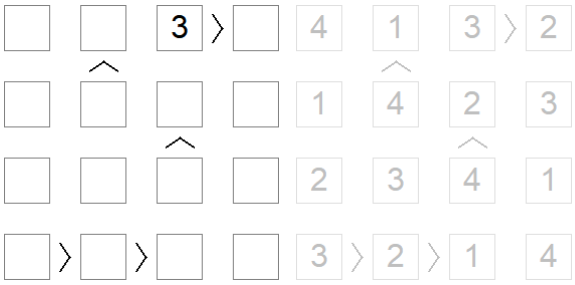
P02 CSP and KRR

学号	姓名	专业(方向)
18340215	张天祎	计科（大数据与人工智能）

1.Futoshiki (GAC, C++/Python)

1.1 Description

Futoshiki is a board-based puzzle game, also known under the name Unequal. It is playable on a square board having a given fixed size (4 × 4 for example), please see Figure 1. The purpose of the game is to discover the digits hidden inside the board's cells; each cell is filled with a digit between 1 and the board's size. On each row and column each digit appears exactly once; therefore, when revealed, the digits of the board form a so-called Latin square. At the beginning of the game some digits might be revealed. The board might also contain some inequalities between the board cells; these inequalities must be respected and can be used as clues in order to discover the remaining hidden digits. Each puzzle is guaranteed to have a solution and only one. You can play this game online: http://www.futoshiki.org/.



1.2 Tasks

1.Describe with sentences the main ideas of the GAC algorithm and the main differences between the GAC and the forward checking (FC) algorithm.(10 points)

- 在每次变量实例化后，先将所有和该变量相关的约束加入队列，对每个约束中的所有非实例化变量都检测它的值域中的各实例是不是GAC（其它变量能保证该实例下至少有一个解）的，若不是，则剔除，每剔除一个非GAC值，需要将它的相关约束重新加入队列。
- 两算法的不同主要在变量实例化后的Check的过程。GAC算法根据所有约束进行Check，确保搜索空间中的每一个状态都是GAC的，这需要每个约束的每个未实例化变量的域的每个值都是GAC的；FC算法根据仅有一个未实例化变量的约束进行Check，检测该未实例化变量域内的每个值，筛去不合适的值。GAC的Check过程更复杂，开销更大，也更强大；但考虑到这样的约束传播的开销，在一些问题中，FC算法也不失为一种优秀的算法。

2.The GAC Enforce procedure from class acts as follows: when removing d from CurDom[V], push all constraints C' such that V ∈ scope(C') and C' ∉ GACQueue onto GACQueue. What's the reason behind this operation? Can it be improved and how?(20 points)

- 在有关V的约束中的其它变量的域中的一些值可能是被d所保证有解的，当d被从V的值域中移除时，就会触发连锁反应，那些值也可能变成无解的。所以需要把V相关的所有约束都重新加入队列进行重新检测（已经进入队列的约束当然是不用加入的）。
- 在GAC_Enforce的过程中可以把d保证的约束记录下来，当d被移除时，不用添加V的所有约束，只需要添加d所保证的约束即可。

