中山大学计算机学院

人工智能

本科生实验报告

课程名称：Artificial Intelligence

学号	23336266	姓名	熊彦钧

一、 实验题目

1.命题逻辑的归结原理

2.最一般合一算法

二、 实验内容

实验一

1. 算法原理

1：归结原理是命题逻辑和一阶逻辑中用于自动推理和定理证明的核心方法。其基本思想是通过归结规则逐步消除互补文字，最终推导出空子句，从而证明命题的有效性。

步骤：1.将带证命题转化为合取范式->2.对包含互补文字的子句进行归结->3.重复归结直到生成空句子

2. 关键代码展示

①在归结子句的过程中，只有通过遍历才能找到包含互补，而在遍历的过程中，我们不能每次都从头开始，否则会陷入死循环，我们需要将遍历过的旧子句清除，并在列表中加入新子句。以下是部分代码展示：

```
def Resolu  • 此文件存在 2 个问题

    existing_clauses = set(clauses) #存储已有的子句,避免重复
    used_clauses_idx = set()

    new_clause_idx = len(clauses) + 1

    old_idx = 0
    new_idx = len(clauses)  #新句子的起始索引
    while old_idx < len(clauses):
        new_clause_added = False    #判读是否有新子句

        #遍历子句对
        for i in range(new_idx):
            if i in used_clauses_idx:
                continue

            for j in range(i+1, len(clauses)):
                if j in used_clauses_idx:
                    continue

                resolvent_found, new_clause, i_part, j_part = resolve(claus

                if resolvent_found and new_clause not in existing_clauses:
                    clauses.append(new_clause)
                    existing_clauses.add(new_clause)

                    used_clauses_idx.add(i)
                    used_clauses_idx.add(j)

                    #添加到结果中
                    step = f"{new_clause_idx} R[{i+1}{i_part},{j+1}{j_part}
                    result.append(step)

                    new_clause_idx += 1
                    new_clause_added = True

                    #结束循环
                    if new_clause == ():
                        return result

        if not new_clause_added:
            break

        old_idx = new_idx
        new_idx = len(clauses)

    return result
```

②对每一对子句，我们需要判断是否需要归结，判断的方法是看两个子句中有没有互补的文字，这一步我们可以通过 Python 对元素的快速搜索获得，获得新子句后，我们还要进行排序和去重，以防止陷入死循环

```python
#这个函数的作用是解析子句
def resolve(clause1, clause2):
    for i, lit1 in enumerate(clause1):
        for j, lit2 in enumerate(clause2):
            if (lit1.startswith('~') and lit1[1:] == lit2) or (lit2.startswith('~') and lit2[1:] == lit1):
                new_clause_list = []
                for lit in clause1:
                    if lit != lit1:
                        new_clause_list.append(lit)
                for lit in clause2:
                    if lit != lit2:
                        new_clause_list.append(lit)

                #对新子句进行排序和去重
                new_clause = tuple(sorted(set(new_clause_list)))

                i_part = get_literal_marker(clause1, i)
                j_part = get_literal_marker(clause2, j)

                return True, new_clause, i_part, j_part

    return False, (), "", ""
```

剩余代码详见附件，在此处省略

实验二

1.算法原理

最一般合一算法（MGU）用于一阶逻辑中寻找使两个表达式一致的变量替换。MGU是使两个表达式一致的最通用的替换。

步骤：1.初始化->2.对变量进行递归合一->3.遇到函数和谓词，递归合一对应的参数

2.关键代码展示

首先，我们需要一个函数来判断是需要递归变量还是要递归函数，这一部分的代码如下：

```
def is_variable(term):
    """检查是否是变量"""
    return term.islower()
```

```
def unify_term(term1, term2, substitution):
    """尝试统一两个项 term1 和 term2，返回新的替换字典"""
    if term1 == term2:
        return substitution     #如果两项相同，直接替换
    elif is_variable(term1):    #如果 term1 是变量
        return unify_variable(term1, term2, substitution)
    elif is_variable(term2):    #如果 term2 是变量
        return unify_variable(term2, term1, substitution)
    elif term1.startswith('(') and term2.startswith('('):
        return unify_function(term1, term2, substitution)   #如果两项都是函数
    else:
        return None #无法统一
```

判断完成之后，合一变量和合一函数的部分需要分开编写，因为他们之间的逻辑有些许不同，但是相同的是，都需要注意判断变量是否已经存在项中。合一变量和函数的部分分别如下：

```
def unify_variable(var, term, substitution):
    """统一变量与其他项"""
    if var in substitution: #如果变量已经有替换
        return unify_term(substitution[var], term, substitution)
    if term == var:
        return substitution #如果变量已经是该项，直接返回
    if occurs_check(var, term):
        return None
    substitution[var] = term     #将变量替换为项
    return substitution
```

```
def unify_function(func1, func2, substitution):
    """统一函数"""
    if func1[0] != func2[0]:
        return None #函数名不同，无法统一
    args1 = split_function(func1)
    args2 = split_function(func2)
    for arg1, arg2 in zip(args1, args2):
        substitution = unify_term(arg1, arg2, substitution) #逐个统一参数
        if substitution is None:
            return None
    return substitution
```

剩余代码详见附录

三、 实验结果及分析

1. 实验结果展示示例

根据该输入，实验一程序的输出如下：

```
if __name__ == "__main__":
    KB = [("FirstGrade",), ("~FirstGrade", "Child"), ("~Child",)]
    result = ResolutionProp(KB)
    for step in result:
        print(step) #输出结果
```

```
PS C:\Users\jhinx\Desktop\大学课件\人工智能作业> & C:/Python313/python.exe "c:/Users
/jhinx/Desktop/大学课件/人工智能作业/3-4 归结原理/第三周作业1（归结原理）.py"
1 (FirstGrade,)
2 (~FirstGrade,Child)
3 (~Child,)
4 R[1,2a]=(Child,)
5 R[3,4]=()
PS C:\Users\jhinx\Desktop\大学课件\人工智能作业>
```

经过检验，实验一结果正确；

根据以下输入，实验二结果如下：

```
# 示例测试
print(MGU('P(xx,a)', 'P(b,yy)'))  # 输出 {'xx': 'b', 'yy': 'a'}
print(MGU('P(a,xx,f(g(yy)))', 'P(2zz,f(zz),f(uu))'))  # 输出 {'zz': 'a', 'x
```

```
PS C:\Users\jhinx\Desktop\大学课件\人工智能作业> & C:/Python313/python.exe "c:/Users
/jhinx/Desktop/大学课件/人工智能作业/3-4 归结原理/第三周作业2（最一般合一算法）.py"
{'xx': 'b', 'a': 'yy'}
{'a': '2zz', 'xx': 'f(zz)', 'f(g(yy))': 'f(uu)'}
PS C:\Users\jhinx\Desktop\大学课件\人工智能作业>
```

经过检验，实验二结果正确，实验任务完成。