**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** |  | **姓名** |  |

# **Infer.py(第三周作业一)：**

# **实验题目**



# 实验内容

1. **算法原理**
2. 初始化

将知识库（KB）中的所有子句存储在一个列表中，保持顺序。

初始化一个集合 used\_pairs，用于记录已经尝试过归结的子句对，避免重复归结。

初始化一个列表 steps，用于记录每一步的推理过程，方便输出。

1. 归结推理过程:

尝试所有未归结的子句对：

如果没有新生成的子句，则考虑所有未归结的子句对。

使用两层循环生成所有可能的子句对，并检查它们是否已经在 used\_pairs 中。

1. 归结操作：

对于每一对子句，调用 resolve 函数尝试归结。

resolve 函数会检查两个子句中是否存在互补文字。

如果存在，则生成一个新的子句（归结式），并返回归结式及其对应的文字索引。

如果归结式已经存在于当前的子句集中，则跳过；否则，将归结式添加到子句集中，并记录归结步骤。

1. 检查终止条件：

如果生成的归结式是空子句（()），说明找到了矛盾，知识库不可满足，推理成功，返回推理步骤。

如果没有新的子句被添加到子句集中，说明无法继续归结，推理结束。

1. **关键代码展示**

#### **归结主循环**

while True:

added = False

pairs\_to\_try = []

if newly\_added\_indices:

for new\_idx in newly\_added\_indices:

for old\_idx in range(len(new\_sentences)):

if new\_idx != old\_idx and (new\_idx, old\_idx) not in used\_pairs and (old\_idx, new\_idx) not in used\_pairs:

pairs\_to\_try.append((new\_idx, old\_idx))

if not pairs\_to\_try:

n = len(new\_sentences)

for i in range(n):

for j in range(i+1, n):

if (i, j) not in used\_pairs:

pairs\_to\_try.append((i, j))

#### **归结逻辑**

resolvent, i\_lit\_idx, j\_lit\_idx = resolve(sentence\_i, sentence\_j)if resolvent is not None:

used\_pairs.add((i, j))

if resolvent not in new\_sentences:

new\_sentences.append(resolvent)

newly\_added\_indices.append(len(new\_sentences) - 1)

1. **创新点&优化**

通过 newly\_added\_indices 来记录新生成的子句，并优先使用新添加的子句进行归结。具体步骤如下：

* 在归结过程中，会先检查是否有新添加的子句（存储在 newly\_added\_indices 列表中）。
* 如果有新的子句，它们会优先和已有的子句进行归结，尝试尽早使用新生成的子句进行归结，从而加快归结过程。

if newly\_added\_indices:

for new\_idx in newly\_added\_indices:

for old\_idx in range(len(new\_sentences)):

if new\_idx != old\_idx and (new\_idx, old\_idx) not in used\_pairs and (old\_idx, new\_idx) not in used\_pairs:

pairs\_to\_try.append((new\_idx, old\_idx))