3.Use the GAC algorithm to implement a Futoshiki solver by C++ or Python. (20 points)

```
class Generalized_Arc_Consistency:
    def __init__(self, maps, less_constraints):
        self.size = len(maps[0])
        self.maps = maps
        self.less_constraints = []
        self.cons_index = {}
        self.que = []
        self.nodes = 0
        self.inference = 0
        for i in range(self.size):
            for j in range(self.size):
                self.cons_index[(i, j)] = []
        # 添加约束表和约束索引
        for i in less_constraints:
            self.less_constraints.append((0, (i[0], i[1]), (i[2], i[3]))) # 不等
式约束为三元组(0, (i1, j1), (i2, j2))
            self.cons_index[(i[0], i[1])].append(((i[0], i[1]), (i[2], i[3]))) #
索引下标为位置元组(i, j)， 返回值为一列表
            self.cons_index[(i[2], i[3])].append(((i[0], i[1]), (i[2], i[3])))
        self.domain = []
        self.size = len(maps[0])
        # 根据maps初始化域
        for i in range(self.size):
            tem = []
            for j in range(self.size):
                if self.maps[i][j]:
                    tem.append([self.maps[i][j]]) # 被赋值后域内仅一个值
                else:
                    tem.append(list(range(1, self.size + 1)))  # range是不可变
list，不能使用remove
            self.domain.append(tem)
        # 初始化进行一次全面的GAC_Enforce
        self.add_all_cons()
        self.GAC_Enforce()

    def add_all_cons(self):
        for i in range(self.size):
            for j in range(self.size):
                self.add_cons_pos(i, j)

    def add_cons_pos(self, i, j):
        for c in self.cons_index[(i, j)]:  # 添加不等式约束
            self.add_cons((0, c[0], c[1]))
        self.add_cons((i + 1, -(j + 1))) # 添加不等约束
        self.add_cons((-(i + 1), j + 1))

    def add_cons(self, cons):
        if cons in self.que:
            return
        else:
            self.que.append(cons)

    def check_empty(self, i ,j):
        if self.domain[i][j] == []:
            self.que = []
```

```
            return True
        return False

    def check_end(self):
        count = 0
        hold = []
        for i in range(self.size):
            for j in range(self.size):
                if not self.maps[i][j]:
                    count += 1
                    hold.append((i,j))
                    if count > 1:
                        return False
        i,j = hold[0][0],hold[0][1]
        self.maps[i][j] = self.domain[i][j][0]
        return True

    def min_domain_branch(self):
        # 选取最小域的变量拓展
        size = 1
        while(1):
            for i in range(self.size):
                for j in range(self.size):
                    if not self.maps[i][j]:
                        if size == len(self.domain[i][j]):
                            return i, j
            size += 1

    def GAC_Enforce(self):
        begin = timer()
        del_list = []
        empty = False
        while(self.que):
            curr = self.que[0]
            if curr[0] == 0: # 不等式约束
                v1 = curr[1]
                v2 = curr[2]
                if not empty and not self.maps[v1[0]][v1[1]]: # 对第一个变量，未赋值
才考虑
                    tem = copy.deepcopy(self.domain[v1[0]][v1[1]])
                    for i in tem:
                        satisfy = False
                        for j in self.domain[v2[0]][v2[1]]:
                            if i < j:
                                satisfy = True
                                break
                        if not satisfy:
                            self.domain[v1[0]][v1[1]].remove(i)
                            # print((v1[0], v1[1], i))
                            del_list.append((v1[0], v1[1], i))
                            if self.check_empty(v1[0], v1[1]):
                                empty = True
                                break
                            self.add_cons_pos(v1[0], v1[1])
                if not empty and not self.maps[v2[0]][v2[1]]:  # 对第二个变量，未赋
值才考虑
                    tem = copy.deepcopy(self.domain[v2[0]][v2[1]])
                    for j in tem:  # 对第二个变量
```

```
                    satisfy = False
                    for i in self.domain[v1[0]][v1[1]]:
                        if i < j:
                            satisfy = True
                            break
                    if not satisfy:
                        self.domain[v2[0]][v2[1]].remove(j)
                        # print((v2[0],v2[1],j))
                        del_list.append((v2[0], v2[1], j))
                        if self.check_empty(v2[0], v2[1]):
                            empty = True
                            break
                        self.add_cons_pos(v2[0], v2[1])
            else:
                if curr[0] < 0: #行约束
                    i = -curr[0] - 1
                    j = curr[1] - 1
                    if not self.maps[i][j]: # 每个未赋值变量
                        tem = copy.deepcopy(self.domain[i][j])
                        for p in tem: # 每个可能值
                            if not empty:
                                for k in range(self.size):  # 行内每一个位置
                                    if k != j:  # 自己不和自己比
                                        flag = False
                                        for m in self.domain[i][k]:
                                            if p != m:  # 找到一个可能值
