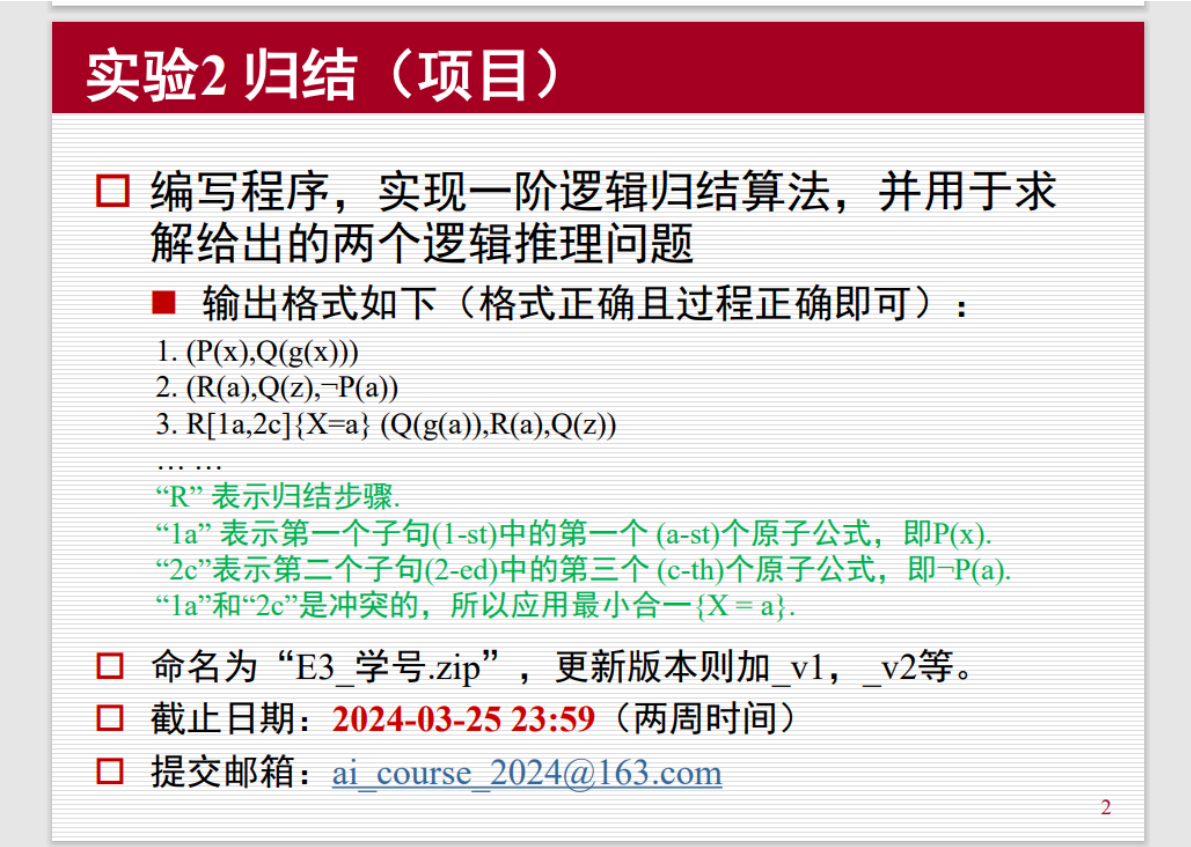
中山大学计算机学院 本科生实验报告（2023学年春季学期）

课程名称：Artificial Intelligence 人工智能

教学班级		专业（方向）	
学号 2233 6173		姓名 罗弘杰	

实验题目



实验2 归结 问题1

- Alpine Club
  - 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))

```
[sysu_hpcedu_302@cpn238 ~/scc22/lsr/mp_linpack/resoluation]$ python main.py
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[2,11a](w=mike) = ¬C(mike),S(mike)
R[2,6a](x=mike) = S(mike),C(mike)
R[5,9a](u=snow) = ¬L(mike,snow)
R[12b,13a] = S(mike)
R[8a,14](z=mike) = ¬S(mike)
R[15,16] = []
```