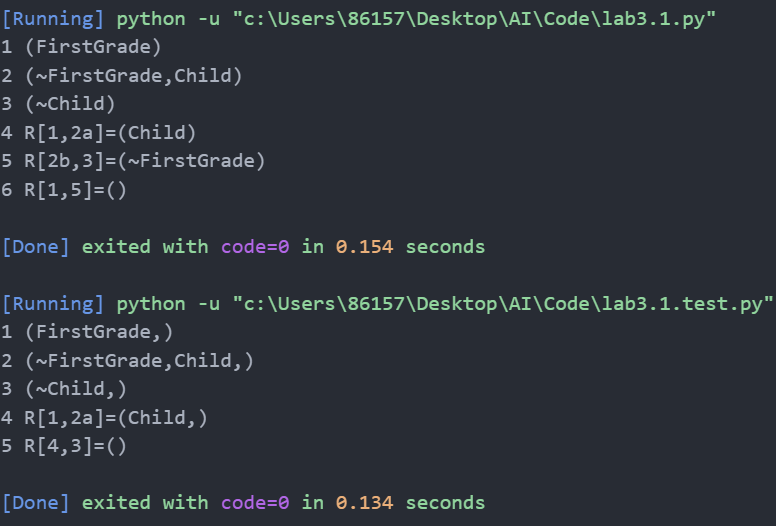
# **效果：与理论相契合，符合我们的思考逻辑**

**简单高效（测试样例分析见结果）**

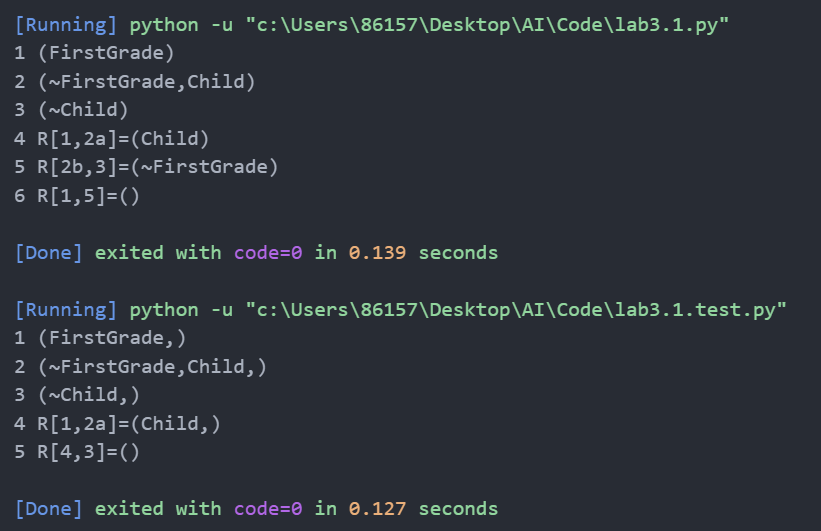
# **实验结果及分析**

1. 实验结果展示示例（可图可表可文字，尽量可视化）

图一：



图二：

（2） 评测指标展示及分析（可分析运行时间等）

如图一图二样例所示：通过 newly\_added\_indices 来记录新生成的子句，并优先使用新添加的子句进行归结，对比修改之前的程序，运行速度加快。  
时间复杂度：O(n^2×m)

空间复杂度：O(n^2+n×m)

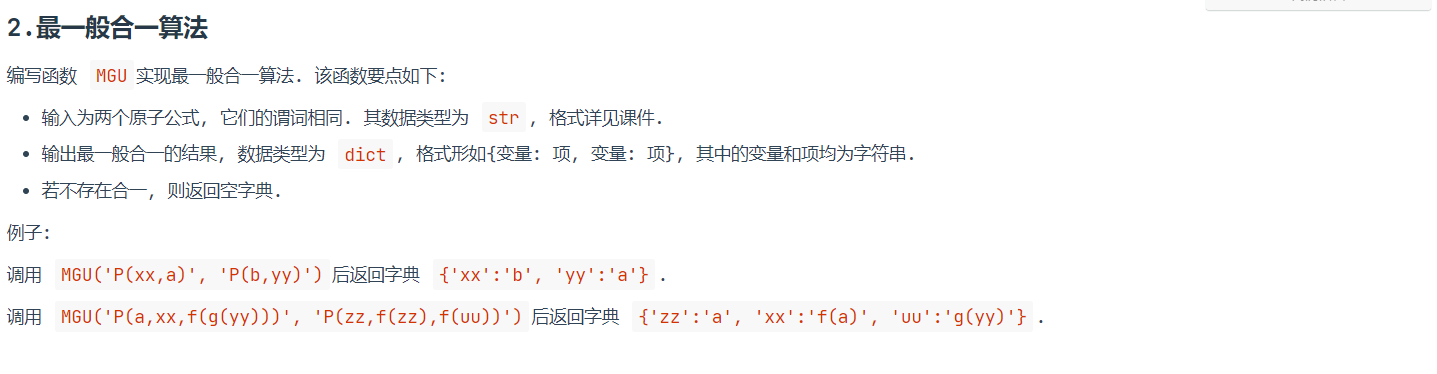
其中 n 是子句的数量，m 是所有不同文字的总数。

# **参考资料**

1. 部分debug及优化建议参考了ai大模型的建议。
2. 阅读参考了以下文章： **[用python做归结演绎推理\_python基础归结-CSDN博客](https://blog.csdn.net/weixin_39724441/article/details/111526296?ops_request_misc=%7B%22request%5Fid%22%3A%221a554c5604a12a6c804c50ec3a5cc06f%22%2C%22scm%22%3A%2220140713.130102334.pc%5Fall.%22%7D&request_id=1a554c5604a12a6c804c50ec3a5cc06f&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~first_rank_ecpm_v1~rank_v31_ecpm-2-111526296-null-null.142^v102^pc_search_result_base6&utm_term=python%E5%BD%92%E7%BB%93%E6%BC%94%E7%BB%8E%E6%8E%A8%E7%90%86&spm=1018.2226.3001.4187)**