                                                flag = True
                                                break
                                        if not flag:
                                            self.domain[i][j].remove(p)
                                            del_list.append((i, j, p))
                                            if self.check_empty(i, j):
                                                empty = True
                                                break
                                            self.add_cons_pos(i, j)
                                            break
                        for k in range(self.size): # 检查行内其它变量
                            if not empty and k != j and not self.maps[i][k]:
                                tem = copy.deepcopy(self.domain[i][k])
                                for p in tem:
                                    flag = False
                                    for m in self.domain[i][j]:
                                        if p != m:
                                            flag = True
                                            break
                                    if not flag:
                                        self.domain[i][k].remove(p)
                                        del_list.append((i, k, p))
                                        if self.check_empty(i, k):
                                            empty = True
                                            break
                                        self.add_cons_pos(i, k)
                                        break
                else: # 列约束
                    i = curr[0] - 1
                    j = -curr[1] - 1
                    if not self.maps[i][j]:  # 每个未赋值变量
                        tem = copy.deepcopy(self.domain[i][j])
```

```
                        for p in tem:  # 每个可能值
                            if not empty:
                                for k in range(self.size):  # 行内每一个位置
                                    if k != i:  # 自己不和自己比
                                        flag = False
                                        for m in self.domain[k][j]:
                                            if p != m:  # 找到一个可能值
                                                flag = True
                                                break
                                        if not flag:
                                            self.domain[i][j].remove(p)
                                            del_list.append((i, j, p))
                                            if self.check_empty(i, j):
                                                empty = True
                                                break
                                            self.add_cons_pos(i, j)
                                            break
                    for k in range(self.size): # 检查列内其它变量
                        if not empty and k != i and not self.maps[k][j]:
                            tem = copy.deepcopy(self.domain[k][j])
                            for p in tem:
                                flag = False
                                for m in self.domain[i][j]:
                                    if p != m:
                                        flag = True
                                        break
                                if not flag:
                                    self.domain[k][j].remove(p)
                                    del_list.append((k, j, p))
                                    if self.check_empty(k, j):
                                        empty = True
                                        break
                                    self.add_cons_pos(k, j)
                                    break
            if not empty:
                self.que.remove(curr)
        end = timer()
        self.inference += end - begin
        return [del_list, not empty]

    def GAC(self, i, j):
        self.nodes += 1
        if self.check_end():
            return True
        # original version
        # if i==self.size-1 and j ==self.size-1: # 终止
        #     if not self.maps[i][j]: # 无值时设定值
        #         self.maps[i][j] = self.domain[i][j][0]
        #     return True
        # ni = i + (j + 1) // self.size
        # nj = (j + 1) % self.size
        ni,nj = self.min_domain_branch()

        if self.maps[i][j]: # 已有值
            return self.GAC(ni, nj)
        tem = copy.deepcopy(self.domain[i][j])
        for val in tem:
```

```
            rst = self.GAC_Set(i, j, val) #rst第一个元素是被删除的域值（三元组列表，位置+值），第二个元素是T/F
            if rst[1]:
                if self.GAC(ni, nj):
                    return True
            self.GAC_Reset(i, j, rst[0])
        return False

    def GAC_Set(self, i, j, val):
        self.maps[i][j] = val
        del_list = []
        tem = copy.deepcopy(self.domain[i][j])
        for m in tem:
            if m != val:
                self.domain[i][j].remove(m)
                del_list.append((i,j,m))

        self.add_cons_pos(i, j)
        rst = self.GAC_Enforce()
        return [del_list + rst[0],rst[1]]

    def GAC_Reset(self, i, j, tem):
        self.maps[i][j] = 0
        for k in tem:
            self.domain[k[0]][k[1]].append(k[2])
```

