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	信计系统结构方向	专业（方向）	ICS
学号	20337268	姓名	张文沁

一、 实验题目

归结推理的 python 实现

二、 实验内容

1. 算法原理

最一般合一算法和命题逻辑归结算法

- 定理：
  - S |-() 当且仅当 S |= ()， S |= () 当且仅当 S 是不可满足的
  - 通过该定理，我们可得KB |= α 当且仅当 KB ∧¬α 不可满足，于是可以通过反证法证明KB |= α
- 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

- Clausal form (便于计算机处理的形式)
  - 每一个子句对应一个元组，元组每一个元素是一个原子公式/原子公式的否定，元素之间的关系是析取关系，表示只要一个原子成立，该子句成立
    - 如子句¬child∨¬male∨boy对应数据结构(¬child,¬male,boy)，空子句()对应False
  - 元组的集合组成子句集S，子句集中每个句子之间是合取关系，表示每一个子句都应该被满足
  - 由于本次实验重点是归结算法，所以问题输入是已经转换过的clausal form，关于具体转换方式感兴趣的同学可以参考课件
- 单步归结
  - 从两个子句中分别寻找相同的原子，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(¬child,¬female,girl)和(child)合并为(¬female,girl)

2. 关键代码展示（带注释）

思路都大同小异，切割函数谓词和变量常量，寻找可以进行合一的相反函数谓词，比对其常量变量，记录可以进行替换的变量并进行替换，直到找不到可以进行替换的谓词。

一、命题逻辑下的实现：

（1）读文件并处理

```
def read_file(filePath):
    global S
    for line in open(filePath,mode = 'r',encoding='utf-8'):
        line = line.replace(' ','').strip()#防止存在空格
        line = line.split(',')#使用，将语句分隔开
        S.append(line)#存入队列
```

（2）取反函数

```
def opposite(line):#返回取反，用于确定能否进行合一
    if '~' in line:
        return line.replace('~','')
    else:
        return '~' + line
```

（3）合一算法

```
def unify():
    global S
    end = False
    while True:
        if end: break
        father = S.pop()
        for i in father[:]:
            if end: break
            for mother in S[:]:
                if end: break
                j = list(filter(lambda x: x==opposite(i),mother))#判断能否进行合一的简单函数
                if j == []:
                    continue
                else:
                    print('\n亲本子句: ' + ' , '.join(father) + ' 和 ' + ' , '.join(mother))
                    father.remove(i)
                    mother.remove(j[0])
                    if(father == [] and mother == []):
                        print('归结式: NIL')
                        end = True
                    elif father == []:
                        print('归结式: ' + ' , '.join(mother))
                    elif mother == []:
                        print('归结式: ' + ' , '.join(mother))
                    else:
                        print('归结式: ' + ' , '.join(father) + ' , ' + ' , '.join(mother))
```

二、第一种实现：

（1）处理字符串：

```
def readClauseSet(filePath):#这个函数只处理最后一行
    global S #字典，用来记录每一行的谓词和变量
    global cot
    for line in open(filePath,encoding = 'utf-8'):
        line = line.replace(' ', '').strip() #先把空格和不可见字符（如换行）去掉
        #print(line)
        if line[0]=='(': #如果是最后一行
           line=list(line) #把字符串变成了list，好操作
           line[0]='' #去掉头尾两个括号
           line[len(line)-1]=''
           line=''.join(line) #因为是''.join所以相当于加起来
        for i in range(len(line)-2):#因为替换了两个头尾，所以缩短
            if line[i+1]==','and line[i]==')':#这里也是处理最后一行，为了把多个函数分开
                line=list(line)
                line[i+1]=';' #改变符号是为了后面进行split
                line=''.join(line)
                #print(line)
        line = line.split(';') #通过;来分割字符串,每个分到元组元素中,为了和前几行做区分使用了;
        #print(line)
        newele={}
        for i in range(len(line)): #
            str1=line[i] #每次读入的元素
            for j in range(len(str1)-1):
                str2=str1#为了不改变str1的值，做一个保留
                if str1[j]=='(':
                   elename=str2[0:j] #找到了函数名称
                   elemem=str2[j+1:len(str1)-1] #用到了str1,elemem是剩下的字符串
                   elemem=elemem.split(',')#分割开
                   newele[elename]=elemem#填充字典，比如'On': ['aa', 'bb']
                   line[i]=newele
                   break
        newele['posnum']=cot #这个参数对应行数
        #例如第一行执行完之后newele里面是{'On': ['aa', 'bb'], 'posnum': 1}
        cot+=1 #记录行数
        S.append(newele) #可以将S认为是二维字典元组，S最后应该是[{'On': ['aa', 'bb'], 'posnum': 1}, {'On': ['bb', 'cc'], 'posnum': 2}, {'Gree
        #print("\n S here")
        #print(S)
```