**[【人工智能】鲁滨逊归结原理-Python实现\_归结原理实验-CSDN博客](https://blog.csdn.net/qq_41626672/article/details/135885409?ops_request_misc=&request_id=&biz_id=102&utm_term=python%E5%BD%92%E7%BB%93%E6%BC%94%E7%BB%8E%E6%8E%A8%E7%90%86&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-1-135885409.142^v102^pc_search_result_base6&spm=1018.2226.3001.4187)  
二、 MGU.py（第三周作业二）：**

# **实验题目**

****

# **实验内容**

1. **算法原理**
2. 解析原子公式

parse\_atom(atom): 提取谓词和参数列表。

parse\_terms(params\_str): 解析参数列表，支持嵌套结构。

1. 变量以及复合项判断

is\_variable(term): 判断是否为变量（以小写字母开头，且非复合项）。

is\_compound\_term(term): 判断是否为复合项（包含括号）。

1. 变量替换

apply\_replace(term, replace): 递归应用替换，将变量替换为具体值或复合项。

1. 合一算法

unify(x, y, replace):

若 x == y，直接返回替换映射 replace。

若 x 或 y 是变量，使用 unify\_variable 进行合一。

若 x 和 y 是复合项，则递归合一其参数。

1. 变量合一逻辑

若变量已在替换映射 replace 中，则递归合一。

若变量出现在 term 中（避免循环替换），返回 None。

否则，将 var -> term 存入 replace。

1. MGU 计算

MGU(atom1, atom2): 计算两个原子公式的最一般合一。

若谓词不同或参数数目不同，返回 {}。

逐个参数调用 unify 进行合一，返回最终替换映射。

**（2）关键代码展示（可选）**

# 最一般合一（MGU）核心逻辑

def unify(*x*, *y*, *replace*):

    if replace is None:

        return None

    if x == y:

        return replace

    if is\_variable(x):

        return unify\_variable(x, y, replace)

    if is\_variable(y):

        return unify\_variable(y, x, replace)

    if is\_compound\_term(x) and is\_compound\_term(y):

        xf, xp = parse\_atom(x)

        yf, yp = parse\_atom(y)

        if xf != yf or len(xp) != len(yp):

            return None

        for i in range(len(xp)):

            replace = unify(xp[i], yp[i], replace)

            if replace is None:

                return None

        return replace

    return None

# 计算两个原子公式的最一般合一（MGU）

def MGU(*atom1*, *atom2*):

    pred1, params1 = parse\_atom(atom1)

    pred2, params2 = parse\_atom(atom2)

    if pred1 != pred2 or len(params1) != len(params2):

        return {}

    replace = {}

    for p1, p2 in zip(params1, params2):

        replace = unify(p1, p2, replace)

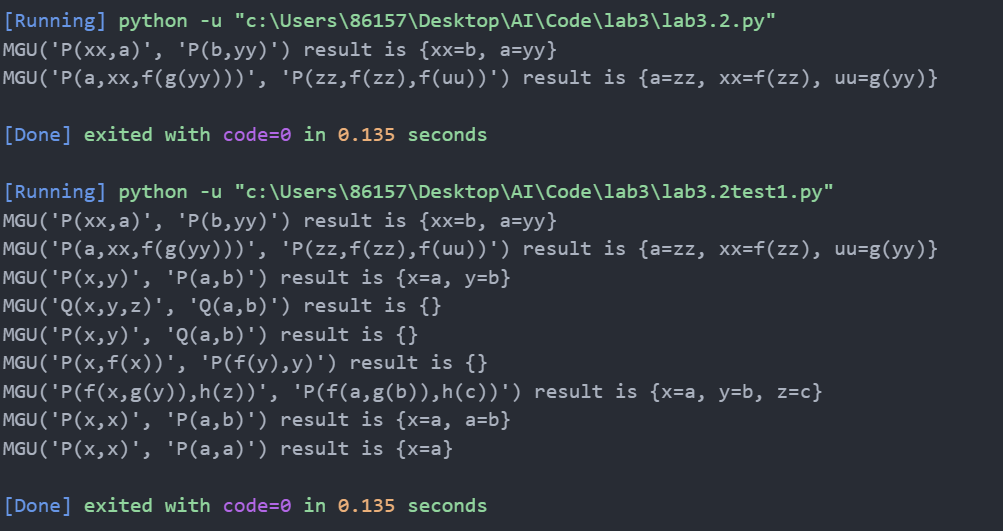
        if replace is None:

