人工智能实验报告

课程名称：**Artificial Intelligence**

专业（方向）：信息与计算科学

学号：23323035

姓名：崔行健

实验题目：归结原理实验

1. 实验内容

1.1 算法原理

1.2 伪代码

1.3 关键代码展示

2 实验结果及分析

人工智能实验报告

课程名称：**Artificial Intelligence**

专业（方向）：信息与计算科学

学号：23323035

姓名：崔行健

实验题目：归结原理实验

---

1. 实验内容

1.1 算法原理

一阶逻辑归结算法的核心是通过消解互补文字对推导出空子句，从而证明查询的否定与知识库矛盾。具体步骤如下：

1. **子句标准化**：将知识库和查询转换为析取式，统一变量命名避免冲突。
2. **寻找互补对**：遍历子句对，若存在文字`L1`和`~L2`（或反之）且可通过合一（Unification）匹配，则进行消解。
3. **最一般合一（MGU）**：为互补文字对中的变量和常量建立替换规则，使两者结构一致。
4. **生成新子句**：应用MGU替换后，合并两个子句并删除互补对，生成新子句。若新子句为空，则证明完成。

1.2 伪代码

MGU算法伪代码

```
function MGU(args1, args2):
    if len(args1) != len(args2): return None
    unification = {}
    for i in 0 to len(args1)-1:
        val1 = args1[i], val2 = args2[i]
        if val1 == val2: continue
        if val1是变量且val2是常量:
            unification[val1] = val2
        elif val2是变量且val1是常量:
            unification[val2] = val1
        else: return None  # 无法合一
    return unification
```

归结算法伪代码

```
function resolution(KB, query):
    clauses = KB + [query]
    while True:
        生成所有可能的子句对(C1, C2)
        for each (C1, C2) in clauses:
            for L1 in C1, L2 in C2:
                if L1和L2是互补对:
                    mgu = MGU(L1, L2)
                    if mgu存在:
                        应用mgu到C1和C2，生成新子句C_new
                        if C_new为空: 返回"证明成功"
                        if C_new不在clauses中: 添加到clauses
        if 无新子句生成: 返回"证明失败"
```

1.3 关键代码展示

互补判断

```
def is_complement(self, literal1, literal2):
    pred1 = literal1.split('(')[0].lstrip('~')
    pred2 = literal2.split('(')[0].lstrip('~')
    return (pred1 == pred2) and (
        (literal1.startswith('~') ^ literal2.startswith('~'))
    )
```

MGU实现

```
def mgu(self, literal1, literal2):
    args1 = self.get_arguments(literal1)
    args2 = self.get_arguments(literal2)
    if len(args1) != len(args2): return None
    unification = {}
    for val1, val2 in zip(args1, args2):
        if val1 == val2: continue
        if self.is_variable(val1) and self.is_constant(val2):
            unification[val1] = val2
        elif self.is_variable(val2) and self.is_constant(val1):
            unification[val2] = val1
        else: return None
    return unification
```

归结步骤生成

```
def resolve(self, clause1, clause2, lit1_idx, lit2_idx):
    new_clause = list(clause1[:lit1_idx] + clause1[lit1_idx+1:] +
                      clause2[:lit2_idx] + clause2[lit2_idx+1:])
    return tuple(OrderedDict.fromkeys(new_clause))  # 去重
```

