中山大学计算机学院

人工智能

本科生实验报告

（2022学年春季学期）

课程名称：Artificial Intelligence

教学班级	计科1班	专业（方向）	计算机科学与技术
学号	21307035	姓名	邓栩瀛

一、实验题目

编写程序，实现一阶逻辑归结算法，并用于求解给出的三个逻辑推理问题，要求输出按照如下格式：

1. $(P(x), Q(g(x)))$
2. $(R(a), Q(z), \neg P(a))$
3. $R[1a, 2c]X = a(Q(g(a)), R(a), Q(z))$

… …
“R” 表示归结步骤.
“1a” 表示第一个子句(1-th)中的第一个 (a-th)个原子公式，即P(x).
“2c”表示第二个子句(1-th)中的第三个 (c-th)个原子公式，即¬P(a).
“1a”和“2c”是冲突的，所以应用最小合一{X = a}

二、实验内容

1.算法原理

归结算法：

- 将α取否定，加入到KB当中
- 将更新的KB转换为clausal form得到S
- 反复调用单步归结
- 如果得到空子句，即S⊢()，说明KB∧¬α不可满足,算法终止,可得KB⊨ α
- 如果一直归结直到不产生新的子句，在这个过程中没有得到空子句，则KB⊨α不成立

单步归结

- 使用MGU算法从两个子句中得到相同的原子，及其对应的原子否定
- 去掉该原子并将两个子句合为一个，加入到S子句集合中

合一算法MGU

- $k = 0, \sigma_0 = \{\}, S_0 = \{f, g\}$

- 如果S_k中的公式等价，则返回σ_k作为最一般合一的结果，否则找出S_k中不匹配的项$D_k = \{e_1, e_2\}$
- 如果$e_1 = V$是变量，$e_2 = t$是一个不包含变量的项，将$V = t$通过合一步骤添加到集合σ_k中，并更新为σ_{k+1}，并将S_k中的其它变量V替换成t，得到$S_{k+1}, k = k + 1$，转到第二步，否则合一失败

2.伪代码

```python
def resolution(clauses):
    """
    解析算法的实现函数
    参数:
    clauses: 一个包含多个子句的列表，每个子句由多个文字组成，文字用字符串表示，且可为否定形式（用"~"表示）。
    返回值:
    解析算法的输出，以布尔值表示是否存在一个空子句，即原命题的否定是否可满足。
    """
    new = set()  # 用集合来存储新产生的子句
    while True:
        n = len(clauses)
        pairs = [(clauses[i], clauses[j]) for i in range(n) for j in range(i+1, n)]
        for p, q in pairs:
            resolvents = resolve(p, q)
            if [] in resolvents:  # 如果有空子句，返回 True
                return True
            new.update(resolvents)
        if new.issubset(clauses):  # 如果没有新的子句了，返回 False
            return False
        clauses = clauses.union(new)  # 将新产生的子句加入原始子句集合中

def resolve(p, q):
    """
    计算两个子句的合取式中的所有归结项
    参数:
        p: 第一个子句，由多个文字组成，文字用字符串表示，且可为否定形式（用"¬"表示）。
        q: 第二个子句，由多个文字组成，文字用字符串表示，且可为否定形式（用"¬"表示）。
    返回值:
        一个列表，包含所有归结项，每个归结项是一个由多个文字组成的子句。
    """
    resolvents = []
    for literal in p:
        if "¬" in literal:  # 如果当前文字为否定形式，则去除"~"符号后与 q 中的文字匹配
            resolvents += [[x for x in p if x ≠ literal] + [y for y in q if y ≠literal[1:]]]
        else:  # 否则直接与 q 中的文字匹配
            resolvents += [[x for x in p if x ≠ literal] + [y for y in q if y ≠"¬"+literal]]
    return resolvents

def is_negative(literal):
    # 判断一个谓词或量词表达式是否为否定形式
```

```
    return literal.startswith("¬")

def is_complementary(literal1, literal2):
    """
    判断两个谓词或量词表达式是否互为互补形式
    参数:
        literal1: 第一个谓词或量词表达式，用字符串表示。
        literal2: 第二个谓词或量词表达式，用字符串表示。
    返回值:
        如果两个表达式互为互补形式，返回 True；否则返回 False。
    """
    return (not is_negative(literal1) and literal1[1:] == literal2) or (not
is_negative(literal2) and literal2[1:] == literal1)
```

