Pair Programming Equitable Participation & Honesty Affidavit

We the undersigned promise that we have in good faith attempted to follow the principles of pair programming. Although we were free to discuss ideas with others, the implementation is our own. We have shared a common workspace and taken turns at the keyboard for the majority of the work that we are submitting. Furthermore, any non programming portions of the assignment were done independently. We recognize that should this not be the case, we will be subject to penalties as outlined in the course syllabus.

Jinqi Cheng 4/14/2020

Pair Programmer 1 (print & sign your name, then date it)

Hsuan-Yu Liu 4/14/2020

Pair Programmer 2 (print & sign your name, then date it)

Hsuan Yu Liu 823327369

Part 1

Question 1: Show what happens to the variable domains when forward-checking is employed after each of the first two decisions:

a. CA is assigned B.

b. AZ is assigned O.

	CA	NV	UT	AZ	CO	NM	TX
Initial Domains	R B Y O	R B Y O	R B Y O	R B Y O	R B Y O	R B Y O	R B Y O
CA is assigned B	B	R Y O	R B Y O	R Y O	R B Y O	R B Y O	R B Y O
AZ is assigned O	B	R Y	R B Y	O	R B Y	R B Y	R B Y O

a. NV and AZ remove 'B' in their domains.

b. NV, UT, and NM remove 'O' in their domains

Question 2: Using maintaining arc consistency instead of forward checking.

	CA	NV	UT	AZ	CO	NM	TX
Initial Domains	R B Y O	R B Y O	R B Y O	R B Y O	R B Y O	R B Y O	R B Y O
CA is assigned B	B	R Y O	R B Y O	R Y O	R B Y O	R B Y O	R B Y O
AZ is assigned O	B	R Y	R B Y	O	R B Y	R B Y	R B Y O

a. Check (CA, NV)→(CA, AZ)→(AZ, NM) → (AZ, CO) → (AZ, UT) → (AZ, NV) → (NV, UT) → (NV, AZ): NV and AZ remove 'B' in their domains.

b. Check (AZ, NV)→(AZ, UT) →(AZ, CO)→(AZ, NM)→ (NV, UT) → (UT, CO) →(CO, NM) → (NM, CO)→(NM,TX): NV, UT, CO, and NM remove 'O' in their domains

---

Question 3: Show how the conflict sets would be updated for these two moves if we were using conflict-directed backjumping.

{}← Conflict set ; ◯← Assigned

Conflict set	CA	NV	UT	AZ	CO	NM	TX
Initial set							
CA is assigned B	B	{ CA= B }		{ CA= B }			
AZ is assigned O	B	{ CA= B, AZ=O }	{ AZ=O }	O	{ AZ=O }	{ AZ=O }	

Question 4: Suppose students have the following set of simplified constraints:

i. Students cannot be enrolled in overlapping courses.
ii. Students cannot enroll in classes on days that they work.
iii. A student may not enroll in a course that does not have space.

a. Write a specification for this constraint satisfaction problem.

Variables:
Divide a student's timetable into four parts.

1. MW 14:00~15:15 → S_iMW1 (A period, 14:00 to 15:15, of Student i on Monday and Wednesday)

2. MW 15:00~16:15 → S_iMW2 (A period, 15:00 to 16:15, of Student i on Monday and Wednesday)

3. TTh 14:00~15:30 → S_iTTh2 (A period, 14:00 to 15:30, of Student i on Tuesday and Thursday)

4. TTh 18:00~19:15 → S_iTTh2 (A period, 18:00 to 19:15, of Student i on Tuesday and Thursday)

One student has 4 periods, so we have i*4 variables.

For example,
Variables= { S_1MW1, S_1MW2, S_1TTh1, S_1TTh2, S_2MW1, S_2MW2, S_2TTh1, S_2TTh2, …}

Domain:
Four Courses, C_1, C_2, C_3 and C_4. One Empty, E, and a Work, W.

C_1: A course whose Tutorial is 1

E: The period is free.

C_2: A course whose Tutorial is 2

W: Students work either days.

C_3: A course whose Tutorial is 3

C_4: A course whose Tutorial is 4

For example,

S_1MW1 = W means S_1 works either Monday or Wednesday.