3

4 一阶逻辑归结算法

- 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
    - 例如(¬Student(x),HardWorker(x))和(HardWorker(sue))合并为(¬Student(sue))

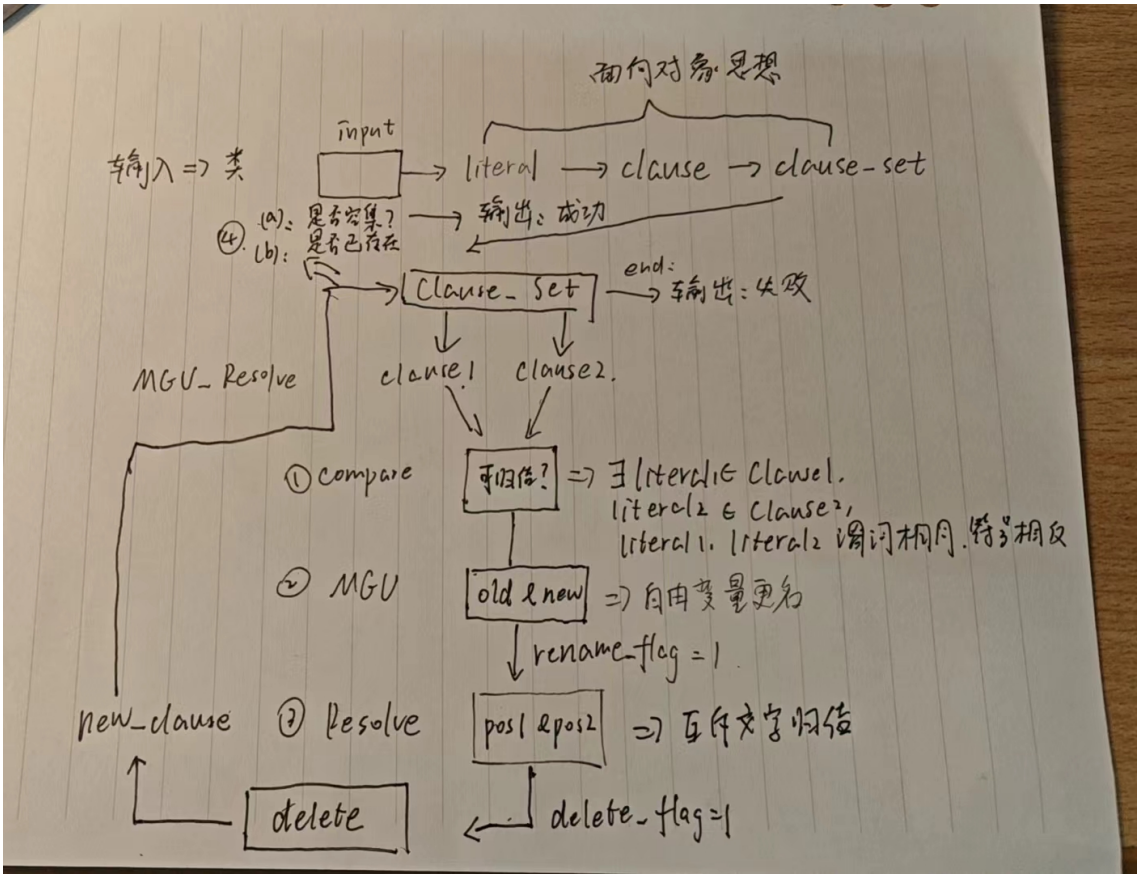
实验内容

1. 算法原理：

归结原理：
将析取式作为子句，子句中的文字都是一个个原子公式；要判断的命题取反加入到子句集合。并列的子句之间是合取的关系；如果两个子句之间存在原子公式是互斥的（谓词相同，作用对象相同，符号相反），则可以把这一对原子公式删除，并将删除后的新公式加入到子句集中。如果加入的子句是空集，那么该命题正确，否则如无法得到空集合，则归结失败