4.Explain any ideas you use to speed up the implementation. (10 points) (10 points)

- 初始化GAC问题时，将所有约束加入队列，进行一次全面的GAC_Enforce。
- 延伸结点时优先延伸域最小的节点。

5. Run the following 5 test cases to verify your solver's correctness. We also provide test file "datai.txt" for every test case i. Refer to the "readme.txt" for more details. (20 points)

结果如下：

```
3 2 4 5 1
4 1 2 3 5
2 4 5 1 3
1 5 3 4 2
5 3 1 2 4

GAC Total Time: 0.0059746999999999995 s
Number of Nodes Searched: 44
Average Inference Time Per Node:  0.12173636363636389 ms


3 2 4 5 1
4 1 2 3 5
2 4 5 1 3
1 5 3 4 2
5 3 1 2 4

FC Total Time: 0.003628199999999998 s
Number of Nodes Searched: 141
Average Inference Time Per Node:  0.016079432624116208 ms
```

```
4 1 3 5 2 6
1 5 2 6 4 3
3 4 1 2 6 5
5 6 4 3 1 2
2 3 6 4 5 1
6 2 5 1 3 4

GAC Total Time: 0.007084200000000013 s
Number of Nodes Searched: 59
Average Inference Time Per Node:  0.1056457627118638 ms


4 1 3 5 2 6
1 5 2 6 4 3
3 4 1 2 6 5
5 6 4 3 1 2
2 3 6 4 5 1
6 2 5 1 3 4

FC Total Time: 0.006631899999999996 s
Number of Nodes Searched: 332
Average Inference Time Per Node:  0.013417469879518333 ms
```

```
2 4 3 5 1 7 6
4 1 6 2 7 5 3
1 2 5 7 3 6 4
5 6 7 1 4 3 2
7 3 2 4 6 1 5
3 5 1 6 2 4 7
6 7 4 3 5 2 1

GAC Total Time: 0.032617799999999975 s
Number of Nodes Searched: 184
Average Inference Time Per Node:  0.15812010869564982 ms


2 4 3 5 1 7 6
4 1 6 2 7 5 3
1 2 5 7 3 6 4
5 6 7 1 4 3 2
7 3 2 4 6 1 5
3 5 1 6 2 4 7
6 7 4 3 5 2 1

FC Total Time: 3.2862274 s
Number of Nodes Searched: 86114
Average Inference Time Per Node:  0.027972809299301227 ms
```

```
8 7 6 5 1 2 3 4
4 8 2 3 6 5 7 1
7 2 3 4 8 1 6 5
6 3 1 2 4 7 5 8
5 4 8 1 7 3 2 6
2 6 5 8 3 4 1 7
1 5 7 6 2 8 4 3
3 1 4 7 5 6 8 2

GAC Total Time: 0.23882519999999996 s
Number of Nodes Searched: 418
Average Inference Time Per Node:  0.551611483253595 ms


8 7 6 5 1 2 3 4
4 8 2 3 6 5 7 1
7 2 3 4 8 1 6 5
6 3 1 2 4 7 5 8
5 4 8 1 7 3 2 6
2 6 5 8 3 4 1 7
1 5 7 6 2 8 4 3
3 1 4 7 5 6 8 2

FC Total Time: 4.583847500000001 s
Number of Nodes Searched: 122388
Average Inference Time Per Node:  0.028447809425759113 ms
```

```
9 8 6 1 3 7 5 4 2
5 3 8 6 2 4 9 1 7
8 6 4 2 1 5 3 7 9
1 7 5 9 6 8 4 2 3
4 2 3 5 7 9 8 6 1
7 9 2 8 4 6 1 3 5
2 1 9 4 8 3 7 5 6
3 5 1 7 9 2 6 8 4
6 4 7 3 5 1 2 9 8

GAC Total Time: 0.5420670000000012 s
Number of Nodes Searched: 1293
Average Inference Time Per Node:  0.3964863882443885 ms


9 8 6 1 3 7 5 4 2
5 3 8 6 2 4 9 1 7
8 6 4 2 1 5 3 7 9
1 7 5 9 6 8 4 2 3
4 2 3 5 7 9 8 6 1
7 9 2 8 4 6 1 3 5
2 1 9 4 8 3 7 5 6
3 5 1 7 9 2 6 8 4
6 4 7 3 5 1 2 9 8

FC Total Time: 6.9635754 s
Number of Nodes Searched: 142989
Average Inference Time Per Node:  0.03897753393617418 ms
```