（2）主算法：

截取了主要的思路，找到 father，判断是否有谓词可以作为 mother，下面进行一系列的判断替换操作，例如更新 V（替换值表），更新 cot（行数），更新 father,mother 等。

```
def unify():
    global S
    global cot
    end = False
    while True:
        if end: break
        father = S.pop()
        jflag=False #判断函数的可合成性
        for i,val in father.items(): #i是索引，val是里面的值
            if end: break
            for mother in S:
                #print("mother",mother)
                if end: break
                j=[]
                pos2=''
                pos3=''
                for key,value in mother.items(): #一个意思，key是索引，value是里面的值
                    if key==opposite(i):#大前提
                        dif=0#判断不同的谓词有多少个
                        for pos1 in value:#mother里面的变量
                            if pos1 in val:
                                continue
                            else:
                                dif+=1
                                pos2=pos1#记录不同的谓词
                        if dif<=1:#说明可以进行合成
                            for h in range(len(val)):#一种思路是找变量x,y,z，最开始的想法，但是会全盘替换，难以实现，此思路为直接判断不同的谓词有
                                if val[h] in value:#存在一样的就跳过
                                    continue
                                else:
                                    pos3=val[h]#记录father和mother里面不同的那个变量
                                    #print("pos3",pos3)
                            flag=True#说明可以合成
                            #print("V",V)
                            for key1 in V.values():#如果这个函数已经被替换过了，就不可再替换，为了修正82行的错误
                                if pos2 ==key1:
                                    flag=False
                                    break
                            if flag==True:
                                j.append(key) #记录每次取到的可替换函数
                                break
```

三、第二种实现：

（1）基本数据结构：

```
#原子公式
class Formula():
    def __init__(self, flag, predicate, variable):
        self.flag = flag #表示一个原子公式中的是或非，1或0
        self.predicate = predicate #表示原子公式中的谓词名称，字符串
        self.variable = variable #表示原子公式中的变元或常量，列表

    def print(self):
        if self.flag == 0:
            print("!", end="")
        print(self.predicate, "(", sep="", end="")
        for i in range(len(self.variable)):
            if i != 0:
                print(",", end="")
            print(self.variable[i], end="")
        print(")", end="")

    def __eq__(self, other):
        if (self.flag == other.flag) and (self.predicate == other.predicate) and (self.variable == other.variable):
            return True
        else:
            return False
```

```python
#子句
class Clause():
    def __init__(self, formulas, number=-1, left=None, right=None):
        self.formulas = formulas #子句中的所有原子公式，列表
        self.left = left #左子树，构建输出时使用
        self.right = right #右子树
        self.number = number #该子句的序号，如果为前提条件，则序号为在子句集中的位置；如果是后续归结出的子句，则序号标注为-1

    def print(self):
        if len(self.formulas) == 0:
            print("[]")
            return
        if len(self.formulas) == 1:
            self.formulas[0].print()
            print()
        else:
            for i in range(len(self.formulas)):
                if i != 0:
                    print(",", end="")
                self.formulas[i].print()
            print()
```

(2) 处理字符串：

```python
def get_condition(s):
    n = input()
    for i in range(int(n)):
        clause_str = input() #输入一个条件子句
        clause_formulas = [] #用于储存一个子句中的原子公式
        #如果以"("开头说明该子句中有多个原子公式
        if(clause_str[0] == "("):
            clause_str = clause_str[1:-1] #去掉字符串前后的括号
            formulas = clause_str.split(", ") #分割得到原子公式
            for j in range(len(formulas)):
                temp = build_formula(formulas[j]) #构建Formula对象
                #temp.print()
                clause_formulas.append(temp)
        #子句即为原子公式
        else:
            clause_formulas.append(build_formula(clause_str))
        clause = Clause(clause_formulas, len(s))
        s.append(clause)
    return n
```