S_2MW1 = C_1 means S_2 take a course C_1.

Constraints:

1. MW1 and MW2 cannot take course simultaneously. If MW1 = C_1, the other must be E.

2. TTh1 and TTh2 cannot take course simultaneously. If TTh1 = C_1, the other must be E

3. If MW_x is W then the other must be W as well. → MW1 = W ↔MW2 = W

4. If TTh_x is W then the other must be W as well. → TTh1 = W ↔ TTh2 = W

5. The number of C_x cannot be more than 30. E.g. Cannot be more than 30 students taking C_1.

b. Show how encapsulation could be used to create a binarize the constraint between multiple students the enrollment limit.

i. Students cannot be enrolled in overlapping courses. & ii. Students cannot enroll in classes on days that they work.
Combine MW1 and WM2 into a set and TTh1 and TTh2 into a set. The elements in a set cannot be two C_i or more. Also, if an element is W, the other must be W too.
E.g. $U_{i,ii}$ = [{ S_1MW1= C1, S_1MW2= C2 }, { S_1MW1= C1, S_1MW2= E }, { S_1TTh1= C3, S_1TTh2= W }, { S_1TTh1= E, S_1TTh2= C4 }, …]
{ S_1MW1= C1, S_1MW2= C2 } = False
{ S_1MW1= C1, S_1MW2= E } = True
{ S_1TTh1= C3, S_1TTh2= W } = False
{ S_1TTh1= W, S_1TTh2= W } = True
{ S_1TTh1= E, S_1TTh2= W } = False
{ S_1TTh1= E, S_1TTh2= C4 } = True

iii. A student may not enroll in a course that does not have space:

We can combine all S_xMW1 to be a set and count how many C1. If the number of C1 is larger than 30 then it conflicts. Following this rule, we can combine different MW and TTh to be sets respectively.

E.g. U_{iii} = [{S_1 = C1, S_2 =C1, S_3 = C1, S_4 = E, S_5 =E …, S_i =W…, S_j =C1…}, {S_1 = C2, S_2 =C2, S_3 = C2, S_4 = E, S_5 =E …, S_i =W…, S_j =C2…}, {S_1 = C3, S_2 =C3, S_3 = C3, S_4 = E, S_5 =E …, S_i =W…, S_j =C3…}, {S_1 = C4, S_2 =C4, S_3 = C4, S_4 = E, S_5 =E …, S_i =W…, S_j =C4…}].

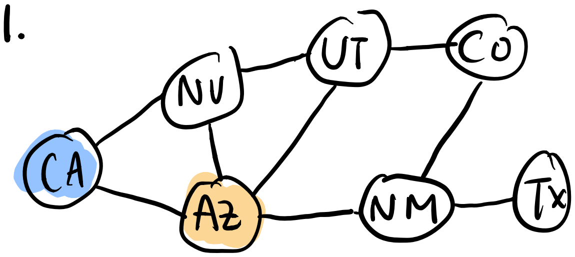
If {S_1 = C1, S_2 =C1, S_3 = C1, S_4 = E, S_5 =E …, S_i =W…, S_j =C1…} contains more than 30 C1, then it is failure.

If {S_1 = C2, S_2 =C2, S_3 = C2, S_4 = E, S_5 =E …, S_i =W…, S_j =C2…} contains more than 30 C2, then it is failure.

If {S_1 = C3, S_2 =C3, S_3 = C3, S_4 = E, S_5 =E …, S_i =W…, S_j =C3…} contains more than 30 C3, then it is failure.

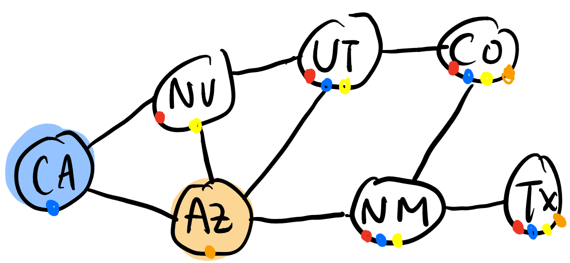
Jinqi Cheng Assignment5 PartI

1.