6. Run the FC algorithm you implemented in E04 and the GAC algorithm you implemented in Task 3 on the 5 test cases, and fill in the following table. In the table, "Total Time" means the total time the algorithm uses to solve the test case, "Number of Nodes Searched" means the total number of nodes traversed by the algorithm, and "Average Inference Time Per Node" means the average time for constraint propagation (inference) used in each node (note that this time is not equal to the total time divided by the number of nodes searched). Analyse the reasons behind the experimental results, and write them in your report. (20 points)

TestCase Number	Method	Time Used	Nodes Searched	Time Per Node
1	FC GAC	0.004s 0.006s	141 44	0.016ms 0.122ms
2	FC GAC	0.006s 0.007s	332 59	0.013ms 0.106ms
3	FC GAC	3.286s 0.032s	86114 184	0.028ms 0.158ms
4	FC GAC	4.584s 0.239s	122388 418	0.028ms 0.552ms
5	FC GAC	6.964s 0.542s	142989 1293	0.039ms 0.396ms

总体上看，GAC的节点平均推理时间都在FC节点平均推理时间的10倍左右，而两者搜索节点数之比则随问题规模大致呈$O(n^3)$增长。例子1和2中，FC的表现甚至比GAC要好，可见在小规模问题中，拓展结点数尚不多，FC由于推理开销少，是优于GAC的；但在大规模问题例子3、4和5中，拓展结点数对算法效率的影响占据了上风，此时由于GAC剪枝更成功，是一棵比较瘦的搜索树，虽然推理开销还是较大，但拓展结点少得多，所以更快。