(3) 主算法：

```python
#将子句集中的子句两两进行比较，并进行归结合一
def matching(s, start_clause):
    n = len(s) #先暂时不考虑在归结过程中产生的新的子句
    for i in range(n):
        clause1 = s[i]
        for j in range(max(i + 1, start_clause), n):
            clause2 = s[j]
            #对两个子句进行归结合一
            end = unify(clause1, clause2, s)
            #如果归结结束
            if end == 1:
                return 1
    return 0
```

```python
#对两个子句进行归结合一
def unify(clause1, clause2, s):
    #为了保存子句的原始信息用于最后的输出，先将子句进行拷贝
    clause1_c = copy_clause(clause1)
    clause2_c = copy_clause(clause2)
    #在两个子句中找是否有可以匹配的原子公式
    for i in range(len(clause1_c.formulas)):
        formula1 = clause1_c.formulas[i]
        for j in range(len(clause2_c.formulas)):
            formula2 = clause2_c.formulas[j]
            #如果是非相反，谓词相同
            if (formula1.flag != formula2.flag) and (formula1.predicate == formula2.predicate):
                for k in range(len(formula1.variable)):
                    #如果第一个是变量，第二个是常量，则对第一个子句进行合一
                    if (len(formula1.variable[k]) == 1) and (len(formula2.variable[k]) != 1):
                        change_variable(clause1_c, formula1.variable[k], formula2.variable[k])
                    #如果第一个是常量，第二个是变量，则对第二个子句进行合一
                    if (len(formula1.variable[k]) != 1) and (len(formula2.variable[k]) == 1):
                        change_variable(clause2_c, formula2.variable[k], formula1.variable[k])
                #如果可以归结，也就是常量都相等
                if (formula1.variable == formula2.variable):
                    #得到新的原子公式集合
                    new_formulas = get_new_formulas(formula1, formula2, clause1_c, clause2_c)
                    #构建新的子句，并记录薪子句是从哪两个原子公式归结来的
                    new_clause = Clause(formulas=new_formulas, left=clause1, right=clause2)
                    s.append(new_clause)
                    #如果新的子句为空集则归结完成
                    if len(new_clause.formulas) == 0:
                        return 1
                    else:
                        return 0
```

```python
#得到合一过程中变量的替换信息
def get_chg_v(clause1, clause2):
    v = []
    for i in range(len(clause1.formulas)):
        formula1 = clause1.formulas[i]
        for j in range(len(clause2.formulas)):
            formula2 = clause2.formulas[j]
            if (formula1.flag != formula2.flag) and (formula1.predicate == formula2.predicate):
                for k in range(len(formula1.variable)):
                    # 如果第一个是变量，第二个是常量
                    if (len(formula1.variable[k]) == 1) and (len(formula2.variable[k]) != 1):
                        v.append(formula1.variable[k])
                        v.append(formula2.variable[k])
                    # 如果第一个是常量，第二个是变量
                    if (len(formula1.variable[k]) != 1) and (len(formula2.variable[k]) == 1):
                        v.append(formula2.variable[k])
                        v.append(formula1.variable[k])
                return v
```

```python
#得到能够抵消的两个原子公式在子句中的位置
def get_f_index(clause1, clause2):
    for i in range(len(clause1.formulas)):
        formula1 = clause1.formulas[i]
        for j in range(len(clause2.formulas)):
            formula2 = clause2.formulas[j]
            if (formula1.predicate == formula2.predicate) and (formula1.flag != formula2.flag):
                #如果子句中只包含一个原子公式，则不需要定位
                if len(clause1.formulas) == 1:
                    f1_inx = -1
                else:
                    f1_inx = i
                if len(clause2.formulas) == 1:
                    f2_index = -1
                else:
                    f2_index = j
                return f1_inx, f2_index
```