主归结循环

```
    def resolution(self):
        clauseset = self.clauses
        clauseset = list(OrderedDict.fromkeys(clauseset))        # 去重
        step = ['归结顺序:'] + self.clauses      #将0位置补充元素，确保编号和列表索引对应
        while True:
            clauseset_len = len(clauseset)
            new_clauseset = []

            # 遍历子句集
            for clause1_index in range(clauseset_len):
                for clause2_index in range (clause1_index + 1, clauseset_len):
                    clause1 = clauseset[clause1_index]
                    clause2 = clauseset[clause2_index]
                    clause1_len = len(clause1)
                    clause2_len = len(clause2)
                    for literal1_index in range(clause1_len):
                        for literal2_index in range(clause2_len):
                            literal1 = clause1[literal1_index]
                            literal2 = clause2[literal2_index]

                            # 判断是否为互补对
                            if self.is_complement(literal1, literal2):
                                if DEBUG:
                                    print(literal1, literal2, "is complement")

                                # 最一般合一项
                                unification = self.mgu(literal1, literal2)
                                if unification == None:
                                    break
                                if DEBUG:   # 未实现功能：若违法，则跳出循环
                                    if unification == False:
                                        print("谓词的参数个数必须相同")
```

```
                                    return False

                                # 最一般合一替换
                                newclause1 = self.substitute(unification,
clause1)
                                newclause2 = self.substitute(unification,
clause2)
                                newclause =  self.resolve(newclause1,
newclause2, literal1_index, literal2_index)

                                # 检查是否为新子句
                                if newclause in clauseset or newclause in
new_clauseset:
                                    break
                                new_clauseset.append(newclause)

                                # 记录步骤
                                index1 = self.index(literal1_index,
clause1_index, clause1_len)
                                index2 = self.index(literal2_index,
clause2_index, clause2_len)
                                sequence = self.sequence(newclause,
unification, index1, index2)

                                step.append(sequence)

                                # 发现空子句则成功
                                if newclause == ():
                                    self.step = step
                                    return
                            literal2_index += 1
                        literal1_index += 1
            if new_clauseset:
                clauseset += new_clauseset
            else:
                return False
```

2. 实验结果及分析

测试案例运行结果

输入文件`test1.txt`内容:

```
KB:
A(tony)
A(mike)
A(john)
L(tony, rain)
L(tony, snow)
(~A(x), S(x), C(x))
(~C(y), ~L(y, rain))
(L(z, snow), ~S(z))
(~L(tony, u), ~L(mike, u))
(L(tony, v), L(mike, v))
QUERY:
(~A(w), ~C(w), S(w))
```

运行后输出：

```
归结顺序:
1 ('A(tony)',)
2 ('A(mike)',)
3 ('A(john)',)
4 ('L(tony,rain)',)
5 ('L(tony,snow)',)
6 ('~A(x)', 'S(x)', 'C(x)')
7 ('~C(y)', '~L(y,rain)')
8 ('L(z,snow)', '~S(z)')
9 ('~L(tony,u)', '~L(mike,u)')
10 ('L(tony,v)', 'L(mike,v)')
11 ('~A(w)', '~C(w)', 'S(w)')
12 R[2,11a]{w=mike} = ('~C(mike)', 'S(mike)')
13 R[2,6a]{x=mike} = ('S(mike)', 'C(mike)')
14 R[8a,9b]{z=mike,u=snow} = ('~S(mike)', '~L(tony,snow)')
15 R[13b,12a] = ('S(mike)',)
16 R[5,14b] = ('~S(mike)',)
17 R[16,15] = ()
```

输入文件`test2.txt`内容：

```
KB:
GradStudent(sue)
(~GradStudent(x), Student(x))
(~Student(x), HardWorker(x))
QUERY:
~HardWorker(sue)
```

运行后输出：

```
归结顺序:
1 ('GradStudent(sue)',)
2 ('~GradStudent(x)', 'Student(x)')
3 ('~Student(x)', 'HardWorker(x)')
4 ('~HardWorker(sue)',)
5 R[3b,4]{x=sue} = ('~Student(sue)',)
6 R[1,2a]{x=sue} = ('Student(sue)',)
7 R[6,5] = ()
```

输入文件`test3.txt`内容：

```
KB:
On(aa,bb)
On(bb,cc)
Green(aa)
~Green(cc)
QUERY:
(~On(x,y), ~Green(x), Green(y))
```

运行后输出：

```
归结顺序:
1 ('On(aa,bb)',)
2 ('On(bb,cc)',)
3 ('Green(aa)',)
4 ('~Green(cc)',)
5 ('~On(x,y)', '~Green(x)', 'Green(y)')
6 R[4,5c]{y=cc} = ('~On(x,cc)', '~Green(x)')
7 R[3,5b]{x=aa} = ('~On(aa,y)', 'Green(y)')
8 R[2,6a]{x=bb} = ('~Green(bb)',)
9 R[1,7a]{y=bb} = ('Green(bb)',)
10 R[9,8] = ()
```

结果分析

1. **正确性**：成功推导出空子句，证明原查询`~B(tony)`与知识库矛盾，验证算法正确性。
2. **步骤清晰性**：输出按步骤展示归结过程，符合参考文档的格式要求。
3. **局限性**：当前实现仅支持变量与常量的合一，未处理含函数项或多元谓词的复杂场景。

核心代码说明：完整代码见附件，关键方法已在上文展示。