2. Resolution

1. Implement the MGU algorithm. (10 points)

见下方代码MGU()函数及相关注释。

2. Using the MGU algorithm, implement a system to decide via resolution if a set of first-order clauses is satisfiable. The input of your system is a file containing a set of first-order clauses. In case of unsatisfiability, the output of your system is a derivation of the empty clause where each line is in the form of "R[8a,12c]clause". Only include those clauses that are useful in the derivation. (10 points)

```
#  -*-  coding:utf-8  -*-
import sys
import re
import copy
import numpy as np

# 助教的代码，把¬改为了!
def deal(kb):
    clauses = []
    id = 1
    for i in kb:
        clause = []
        for item in re.findall(r'!*[a-zA-Z]+\([a-zA-Z,\s]*\)', i):
```

```
            items = re.findall(r'[!a-zA-Z]+', item)
            clause.append(items)
        clauses.append(clause)
        id += 1
    return clauses

def load(i):
    file = open("kb" + str(i) + ".txt", 'r')
    kb = []
    for i in file:
        kb.append(i)
    file.close()
    clauses = deal(kb)
    return clauses

def show(clauses):
    id = 1
    for i in clauses:
        print(id,":", i)
        id += 1

def val(str):
    return len(str)==1

def clause_val(clause):
    length = len(clause)
    for i in range(1, length):
        if val(clause[i]):
            return True
    return False

def solve_end(clause1, clause2):
    # 两个单谓词项才有很可能结束
    c1 = clause1[0]
    c2 = clause2[0]
    if len(clause1)==1 and len(clause2)==1 and not clause_val(c1) and not
clause_val(c2) and (('!' + c1[0]) == c2[0] or c1[0] == ('!' + c2[0])):
        for i in range(1, len(c1)):
            if c1[i] != c2[i]:
                return False
        return True
    return False

def MGU(clause_a, clause_b):
    # 不改原 kb
    clause1 = copy.deepcopy(clause_a)
    clause2 = copy.deepcopy(clause_b)
    sub_sigma = []
    new_clause = []
    len_1 = len(clause1)
    len_2 = len(clause2)
    pop_list = [] # 存应该被删去的谓词
    # 遍历两子句每一个谓词
    for i in range(len_1):
        for j in range(len_2):
            # i 不可能被反复合一，但j有可能会，需要考虑
            if clause2[j] in [k[1] for k in pop_list]:
                continue
```

```
            pre_1 = clause1[i][0]
            pre_2 = clause2[j][0]
            # 纯粹的MGU算法此处判断应该为
            # if pre_1 == pre_2:
            # 在归结时只有谓词差一个‘!’才考虑合一
            if (('!'+pre_1)==pre_2 or pre_1==('!'+pre_2)):
                # 若两谓词有不相等的对应项，不能合一
                flag = True
                for pos in range(1, len(clause1[i])):
                    str1 = clause1[i][pos]
                    str2 = clause2[j][pos]
                    if not val(str1) and not val(str2) and str1 != str2:
                        flag = False
                        break
                if not flag:
                    continue
                # 开始合一
                for pos in range(1, len(clause1[i])):
                    str1 = clause1[i][pos]
                    str2 = clause2[j][pos]
                    if str1 != str2: # 相等则不处理
                        if val(str1): # 子句1中的文字为变量，不论子句2中是什么都直接赋值
                            # 三元组，0表示子句0，1表示子句1；谓词位置；变化关系
                            sub_sigma.append([i, j, str1 + '=' + str2])
                            # 赋值
                            for m in range(len_1):
                                for n in range(1, len(clause1[m])):
                                    if clause1[m][n] == str1:
                                        clause1[m][n] = str2
                        elif val(str2):
                            sub_sigma.append([i, j, str2 + '=' + str1])
                            for m in range(len_2):
                                for n in range(1, len(clause2[m])):
                                    if clause2[m][n] == str2:
                                        clause2[m][n] = str1
                # 这两个子句已经互反了，把他们放到删除列表中
                pop_list.append((clause1[i], clause2[j]))
                break
    if sub_sigma == []: # 完全不能合一
        new_clause = []
    else:
        for i in pop_list:
            clause1.remove(i[0])
            clause2.remove(i[1])
        new_clause = clause1
        # 此处有bug，子句有可能一样，要一个一个添加
        for m in clause2:
            if m not in new_clause:
                new_clause.append(m)
    return new_clause, sub_sigma

def show_pros(ori_len, out, clauses):
    id = ori_len + 1
    map = {}
    for i in range(ori_len+1):
        map[i] = i+1
    for i in range(len(out)):
        s3 = str(out[i][5])
```

```
        s4 = str(clauses[out[i][0]])
        print()
        # ori_id output version
        # s1 = str(out[i][1]+1) + str("" if len(clauses[out[i][1]]) == 1 else
chr(ord('a') + out[i][3]))
        # s2 = str(out[i][2]+1) + str("" if len(clauses[out[i][2]]) == 1 else
chr(ord('a') + out[i][4]))
        # print("%d. R[%s,%s]%s   %s" % (out[i][0] + 1, s1, s2, s3, s4))
        s5 = str(map[out[i][1]]) + str("" if len(clauses[out[i][1]]) == 1 else
chr(ord('a') + out[i][3]))
        s6 = str(map[out[i][2]]) + str("" if len(clauses[out[i][2]]) == 1 else
chr(ord('a') + out[i][4]))
        print("%d. R[%s,%s]%s   %s" % (id, s5, s6, s3, s4))
        map[out[i][0]] = id
        id += 1

def pick_min(j_list, clauses):
    length = len(j_list)
    size = 1
    while(1):
        pos = 0
        while (pos < length):
            j = j_list[pos]
            if len(clauses[j]) == size:
                j_list.pop(pos)
                return j
            pos += 1
        size += 1

def test(num):
    clauses = load(num)
    show(clauses)
    rst = []
    out = []
    que = []
    for i in range(len(clauses)):
        rst.append((i,-1,-1,-1,-1,[])) # 规则索引,i,j,a,b,变换
    ori_len = len(clauses)
    id = len(clauses)
    flag = False
    while(not flag):
        length = len(clauses)
        hold = []
        for i in range(length):
            if not flag:
                j_list = list(range(i+1, length))
                while(j_list):
                    j = pick_min(j_list, clauses)
                    new_clause, curr_sigma = MGU(clauses[i], clauses[j])
                    # 结束
                    if solve_end(clauses[i], clauses[j]):
                        hold.append([])
                        rst.append((id,i,j,0,0,[]))
                        que.append((id,i,j,0,0,[]))
                        id += 1
                        flag = True
                        break
                    # 有效合一
```

```
                    if new_clause != [] and new_clause not in clauses and
new_clause not in hold:
                        hold.append(new_clause)
                        # 添加推理规则链
                        rst.append((id,i,j,curr_sigma[0][0],curr_sigma[0][1],
[k[2] for k in curr_sigma]))
                        id += 1
        if hold == []:
            break
        clauses += hold
    # 层序遍历,找回推理依据
    while(que):
        hold = []
        while(que):
            curr = que.pop(0)
            if curr[1] != -1:
                out.append(curr)
            else:
                continue
            hold.append(rst[curr[2]])
            hold.append(rst[curr[1]])
        que = hold
    out = out[::-1]
    if flag:
        show_pros(ori_len, out, clauses)
    else:
        print("KB is Satisfiable.")

def main():
    DAT_NUM = 3
    if len(sys.argv) == 1:
        for i in range(DAT_NUM):
            print("TEST: ",i)
            test(i)
            print()
            print()
    else:
        test(int(sys.argv[1]))

if __name__ == "__main__":
    main()
```