	CA	NV	AZ	UT	NM	CO	TX
initial domains	RBYO	RBYO	RBYO	RBYO	RBYO	RBYO	RBYO
After CA = B	B	RYO	RYO	RBYO	RBYO	RBYO	RBYO
After AZ = O	B	RY	O	RBY	RBY	RBYO	RBYO

2.



	CA	NV	AZ	UT	NM	CO	TX
initial domains	RBYO	RBYO	RBYO	RBYO	RBYO	RBYO	RBYO
After CA = B	B	RYO	RYO	RBYO	RBYO	RBYO	RBYO
After AZ = O	B	RY	O	RBY	RBY	RBYO	RBYO

After CA = B

Quene = (CA, NV) (CA, AZ),

NV = RYO, AZ = RYO

After AZ = O

Queue = (AZ, NV), (AZ, UT), (AZ, NM)

NV = RY, UT = RBY, NM = RBY

3.

	CA	NV	AZ	UT	NM	CO	TX
After CA = B	B	{CA = B}	{CA = B}	∅	∅	∅	∅
After AZ = O	B	{CA = B, AZ = O}	O	{AZ = O}	{AZ = O}	∅	∅

4.

a. Assume we have 2 classes: Student and Course

```
class Student {
    self.name = " "        # Student's name
    self.courses = [ ]     # Student's course list
    self.workday = [ ]     # Days this student work

class Course {
    self.name = " "        # Course name
    self.time = " "        # Start & end time of the course
    self.day = " "         # day of the course
    self.capacity = 30     # Capacity of the course
    self.size = 0          # Current size of the course
```

CSP Specifications:

$X = \{$Student1, Student2, ..., Student n$\}$ # instances of Student

$D_x = \{$All student names, course lists, workdays$\}$ # instance variables of Student

$Y = \{$Course 1, Course 2, Course 3, Course 4$\}$ # instances of Course, total 4.

$D_y = \{$courses name, time, day, capacity, size$\}$ # instance variables of Student

c = {

```
for ∃ student in X:          # the only overlapping courses are 1 and 3
    if 1 in Student.courses and 3 in Student.Courses:
        return False

for ∃ course in Y:           # in case the class is full
    if course.size > course.capacity:
        return False

for ∃ student in X:
    for ∃ course in Y:       # in case students need to work
        if student.workday overlaps course.day:
            return False
```

b. Create an encapsulated variable U

Cartisan product $U = X \times Y$

We can binarize the constraint by multiplying X and Y.

Hsuan-Yu Liu & Jinqi Cheng A5 PartII Source code

driver.py

```
from csp_lib.sudoku import (Sudoku, easy1, harder1)
from constraint_prop import AC3
from csp_lib.backtrack_util import mrv, forward_checking, first_unassigned_variable, unordered_c
from backtrack import backtracking_search

for puzzle in [easy1, harder1]:
    s = Sudoku(puzzle)  # construct a Sudoku problem
    print("---initial state---")
    s.display(s.infer_assignment())
    print("\n")
    AC3(s)

    print("---after AC3---")
    s.display(s.infer_assignment())
    print("\n")

    if not s.goal_test(s.curr_domains):
        solved = backtracking_search(s, select_unassigned_variable=mrv, inference=forward_checki
        print("---after backtracking---")
        s.display(solved)
```

backtrack.py

```python
from csp_lib.backtrack_util import (first_unassigned_variable,
                                    unordered_domain_values,
                                    no_inference)

def backtracking_search(csp,
                        select_unassigned_variable=first_unassigned_variable,
                        order_domain_values=unordered_domain_values,
                        inference=no_inference):
    """backtracking_search
    Given a constraint satisfaction problem (CSP),
    a function handle for selecting variables,
    a function handle for selecting elements of a domain,
    and a set of inferences, solve the CSP using backtrack search
    """
    # See Figure 6.5] of your book for details

    def backtrack(assignment):
        """Attempt to backtrack search with current assignment
        Returns None if there is no solution.  Otherwise, the
        csp should be in a goal state.
        raise notImplemented
        """

        """
        # MRC will throw an error as all variables have been assigned.
        Thus, return solution when all variables are assigned.
        """
        if len(assignment)==81:
            return assignment

        unassigned_var = select_unassigned_variable(assignment, csp)
        if unassigned_var is not None:
            for domain in order_domain_values(unassigned_var, assignment, csp):
                removals=[]
                csp.assign(unassigned_var, domain, assignment)
                infer = inference(csp, unassigned_var, domain, assignment, removals)
                if infer:
                    assignment = backtrack(assignment)
                    if csp.goal_test(assignment):
                        return assignment
                csp.unassign(unassigned_var, assignment)
                csp.restore(removals)
        return assignment

    # Call with empty assignments, variables accessed
    # through dynamic scoping (variables in outer
    # scope can be accessed in Python)
    result = backtrack({})
    assert result is None or csp.goal_test(result)
    return result
```