3.关键代码展示（带注释）

```
def algorithm(clauses, Path, ALPH, count):
    end = False
    '''
    查找列表 clauses 中长度为 2 的元素，然后在其中找到两个元素，
    它们的第一个字典元素的反义相同且值相同，然后输出一个 R[ ] = [] 的字符串。
    如果找到了这样的两个元素，变量 end 将被设置为 True，从而退出循环。
    '''
    index = 0
    while True:
        if end:
            break
        cur = []
        for m in range(len(clauses)):
            if len(clauses[m]) == 2:  # 只有一个谓词公式
                cur.append(clauses[m])
        for m in range(len(cur)):
            for n in range(m + 1, len(cur)):
                key1 = list(cur[m][0].keys())  # 读取谓词
                key2 = list(cur[n][0].keys())
                value1 = list(cur[m][0].values())  # 读取变量或参数
                value2 = list(cur[n][0].values())
                if key1[0] == reverse(key2[0]) and value1[0] == value2[0]:
                    end = True
                    print('R[', end='')
                    print(cur[m][len(cur[m]) - 1], ',', cur[n][len(cur[n]) - 1],
sep='', end='] = []')
                    break
            if end:
                break
        # 依次读取字句，并依次与其它字句进行比较
        parent1 = clauses[index]
```

```
        index = index + 1
        for index1 in range(len(parent1) - 1):# 遍历字句parent1
            if end:# end为true,结束循环
                break
            list_1 = list(parent1[index1].keys())
            i = list_1[0]
            val_1 = list(parent1[index1].values())
            val = val_1[0]# 字句1的参数
            for parent2 in clauses:  # 依次与其它字句进行比较
                if end:# end为true,结束循环
                    break
                str = []
                pos1 = ''
                pos2 = ''
                for index2 in range(len(parent2) - 1):# 遍历字句parent2
                    key_1 = list(parent2[index2].keys())
                    key = key_1[0]
                    value_1 = list(parent2[index2].values())
                    value = value_1[0]# 字句2的参数
                    if key == reverse(i):# 两个句子的谓词互为反义
                        different = 0
                        for pos in value:# 记录参数
                            if pos in val:
                                continue
                            else:
                                different = different + 1
                                pos1 = pos # 记录变量or变量值
                        if different <= 1: # 不多于1个不同点
                            for h in val:
                                if h in value:
                                    continue
                                else:
                                    pos2 = h # 记录变量or变量值
                            flag = False # 判断是否跳出循环
                            if pos1 == '' and pos2 == '':
                                flag = True
                            if len(pos1) == 1 and len(pos2) > 1:# 变量与值的位置互换
                                temp = pos2
                                pos2 = pos1
                                pos1 = temp
                                flag = True
                            if len(pos2) == 1 and len(pos1) > 1:# 跳出循环
                                flag = True
                            if flag:
                                str.append(key)# 记录谓词
                                break
                if not str:# 谓词集为空
                    continue
                else:
                    m = 0
```

```
                for k1 in range(len(parent1) - 1):
                    line_parent1 = list(parent1[k1].keys())#将谓词集列表化
                    if line_parent1[0] == i:
                        break
                    m = m + 1
                n = 0
                for k2 in range(len(parent2) - 1):
                    line_parent2 = list(parent2[k2].keys())#将谓词集列表化
                    if line_parent2[0] == key:
                        break
                    n = n + 1
                # 检查now是否已经在Path中，减少重复字句
                now = []
                now.append(parent1[len(parent1) - 1])#记录字句的序号
                now.append(parent2[len(parent2) - 1])#记录字句的序号
                now.sort()
                now_flag = True
                for now_1 in Path:
                    if now_1 == now:
                        now_flag = False
                        break
                if not now_flag:
                    break
                Path.append(now)# 若now不在Path中，则添加到Path
                '''
                深拷贝创建两个新列表 new_line1 和 new_line2
                遍历这两个列表并删除其中与 parent1 和 parent2 中最后一个元素相同的元素
                如果 pos1 和 pos2 都不为空，则遍历 new_line1 和 new_line2 列表中的元
素，
                并将其中所有值为 pos2 的位置替换为 pos1。
                '''
                new_line1 = copy.deepcopy(parent1)
                new_line2 = copy.deepcopy(parent2)
                for index4 in range(len(new_line1) - 1):
                    if new_line1[index4] == parent1[index1]:
                        new_line1.remove(new_line1[index4])
                        break
                for index4 in range(len(new_line2) - 1):
                    if new_line2[index4] == parent2[index2]:
                        new_line2.remove(new_line2[index4])
                        break
                if pos1 != '' and pos2 != '':
                    for index3 in range(len(new_line1) - 1):
                        temp_list = list(new_line1[index3].values())
                        for k in range(len(temp_list[0])):
                            if temp_list[0][k] == pos2:
                                temp_list[0][k] = pos1