            return {}

    return replace

# **实验结果及分析**

**（1）实验结果展示示例  
第一个示例为实验要求，第二个示例为其他更多复杂示例。**

****

**（2） 评测指标展示及分析**

1. 时间复杂度分析

最优情况（变量直接匹配） O(n)

一般情况（递归合一） O(n²)

最坏情况（深度嵌套复合项） O(n³)

1. 空间复杂度分析

最优情况（无递归） O(n)

一般情况（递归展开） O(n²)

最坏情况（深度递归 + 复杂表达式） O(n²)

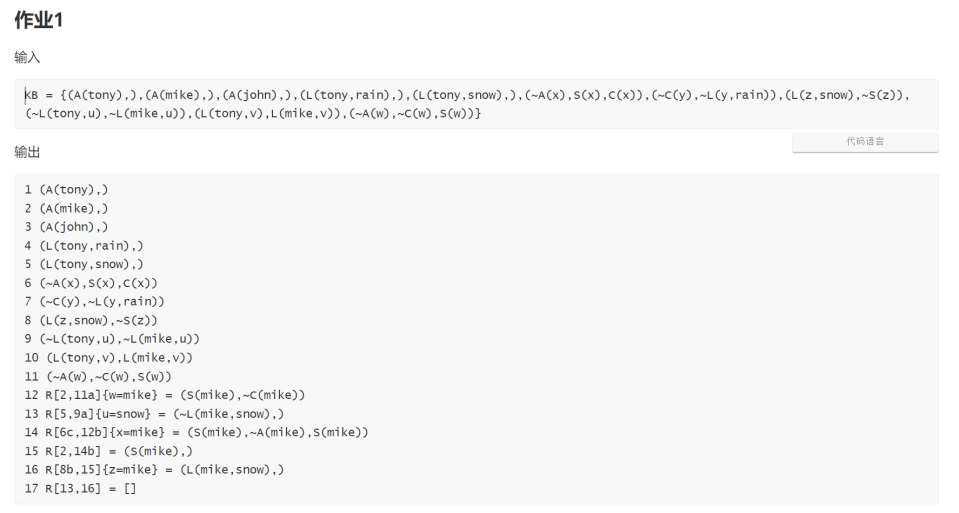
# **参考资料**

1. **部分debug，优化建议参考了ai大模型的建议。**
2. **阅读参考了以下文章：**

[合一算法的Python实现--人工智能\_unify算法-CSDN博客](https://blog.csdn.net/Zhangguohao666/article/details/105571115?ops_request_misc=%7B%22request%5Fid%22%3A%2217bb207511952375bdfa3848249e48e0%22%2C%22scm%22%3A%2220140713.130102334..%22%7D&request_id=17bb207511952375bdfa3848249e48e0&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduend~default-1-105571115-null-null.142^v102^pc_search_result_base6&utm_term=python%E6%9C%80%E4%B8%80%E8%88%AC%E5%90%88%E4%B8%80%E7%AE%97%E6%B3%95&spm=1018.2226.3001.4187)  
[合一算法的Python实现--人工智能\_python实现最一般合一-CSDN博客](https://blog.csdn.net/weijiancheng999/article/details/106941462)  
[vscode中python中文输出乱码问题 - Arlong - SegmentFault 思否](https://segmentfault.com/a/1190000044782749)

# 三、FOL.py（第四周）

# 实验题目



# 实验内容

1. 算法原理
2. generate\_identifier(literal\_idx, clause\_idx, clause\_size)

生成子句和字面量的唯一标识符，用于在归结过程中标识特定的子句和字面量。

1. are\_complementary(lit1, lit2)