3. Explain any ideas you use to improve the search efficiency. (5 points)

- 使用了单元优先的策略，优先使用单文字进行归结。在代码实现(pick_min()函数)中实际做的更多，在确定合一的一个子句后，另一个子句每次都选取未合一过的文字数最小的子句。

4. Run your system on the examples of hardworker(sue), 3-blocks, Alpine Club. Include your input and output files in your report. (15 points)

结果如下：

```
C:\Users\E480\Desktop\AI\pro_2\MY>python KRR.py
TEST:  0
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[6c,11b]['x=w']   [['!A', 'w'], ['S', 'w']]

13. R[5,9a]['u=snow']   [['!L', 'mike', 'snow']]

14. R[2,12a]['w=mike']   [['S', 'mike']]

15. R[8a,13]['z=mike']   [['!S', 'mike']]

16. R[14,15][]   []
```

```
TEST:  1
1 : [['GradStudent', 'sue']]
2 : [['!GradStudent', 'x'], ['Student', 'x']]
3 : [['!Student', 'x'], ['HardWorker', 'x']]
4 : [['!HardWorker', 'sue']]

5. R[1,2a]['x=sue']   [['Student', 'sue']]

6. R[3b,4]['x=sue']   [['!Student', 'sue']]

7. R[5,6][]   []


TEST:  2
1 : [['On', 'aa', 'bb']]
2 : [['On', 'bb', 'cc']]
3 : [['Green', 'aa']]
4 : [['!Green', 'cc']]
5 : [['!On', 'x', 'y'], ['!Green', 'x'], ['Green', 'y']]

6. R[3,5b]['x=aa']   [['!On', 'aa', 'y'], ['Green', 'y']]

7. R[4,5c]['y=cc']   [['!On', 'x', 'cc'], ['!Green', 'x']]

8. R[1,6a]['y=bb']   [['Green', 'bb']]

9. R[2,7a]['x=bb']   [['!Green', 'bb']]

10. R[8,9][]   []
```

5. What do you think are the main problems for using resolution to check for satisfiability for a set of first-order clauses? Explain. (10 points)

- 最大的困难是不知道从KB中先选哪两个子句进行合一
- 使用单元优先策略是一个可能的解决策略，但实际上它也不是完备的
- 个人感觉根本上是缺少一个启发式的策略或者说搜索时的启发式函数，来指导从KB中挑选两个子句进行合一的过程