                    for index3 in range(len(new_line2) - 1):
                        temp_list = list(new_line2[index3].values())
                        for k in range(len(temp_list[0])):
```

```
                            if temp_list[0][k] == pos2:
                                temp_list[0][k] = pos1
                    '''
                    n1_flag/n2_flag设置为False，表示还没有找到符合要求的子句
                    遍历clauses列表中的每个子句s，如果s的长度与new_line1/new_line2不同，则跳过该子句
                    否则，遍历s中的元素，查看它们是否都在new_line1/new_line2中出现过。
                    如果是，则将n1_flag/n2_flag设置为True，并跳出循环。
                    finally，如果找到了符合要求的子句，则n1_flag/n2_flag的值为True，否则为False
                    目的：避免选择已经包含在clauses中的某个字句作为新的解，从而避免重复的计算
                    '''
                    n1_flag = False
                    for s in clauses:
                        if len(s) != len(new_line1):
                            continue
                        num = 0
                        for index5 in range(len(s) - 1):
                            for index6 in range(len(s) - 1):
                                if s[index6] == new_line1[index5]:
                                    num = num + 1
                                    break
                        if num == len(s) - 1:
                            n1_flag = True
                            break
                    n2_flag = False
                    for s in clauses:
                        if len(s) != len(new_line2):
                            continue
                        num = 0
                        for index5 in range(len(s) - 1):
                            for index6 in range(len(s) - 1):
                                if s[index6] == new_line2[index5]:
                                    num = num + 1
                                    break
                        if num == len(s) - 1:
                            n2_flag = True
                            break
                    if new_line1 == new_line2:
                        new_line2 = []
                    if len(new_line1) > 1 and len(new_line2) > 1:
                        break
                    if n1_flag or n2_flag:
                        break
                    # 以下为归结结果的输出
                    print('R[', end='')
                    print(parent1[len(parent1) - 1], end='')#最后一位记录的是字句的序号
                    if len(parent1) > 2:# 一个字句中超过一个谓词，则需要用a、b、c等显示
                        print(ALPH[m], end='')
                    print(',', end='')
```

```
                print(parent2[len(parent2) - 1], end='')#最后一位记录的是字句的序号
                if len(parent2) > 2:# 一个字句中超过一个谓词，则需要用a、b、c等显示
                    print(ALPH[n], end='')
                print(']', end='')
                # pos1和pos2非空，输出变量赋值结果
                if pos1 != '' and pos2 != '':
                    print('(', end='')
                    print(pos2, '=', pos1, sep='', end='')
                    print(')', end='')
                print(' = ', end='')
                #输出变量赋值后的字句
                if len(new_line1) > 1:
                    new_line1[len(new_line1) - 1] = count
                    count = count + 1#count为计数器
                    clauses.append(new_line1)#将新子句添加到clauses中
                if len(new_line2) > 1:
                    new_line2[len(new_line2) - 1] = count
                    count = count + 1
                    clauses.append(new_line2)
            '''
            根据两个列表的长度输出它们的归结结果，如果两个列表的长度都小于等于1，则输出空列表
            如果其中一个列表的长度大于1，则遍历该列表中的元素，输出它们的归结结果
            作用：归结推理中输出每一步的归结结果
            '''
                  if len(new_line1) <= 1 and len(new_line2) <= 1:
                    print('[]')#结束
                    end = True
                  elif len(new_line1) <= 1:
                    cnt = 0
                    # 归结结果的序号输出
                    # print('(', new_line2[len(new_line2) - 1], ') ', sep='', end='')
                    for index7 in range(len(new_line2) - 1):
                        item = new_line2[index7]
                        k_1 = list(item.keys())
                        v_1 = list(item.values())
                        print(k_1[0], '(', sep='', end='')
                        for m in range(len(v_1[0])):
                            print(v_1[0][m], end='')
                            if m != len(v_1[0]) - 1:
                                print(',', end='')
                        print(')', end='')
                        if cnt != len(new_line2) - 2:
                            print(',', end='')
                        cnt = cnt + 1
                    print('\n', end='')
                elif len(new_line2) <= 1:
                    cnt = 0
                    # print('(', new_line1[len(new_line1) - 1], ') ', sep='', end='')
```

```
                    for index7 in range(len(new_line1) - 1):
                        item = new_line1[index7]
                        k_2 = list(item.keys())
                        v_2 = list(item.values())
                        print(k_2[0], '(', sep='', end='')
                        for m in range(len(v_2[0])):
                            print(v_2[0][m], end='')
                            if m ≠ len(v_2[0]) - 1:
                                print(',', end='')
                        print(')', end='')
                        if cnt ≠ len(new_line1) - 2:
                            print(',', end='')
                        cnt = cnt + 1
                    print('\n', end='')
```