constraint_prop.py

```python
'''
Constraint propagation
'''
def AC3(csp, queue=None, removals=None):
    """AC3 constraint propagation
    # Hints:
    # Remember that:
    #    csp.variables is a list of variables
    #    csp.neighbors[x] is the neighbors of variable x
    raise NotImplemented
    """
    if not queue:
        variables = csp.variables
        queue = [ (var, neighbor) for var in variables for neighbor in csp.neighbors[var] ]

    while(queue):
        arc = queue.pop()
        if revise(csp, arc[0],arc[1]):
            if not csp.curr_domains[arc[0]]:
                break
            else:
                for neighbor in csp.neighbors[arc[0]]:
                  if neighbor != arc[1]:
                      queue.append((neighbor,arc[0]))
    return csp

def revise(csp, Xi, Xj):
    revised = False
    assignment = csp.infer_assignment()
    domains = csp.curr_domains[Xi]
    for d_val in domains:
        if csp.nconflicts(Xi,d_val, assignment):
            csp.prune(Xi, d_val, None)
            revised= True
    return revised
```

Output

```
---initial state---
. . 3 | . 2 . | 6 . .
9 . . | 3 . 5 | . . 1
. . 1 | 8 . 6 | 4 . .
------+-------+------
. . 8 | 1 . 2 | 9 . .
7 . . | . . . | . . 8
. . 6 | 7 . 8 | 2 . .
------+-------+------
. . 2 | 6 . 9 | 5 . .
8 . . | 2 . 3 | . . 9
. . 5 | . 1 . | 3 . .


---after AC3---
4 8 3 | 9 2 1 | 6 5 7
9 6 7 | 3 4 5 | 8 2 1
2 5 1 | 8 7 6 | 4 9 3
------+-------+------
5 4 8 | 1 3 2 | 9 7 6
7 2 9 | 5 6 4 | 1 3 8
1 3 6 | 7 9 8 | 2 4 5
------+-------+------
3 7 2 | 6 8 9 | 5 1 4
8 1 4 | 2 5 3 | 7 6 9
6 9 5 | 4 1 7 | 3 8 2


---initial state---
4 1 7 | 3 6 9 | 8 . 5
. 3 . | . . . | . . .
. . . | 7 . . | . . .
------+-------+------
. 2 . | . . . | . 6 .
. . . | . 8 . | 4 . .
. . . | . 1 . | . . .
------+-------+------
. . . | 6 . 3 | . 7 .
5 . . | 2 . . | . . .
1 . 4 | . . . | . . .


---after AC3---
4 1 7 | 3 6 9 | 8 2 5
. 3 . | . . . | . . .
. . . | 7 . . | . . .
------+-------+------
. 2 . | . . . | . 6 .
. . . | . 8 . | 4 . .
. . . | . 1 . | . . .
------+-------+------
```

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


---after backtracking---
4 1 7 | 3 6 9 | 8 2 5
6 3 2 | 1 5 8 | 9 4 7
9 5 8 | 7 2 4 | 3 1 6
------+-------+------
8 2 5 | 4 3 7 | 1 6 9
7 9 1 | 5 8 6 | 4 3 2
3 4 6 | 9 1 2 | 7 5 8
------+-------+------
2 8 9 | 6 4 3 | 5 7 1
5 7 3 | 2 9 1 | 6 8 4
1 6 4 | 8 7 5 | 2 9 3
```