判断两个字面量是否互补，即一个字面量是否是另一个字面量的否定。

1. resolve\_clauses(clause1, clause2, idx1, idx2)

对两个子句进行归结，移除互补的字面量，生成新的子句。

1. format\_resolution\_sequence(new\_clause, id1, id2, substitutions)

生成归结步骤的字符串表示形式，包括使用的子句标识符和替换映射。

1. 5substitute\_clause(clause, substitutions)

对子句中的字面量应用替换映射，生成新的子句。

1. resolution(KB)

执行归结过程，从知识库中生成新的子句，直到找到空子句或无法生成新的子句。

1. 7update\_num(num, steps, useful\_steps, init\_size)
2. 更新步骤编号，确保步骤编号在简化后的步骤列表中保持一致。. extract\_parents(seq)
3. 从归结步骤字符串中提取父步骤的编号。. reassign\_sequence(seq, old\_num1, old\_num2, new\_num1, new\_num2)

重新分配步骤编号，更新归结步骤字符串中的编号。

1. simp\_steps(steps, init\_size)

简化归结步骤，去除冗余的步骤，只保留必要的归结步骤。

1. solve(KB)

解决知识库中的归结问题，执行归结过程并简化步骤，返回最终的归结步骤列表

1. 关键代码展示（可选）

# 归结过程

def resolution(*KB*):

    all\_clauses = list(KB)

    support\_list = [all\_clauses[-1]]

    result = []

    processed\_pairs = set()

    while True:

        new\_clauses = []

        for i in range(len(all\_clauses)):

            for j in range(i + 1, len(all\_clauses)):

                if i == j:

                    continue

                clause1, clause2 = all\_clauses[i], all\_clauses[j]

                if (clause1, clause2) in processed\_pairs:

                    continue

                if clause2 not in support\_list and clause1 not in support\_list:

                    continue

                for lit\_idx1 in range(len(clause1)):

                    for lit\_idx2 in range(len(clause2)):

                        lit1, lit2 = clause1[lit\_idx1], clause2[lit\_idx2]

                        if not are\_complementary(lit1, lit2):

                            continue

                        lit1\_clean = lit1.replace('~', '')

                        lit2\_clean = lit2.replace('~', '')

                        mgu\_dict = MGU([lit1\_clean], [lit2\_clean])

                        if mgu\_dict is None:

                            continue

                        clause1\_sub = substitute\_clause(clause1, mgu\_dict)

                        clause2\_sub = substitute\_clause(clause2, mgu\_dict)

                        new\_clause = resolve\_clauses(clause1\_sub, clause2\_sub, lit\_idx1, lit\_idx2)

                        if new\_clause in all\_clauses or new\_clause in new\_clauses:

                            continue

                        processed\_pairs.add((clause1, clause2))

                        id1 = generate\_identifier(lit\_idx1, i, len(clause1))

                        id2 = generate\_identifier(lit\_idx2, j, len(clause2))

                        seq = format\_resolution\_sequence(new\_clause, id1, id2, mgu\_dict)

                        result.append(seq)

                        new\_clauses.append(new\_clause)

                        if new\_clause == ():

