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 *# 示例用法:* c1 = Clause([Literal("P", negated=True), Literal("Q")])  
 c2 = Clause([Literal("Q", negated=True), Literal("R")])  
 c3 = Clause([Literal("P")])  
 clauses = resolve\_multiple(c1, c2, c3)  
 print(clauses.literals)

predicate\_resolution.py

*"""  
该模块包含一个用于执行谓词逻辑归结证明的函数。  
  
该模块定义了 `predicate\_resolution` 函数，该函数接受一组谓词、变量、给定的公式和一个结论公式，  
然后执行归结证明以确定是否可以从给定的公式推导出结论公式。  
  
函数的使用示例包含在模块的 `\_\_main\_\_` 块中，显示了如何调用 `predicate\_resolution` 函数并输出归结证明的结果。  
  
该模块依赖于 `pyprover` 库来执行归结证明。  
  
作者: 水告木南  
创建日期: 2023-10-21  
"""*from pyprover import props, terms, FA, TE, proves  
  
  
def predicate\_resolution(predicates, variables, givens, conclusion):  
 *"""  
 执行给定谓词、变量和公式的归结证明。  
  
 该函数接受一组谓词、变量和公式，执行归结证明，然后返回证明的结果。  
  
 Args:  
 predicates (str): 空格分隔的谓词名字符串。  
 variables (str): 空格分隔的变量名字符串。  
 givens (list): 给定的公式列表。  
 conclusion (str): 要证明的结论公式。  
  
 Returns:  
 bool: 如果给定的公式能够推导出结论，则返回 True；否则返回 False。  
 """  
 # 创建谓词和变量字典* predicate\_dict = {name: prop for name, prop in zip(predicates.split(), props(predicates))}  
 variable\_dict = {name: var for name, var in zip(variables.split(), terms(variables))}  
  
 *# 解析给定的公式和结论* def parse\_formula(formula):  
 for name, prop in predicate\_dict.items():  
 formula = formula.replace(name, f"predicate\_dict['{name}']")  
 for name, var in variable\_dict.items():  
 formula = formula.replace(name, f"variable\_dict['{name}']")  
 return eval(formula)  
  
 givens\_parsed = [parse\_formula(formula) for formula in givens]  
 conclusion\_parsed = parse\_formula(conclusion)  
  
 *# 执行归结证明* result = proves(givens\_parsed, conclusion\_parsed)  
  
 return result  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 predicates = "R S"  
 variables = "x y z"  
 givens = ["FA(x, R(x) >> S(x))", "TE(y, R(y))"]  
 conclusion = "TE(z, S(z))"  
 result = predicate\_resolution(predicates, variables, givens, conclusion)  
 print(f"The conclusion {conclusion} is {'valid' if result else 'invalid'}.")