（4）输出过程：

```python
#输出过程信息
def print_result(stack):
    index1 = len(stack) - 1
    index2 = len(stack) - 2
    while index2 > 0:
        clause1 = stack[index1]
        clause2 = stack[index2]
        # 得到能够抵消的两个原子公式在子句中的位置
        f1_inx, f2_inx = get_f_index(clause1, clause2)
        #将位置信息转化为字母
        f1_str, f2_str = chg_to_str(f1_inx, f2_inx)
        #得到合一过程中变量替换的信息
        v = get_chg_v(clause1, clause2)
        #得到归结后的子句
        next_clause = get_next(stack, clause1, clause2)
        #按照格式输出
        print_str(clause1.number, f1_str, clause2.number, f2_str, v, next_clause)
        index1 -= 2
        index2 -= 2
```

3. 创新点&优化

i. 两个思路最大的改变在于使用的数据结构不一，前者为队列字典，后者用到类和堆栈

ii. 注释清晰

iii. 第一种算法思路简单但是效率低且容易出错，第二种算法整体而言板块更加清晰，分得更细致。

三、 实验结果及分析

1. 实验结果展示示例

一、命题逻辑:

测试样例如下:

```
1   p
2   ~p , ~q , r
3   ~u , q
4   ~t , q
5   t
6   ~r
```

结果如下:

```
----
--------命题逻辑归结推理系统--------
----

亲本子句：~r 和 ~p , ~q , r
归结式：~p , ~q

亲本子句：t 和 ~t , q
归结式：q

亲本子句：q 和 ~p , ~q
归结式：~p

亲本子句：~p 和 p
归结式：NIL
0.005001068115234375
PS F:\CodeFile for Python>
```

二、第一种实现:

（1）样例一

```
----
--------命题逻辑归结推理系统--------
----
4
GradStudent(sue)
(!GradStudent(x),Student(x))
(!Student(x),HardWorker(x))
!HardWorker(sue)
R[4,3b](sue = x) = !Student (x)
R[5,2b]= !GradStudent (x)
R[1,6](sue = x) = []
0.010001897811889648
PS F:\CodeFile for Python>
```

（2）样例二

```
----
--------命题逻辑归结推理系统--------
----
5
On(aa,bb)
On(bb,cc)
Green(aa)
!Green(cc)
(!On(x,y), !Green(x), Green(y))
R[5b,3](x = aa) = !On (aa,y),Green (y)
R[5b,6b](x = y) = !On (y,y),Green (y)和 !On (aa,y)
R[5c,4](y = cc) = !On (x,cc),!Green (x)
R[9a,2](x = bb) = !Green (bb)
R[8,1](y = bb) = []
0.01799154281616211
PS F:\CodeFile for Python>
```

（3）样例三

三、第二种实现：

（1）样例一

```
----
--------命题逻辑归结推理系统--------
----
4
GradStudent(sue)
(!GradStudent(x), Student(x))
(!Student(x), HardWorker(x))
!HardWorker(sue)
R[4,3b](x=sue) = !Student(sue)
R[2a,1](x=sue) = Student(sue)
R[5,6] = []
0.003000497817993164
PS F:\CodeFile for Python>
```

```
----
--------命题逻辑归结推理系统--------
----
4
GradStudent(sue)
(!GradStudent(x), Student(x))
(!Student(x), HardWorker(x))
!HardWorker(sue)
R[4,3b](x=sue) = !Student(sue)
R[2a,1](x=sue) = Student(sue)
R[5,6] = []
0.004000663757324219
PS F:\CodeFile for Python>
```

（2）样例二

```
----
--------命题逻辑归结推理系统--------
----
5
On(aa,bb)
On(bb,cc)
Green(aa)
!Green(cc)
(!On(x,y), !Green(x), Green(y))
R[5c,4](y=cc) = !On(x,cc),!Green(x)
R[5b,3](x=aa) = !On(aa,y),Green(y)
R[6a,2](x=bb) = !Green(bb)
R[7a,1](y=bb) = Green(bb)
R[8,9] = []
0.013004779815673828
PS F:\CodeFile for Python>
```