                            return result

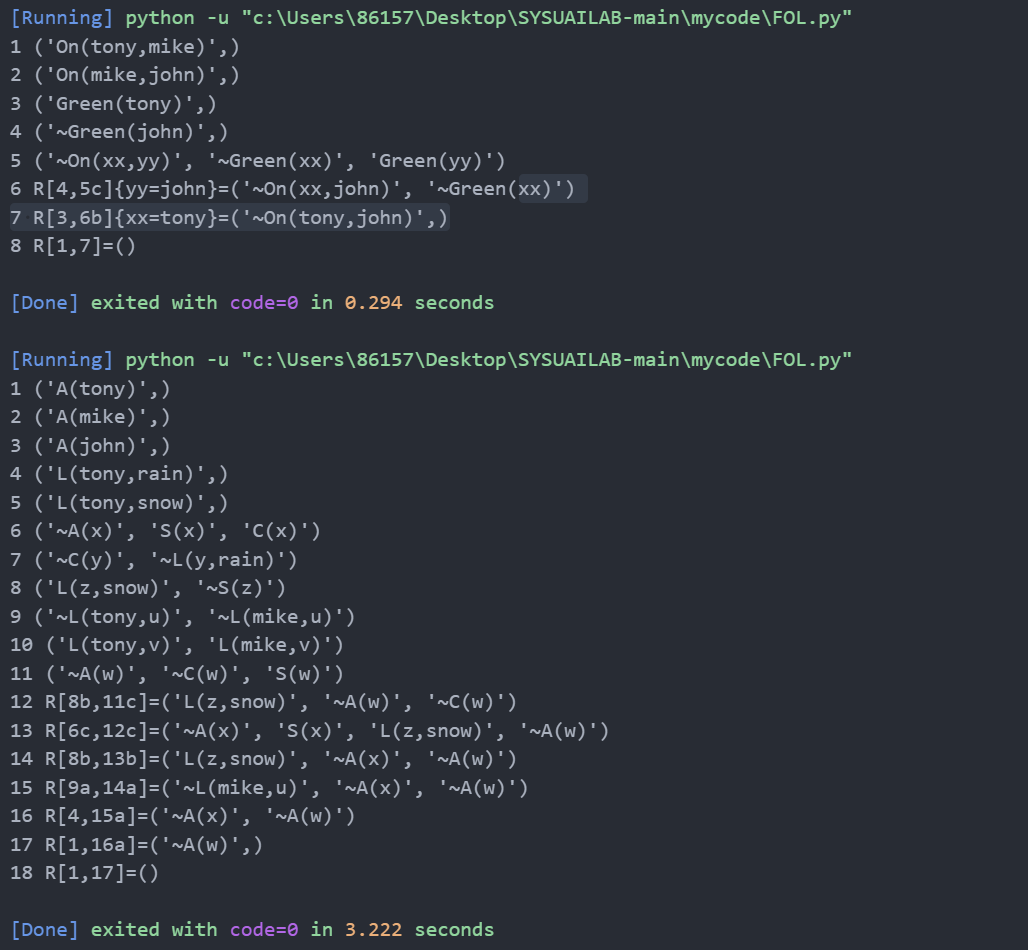
        all\_clauses.extend(new\_clauses)

        support\_list.extend(new\_clauses)

1. 创新点&优化（如果有）

代码实现了****支持集策略（Set of Support Strategy）****。这种策略优先选择包含推理后句子（即支持集中的句子）的子句进行归结，从而减少了不必要的归结步骤，提高了推理效率。这种方法特别适用于处理大型知识库，因为它可以显著减少搜索空间，避免无意义的归结操作，从而更快地找到矛盾或证明知识库的一致性。详见上述函数。