MGU算法：
将两个文字中的自由变量匹配为适合的常量啊，如果可以得到相同的作用对象。说明这两个文字可以被归结

2. 伪代码



3. 关键代码展示（带注释）

```
#MGU且归结函数，输入两个子句，两个子句的编号，输出是否可以合并，以及合并后的子句
def MGU_Resolve(clause_1 : clause, clause_2:clause, a, b, num):  #a,b 是参与
MGU_Resolve的编号
    global counter
    clause1 = copy.deepcopy(clause_1)  #获取副本，避免更改原来的子句
    clause2 = copy.deepcopy(clause_2)
    old = []   #装载旧变量和新变量
    new = []
    delete_flag = 0   #互斥删除标志
    rename_flag = 0   #自由变量更名标志

    #寻找两个子句中的可以被自由变量替换的文字
    for i in range(len(clause1.literals)):
        for j in range(len(clause2.literals)):
            if resolable(clause1.literals[i], clause2.literals[j]):  #如果两个
文字谓词相同且符号相反
```

```
                for k in range(len(clause1.literals[i].variable)):  #寻找子句1
文字中的自由变量
                    if str_is_variable(clause1.literals[i].variable[k]) and
not(str_is_variable(clause2.literals[j].variable[k])): #判断方法：本身是自由而且
对方不是自由
                        rename_flag = 1
                        old.append(clause1.literals[i].variable[k])
                        new.append(clause2.literals[j].variable[k])
                for k in range(len(clause2.literals[j].variable)): #子句2
                    if str_is_variable(clause2.literals[j].variable[k]) and
not(str_is_variable(clause1.literals[i].variable[k])):
                        rename_flag = 1
                        if clause2.literals[j].variable[k] not in old:    #避
免重复添加自由变量
                            old.append(clause2.literals[j].variable[k])
                            new.append(clause1.literals[i].variable[k]) #记录
要改变的变量

    clauses_out = clause([])  #记录生成的子句

    if rename_flag == 1:  #如果有自由变量需要更名
        clause1.rename(old, new)
        clause2.rename(old, new)
        for i in range(len(old)):  #记录更名的变量
            clauses_out.model.append(old[i])
            clauses_out.model.append(new[i])

    #记录删除的元素的位置
    pos1 = []
    pos2 = []

    #搜索两个子句中可以被合并的项，并删除在原子句中的项:方法：遍历两个子句，找到可以合并的
项（谓词相同，变量相同，符号相反），然后删除这两个项
    for i in range(len(clause1.literals)):
        for j in range(len(clause2.literals)):
            if clause1.literals[i].weici == clause2.literals[j].weici and
clause1.literals[i].variable == clause2.literals[j].variable and
clause1.literals[i].fuhao != clause2.literals[j].fuhao:   #如果两个文字的变量相
同，符号相反，那么就可以归结
                delete_flag = 1 #说明可以合并互斥项
                pos1.append(i)
                pos2.append(j)
                break
        if delete_flag == 1:
            break
    #删除元素 合并两个子句

    #合成新子句
    for k in range(len(clause1.literals)):
        if k not in pos1:
            clauses_out.literals.append(clause1.literals[k])
    for k in range(len(clause2.literals)):
        if k not in pos2 and not check_in_clause( clauses_out,
clause2.literals[k]):
            clauses_out.literals.append(clause2.literals[k])
```

```
    if delete_flag == 1:
        clauses_out.parents.append(a)
        clauses_out.parents.append(b)
        return True, clauses_out  #返回子句
    else:
        return False, clauses_out  #没有发生归结
```