（3）样例三

```
----
--------命题逻辑归结推理系统--------
----
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[11a,2](w=mike) = !C(mike),S(mike)
R[6a,2](x=mike) = S(mike),C(mike)
R[9a,5](u=snow) = !L(mike,snow)
R[12a,13b] = S(mike)
R[14,8a](z=mike) = !S(mike)
R[15,16] = []
0.062014102935791016
PS F:\CodeFile for Python>
```

2. 评测指标展示及分析

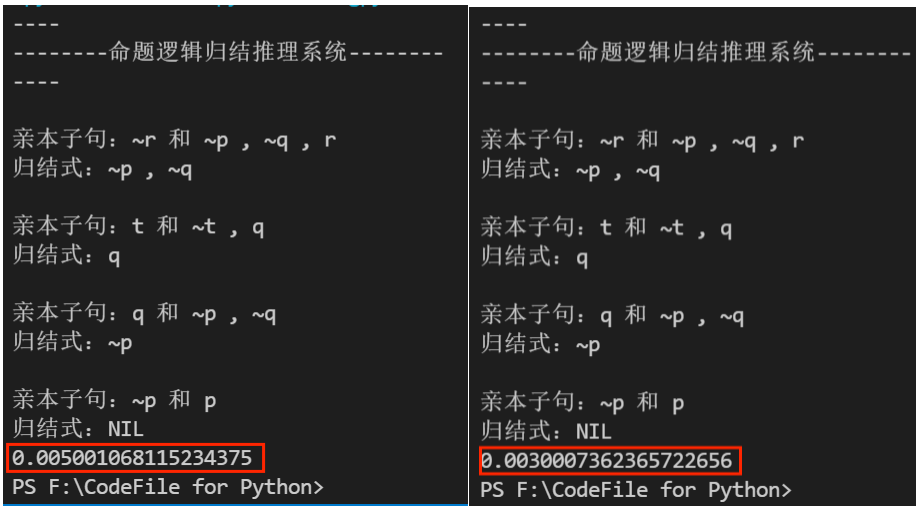
分析运行时间的程序为：

```python
import time
start = time.time()
#executing
end = time.time()
print(end-start)
```

因为有两种实现方式，故下面对这两种实现进行测评（都是在同一个后台情况下进行的测评）每个样例执行五次取其平均值：

一、命题逻辑：

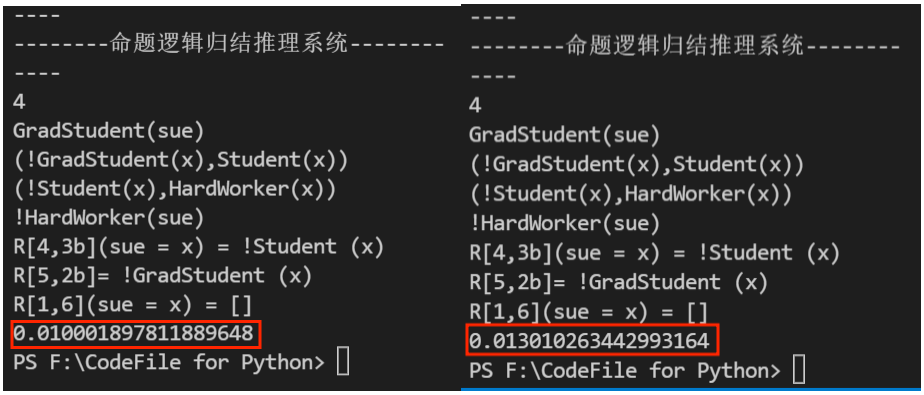
最后一行为运行时间。分析，在六行的测试样例（见结果展示栏）下，多次运行，每次时间不一样，但谓词逻辑的运行时间都为10^{-3}数量级，此数字在输入样例不同的情况下和下两者并无可比之处，只是做为参考。