三、实验结果及分析

1.实验结果展示示例

`BlockWorld.txt`

其中`clauses`为：[[{'On': ['aa', 'bb']}, 1], [{'On': ['bb', 'cc']}, 2], [{'Green': ['aa']}, 3], [{'¬Green': ['cc']}, 4], [{'¬On': ['x', 'y']}, {'¬Green': ['x']}, {'Green': ['y']}, 5]]

```
Please enter filename
BlockWorld.txt
5
On(aa,bb)
On(bb,cc)
Green(aa)
¬Green(cc)
(¬On(x,y),¬Green(x),Green(y))
R[3,5b](x=aa) = ¬On(aa,y),Green(y)
R[4,5c](y=cc) = ¬On(x,cc),¬Green(x)
R[4,6b](y=cc) = ¬On(aa,cc)
R[6a,1](y=bb) = Green(bb)
R[7a,2](x=bb) = ¬Green(bb)
R[9,10] = []
```

`GraduateStudent.txt`

其中`clauses`为[[{'GradStudent': ['sue']}, 1], [{'¬GradStudent': ['x']}, {'Student': ['x']}, 2], [{'¬Student': ['x']}, {'HardWorker': ['x']}, 3], [{'¬HardWorker': ['sue']}, 4]]

```
Please enter filename
GraduateStudent.txt
4
GradStudent(sue)
(¬GradStudent(x),Student(x))
(¬Student(x),HardWorker(x))
¬HardWorker(sue)
R[1,2a](x=sue) = Student(sue)
R[3b,4](x=sue) = ¬Student(sue)
R[5,6] = []
```

`AipineClub.txt`

其中`clauses`为[[{'A': ['tony']}, 1], [{'A': ['mike']}, 2], [{'A': ['john']}, 3], [{'L': ['tony', 'rain']}, 4], [{'L': ['tony', 'snow']}, 5], [{'¬A': ['x']}, {'S': ['x']}, {'C': ['x']}, 6], [{'¬C': ['y']}, {'¬L': ['y', 'rain']}, 7], [{'L': ['z', 'snow']}, {'¬S': ['z']}, 8], [{'¬L': ['tony', 'u']}, {'¬L': ['mike', 'u']}, 9], [{'L': ['tony', 'v']}, {'L': ['mike', 'v']}, 10], [{'¬A': ['w']}, {'¬C': ['w']}, {'S': ['w']}, 11]]

```
Please enter filename
AipineClub.txt
11
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))
(¬A(w),¬C(w),S(w))
R[1,6a](x=tony) = S(tony),C(tony)
R[1,11a](w=tony) = ¬C(tony),S(tony)
R[2,6a](x=mike) = S(mike),C(mike)
R[2,11a](w=mike) = ¬C(mike),S(mike)
R[3,6a](x=john) = S(john),C(john)
R[3,11a](w=john) = ¬C(john),S(john)
R[4,7b](y=tony) = ¬C(tony)
R[4,9a](u=rain) = ¬L(mike,rain)
R[5,9a](u=snow) = ¬L(mike,snow)
R[8a,20](z=mike) = ¬S(mike)
R[11c,21](w=mike) = ¬A(mike),¬C(mike)
R[18,6c](x=tony) = ¬A(tony),S(tony)
R[18,12b] = S(tony)
R[21,6b](x=mike) = ¬A(mike),C(mike)
R[22a,2] = ¬C(mike)
R[25a,2] = C(mike)
R[26,27] = []
```

2.评测指标展示及分析（机器学习实验必须有此项，其它可分析运行时间等）

`BlockWorld.txt`

```
Running Time:  0.117268s
```

`GraduateStudent.txt`

```
Running Time:  0.133975s
```

`AipineClub.txt`

```
Running Time:  0.133678s
```