这是MGU_Resolve函数：1，判断两个子句是否可以通过MGU算法变化为可以被归结的子句对；2，然后进入到归结步骤：记录互斥的原子公式的位置； 3，建立一个新子句，将第二步中不包含的原子加入到该新子句，作为返回对象。

```
   #子句集调用归结函数
   def unify(self):
        global counter
        i = 0
        while i < len(self.KB):  #不要使用for_in_range结构，因为在循环中会改变KB的
长度
            j = 0
            while j < len(self.KB):
                if(i != j):  #不同的子句才能归结(自己和自己归结没有意义
                    flag, new_clause = MGU_Resolve(self.KB[i],
self.KB[j],i,j,len(self.KB))
                    if len(new_clause.literals)==0 and flag ==True:  #如果子句
集合中要添加一个空子句，说明推理正确
                        display_parents(self, new_clause, len(self.KB))
                        print("命题是正确的！^_^")
                        return
                    if flag == True:  #如果有归结成功的子句，那么就添加到子句集合中
                        for k in range(len(self.KB)): #遍历整个子句集合，不要重复
添加
                            if new_clause.literals == self.KB[k].literals:
                                break
                            if k == len(self.KB)-1:
                                self.KB.append(new_clause)
                j+=1
            i+=1
        print("no more clauses can be resolved")
        return
```

在子句集中调用该函数：1，循环将子句对加入到MGU函数中；2，判断返回的子句是否是空集：说明归结成功；若不是空集，且未在集合中出现，那么就将其加入到子句集中，更新子句集长度。

4. 创新点&优化（如果有）

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）

对两个样例的输出结果：

- Alpine Club
  - 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))


```
[sysu_hpcedu_302@cpn238 ~/scc22/lsr/mp_linpack/resoluation]$ python main.py
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[2,11a](w=mike) = ¬C(mike),S(mike)
R[2,6a](x=mike) = S(mike),C(mike)
R[5,9a](u=snow) = ¬L(mike,snow)
R[12b,13a] = S(mike)
R[8a,14](z=mike) = ¬S(mike)
R[15,16] = []
```

```
PS C:\Users\rogers\Documents\learning in cs\ai> python -u "c:\Users\rogers\Documents\learning in cs\ai\lab2\code\main.py"
0: Atony
1: Amike
2: Ajohn
3: Ltonyrain
4: Ltonysnow
5: ~Ax Sx Cx
6: ~Cy ~Lyrain
7: Lzsnow ~Sz
8: ~Ltonyu ~Lmikeu
9: Ltonyv Lmikev
10: ~Aw ~Cw Sw
R[4,8](u = snow) = 19: ~Lmikesnow
R[7,19](z = mike) = 35: ~Smike
R[1,5](x = mike) = 13: Smike Cmike
R[1,10](w = mike) = 14: ~Cmike Smike
R[13,14]() = 92: Smike
R[35,92]() = 323: []

命题是正确的！^_^
用时：0.217571s
PS C:\Users\rogers\Documents\learning in cs\ai>
```

实验2 归结 问题2

- Block World
  - On(aa,bb)
  - On(bb,cc)
  - Green(aa)
  - ¬Green(cc)
  - (¬On(x,y), ¬Green(x), Green(y))

```
[sysu_hpcedu_302@cpn238 ~/scc22/lsr/mp_linpack/resoluation]$ python main.py
5
On(aa,bb)
On(bb,cc)
Green(aa)
¬Green(cc)
(¬On(x,y), ¬Green(x), Green(y))
R[4,5c](y=cc) = ¬On(x,cc),¬Green(x)
R[3,5b](x=aa) = ¬On(aa,y),Green(y)
R[2,6a](x=bb) = ¬Green(bb)
R[1,7a](y=bb) = Green(bb)
R[8,9] = []
```

4

```
PS C:\Users\rogers\Documents\learning in cs\ai> python -u "c:\Users\rogers\Documents\learning in cs\ai\lab2\code\main.py"
0: Oaabb
1: Obbcc
2: Gaa
3: ~Gcc
4: ~Oxy ~Gx Gy
R[0,4](x = aa, y = bb) = 5: ~Gaa Gbb
R[2,5]() = 8: Gbb
R[1,4](x = bb, y = cc) = 6: ~Gbb Gcc
R[3,6]() = 10: ~Gbb
R[8,10]() = 73: []

命题是正确的！^_^
用时：0.017703s
PS C:\Users\rogers\Documents\learning in cs\ai>
```

显示的证明步骤是正确的。

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）

评测指标：推理步数和CPU时间