二、第一种实现：

分析：第一种实现方式数量级为10^{-2}，但是遇到样例三会卡测评，应该是逻辑有错误，但是暂时没有找到好的解决方法。

（1）样例一：



（2）样例二：

```
----
--------命题逻辑归结推理系统--------
----
5
On(aa,bb)
On(bb,cc)
Green(aa)
!Green(cc)
(!On(x,y), !Green(x), Green(y))
R[5b,3](x = aa) = !On (aa,y),Green (y)
R[5b,6b](x = y) = !On (y,y),Green (y)和 !On (aa,y)
R[5c,4](y = cc) = !On (x,cc),!Green (x)
R[9a,2](x = bb) = !Green (bb)
R[8,1](y = bb) = []
0.01799154281616211
PS F:\CodeFile for Python>
```

```
----
--------命题逻辑归结推理系统--------
----
5
On(aa,bb)
On(bb,cc)
Green(aa)
!Green(cc)
On(bb,cc)
Green(aa)
!Green(cc)
(!On(x,y), !Green(x), Green(y))
R[5b,3](x = aa) = !On (aa,y),Green (y)
R[5b,6b](x = y) = !On (y,y),Green (y)和 !On (aa,y)
R[5c,4](y = cc) = !On (x,cc),!Green (x)
R[9a,2](x = bb) = !Green (bb)
R[8,1](y = bb) = []
0.015002965927124023
PS F:\CodeFile for Python>
```

（3）样例三：

三、第二种实现：

分析：第二种实现方式在第一个实例下的数量级为10^{-3}，另外两个样例数量级为10^{-2}，但是数字大于第一种实现方式。

（1）样例一：

```
----
--------命题逻辑归结推理系统--------
----
4
GradStudent(sue)
(!GradStudent(x), Student(x))
(!Student(x), HardWorker(x))
!HardWorker(sue)
R[4,3b](x=sue) = !Student(sue)
R[2a,1](x=sue) = Student(sue)
R[5,6] = []
0.003000497817993164
PS F:\CodeFile for Python>
```

```
----
--------命题逻辑归结推理系统--------
----
4
GradStudent(sue)
(!GradStudent(x), Student(x))
(!Student(x), HardWorker(x))
!HardWorker(sue)
R[4,3b](x=sue) = !Student(sue)
R[2a,1](x=sue) = Student(sue)
R[5,6] = []
0.004000663757324219
PS F:\CodeFile for Python>  F:; cd 'F:\
```

（2）样例二：

```
----
--------命题逻辑归结推理系统--------
----
5
On(aa,bb)
On(bb,cc)
Green(aa)
!Green(cc)
(!On(x,y), !Green(x), Green(y))
R[5c,4](y=cc) = !On(x,cc),!Green(x)
R[5b,3](x=aa) = !On(aa,y),Green(y)
R[6a,2](x=bb) = !Green(bb)
R[7a,1](y=bb) = Green(bb)
R[8,9] = []
0.013004779815673828
PS F:\CodeFile for Python>
```

```
----
--------命题逻辑归结推理系统--------
----
5
On(aa,bb)
On(bb,cc)
Green(aa)
!Green(cc)
(!On(x,y), !Green(x), Green(y))
R[5c,4](y=cc) = !On(x,cc),!Green(x)
R[5b,3](x=aa) = !On(aa,y),Green(y)
R[6a,2](x=bb) = !Green(bb)
R[7a,1](y=bb) = Green(bb)
R[8,9] = []
0.012002706527709961
PS F:\CodeFile for Python>  F:; cd 'F:\
```

（3）样例三：

```
----
--------命题逻辑归结推理系统--------
----
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[11a,2](w=mike) = !C(mike),S(mike)
R[6a,2](x=mike) = S(mike),C(mike)
R[9a,5](u=snow) = !L(mike,snow)
R[12a,13b] = S(mike)
R[14,8a](z=mike) = !S(mike)
R[15,16] = []
0.062014102935791016
PS F:\CodeFile for Python>
```

```
----
--------命题逻辑归结推理系统--------
----
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[11a,2](w=mike) = !C(mike),S(mike)
R[6a,2](x=mike) = S(mike),C(mike)
R[9a,5](u=snow) = !L(mike,snow)
R[12a,13b] = S(mike)
R[14,8a](z=mike) = !S(mike)
R[15,16] = []
0.05201125144958496
PS F:\CodeFile for Python>
```

最终大致可以得出结论：第一种实现方式简单稳定但是有缺陷，第二种实现方式在语句少的情况下性能更优。

四、 参考资料

https://zhangguohao.blog.csdn.net/article/details/105471307