# 实验结果及分析

****

1. 代码在处理第二个样例时比较慢，主要是因为归结演绎推理算法的组合爆炸问题。具体来说：

子句数量多：随着知识库的增大，需要检查的子句对数量呈指数级增长。

MGU 计算复杂：每次归结时都需要计算最一般合一（MGU），这在复杂字面量下非常耗时。

重复子句检查：虽然有去重机制，但检查本身也会消耗大量时间

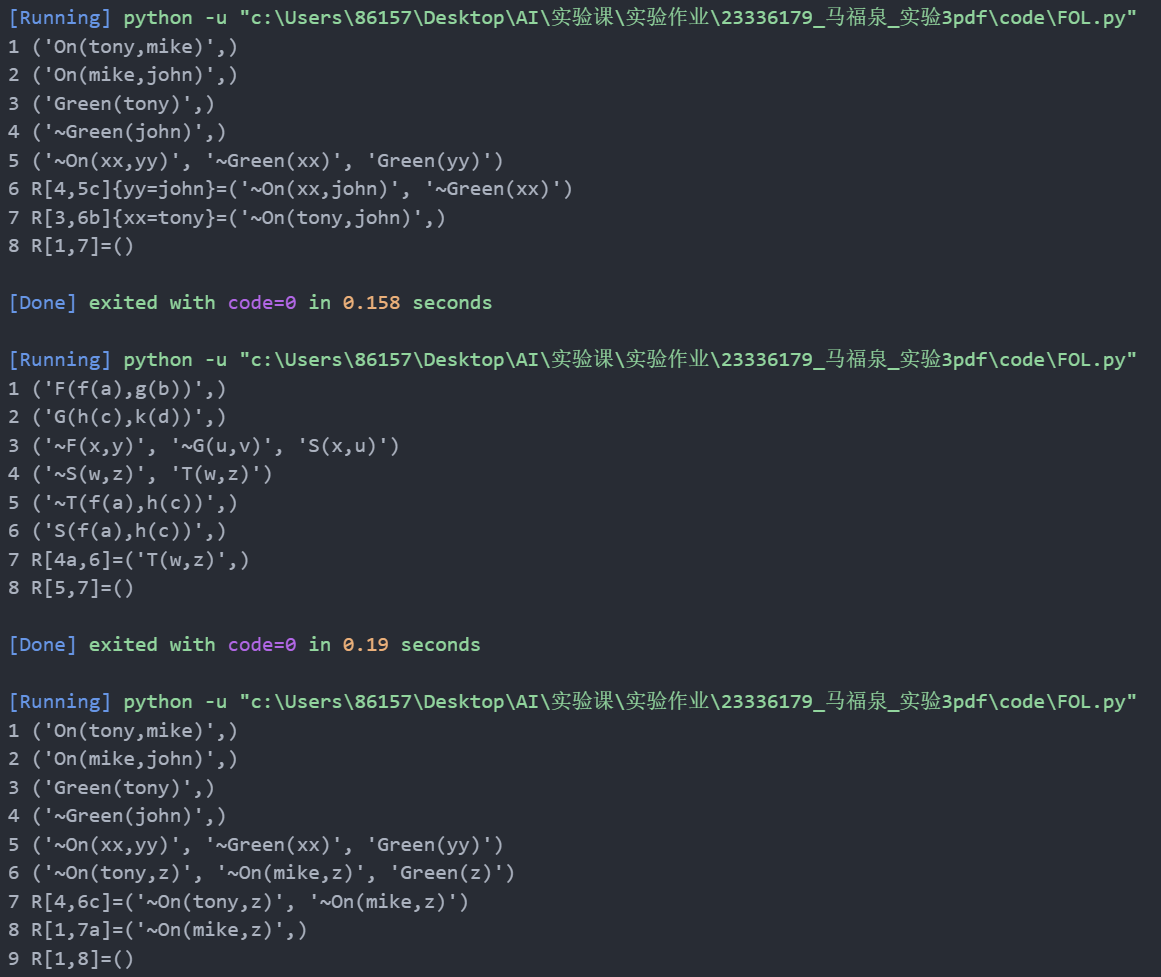
1. 优化思路：

启发式搜索：优先处理较短的子句，减少不必要的归结。

子句索引：快速定位互补字面量，减少比较次数。

剪枝策略：避免重复处理相同的子句对，减少计算量。

优化 MGU：减少 MGU 计算的开销。

1. 更多样例如下：
2. 

# 参考资料

1. **部分debug，代码优化建议参考了ai大模型的建议。**
2. **阅读参考了以下文章：**

[用Python实现命题逻辑归结推理系统--人工智能\_python基于谓词逻辑的归结原理代码-CSDN博客](https://blog.csdn.net/Zhangguohao666/article/details/105471307?ops_request_misc=%7B%22request%5Fid%22%3A%220fec3c97df364d3637bfbd330f97c091%22%2C%22scm%22%3A%2220140713.130102334..%22%7D&request_id=0fec3c97df364d3637bfbd330f97c091&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduend~default-1-105471307-null-null.142^v102^pc_search_result_base6&utm_term=%E4%B8%80%E9%98%B6%E9%80%BB%E8%BE%91%E5%BD%92%E7%BB%93python&spm=1018.2226.3001.4187)  
[基于Python实现的Horn子句归结-CSDN博客](https://blog.csdn.net/newlw/article/details/125971021?ops_request_misc=%7B%22request%5Fid%22%3A%22eaaaf22fea9c0aae3381242e5beee834%22%2C%22scm%22%3A%2220140713.130102334.pc%5Fall.%22%7D&request_id=eaaaf22fea9c0aae3381242e5beee834&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~first_rank_ecpm_v1~rank_v31_ecpm-13-125971021-null-null.142^v102^pc_search_result_base6&utm_term=%E4%B8%80%E9%98%B6%E9%80%BB%E8%BE%91%E5%BD%92%E7%BB%93python&spm=1018.2226.3001.4187)