File - C:\Users\Matthias\Documents\Spring 2018\Ai\Assignments\Assignment04\constraint_prop.py

```
import queue as que
from csp_lib import sudoku
#from types import TupleType
'''
Constraint propagation
'''
def AC3(csp, queue=None, removals=None):
    """
    :param csp:      constraint satisfaction problem
    :param queue:    List of constraints, if queue = none; make all constraint arcs
                     Otherwise, When AC3 called from the mac function,
                     mac populates the queue to only look at the neighbors of the variable that we are assigning.
    :param removals: List of variables and values that have been pruned. This is only useful for
                     backtracking search which will enable us to restore things to a former point.

    :return: True  - All constraints have been propagated and hold
             False - A variables domain has been reduced to the empty set through constraint propagation.
                     The problem cannot be solved from the current configuration of the csp
    """

    ############################################################
    # Construct arcs and add to queue
    # Note:
    #    - csp.variables is a list of variables
    #    - csp.neighbors is a dict
    #        where the key is a variable number 0 through 80,
    #        and the value is a set of the constraints that a particular variable has
    #        in terms of location on the board (row constraints, col constraints, box constraints) as variable numbers
    #        i.e. csp.neighbors[x] is the neighbors of variable x
    #    - e.neighbors[i] is a set which is a non iterable data type
    ############################################################
    def set_up_queue(csproblem, queue):
        """This method sets up arc constraints and stores them inside of a queue as tuples"""
        '''
        # OLD WAY
        for i in range(len(csproblem.variables)):
        temp_variables_value = csproblem.variables[i]  # Note: temp_variables_value is a variable 0-80, changes as I changes
        temp_set = csproblem.neighbors[i].copy()  # SOLUTION: If I didn't copy then I would have all empty sets in csp.neighbors
        assert isinstance(temp_set, set)  # temp_set gets callable methods it deserves if you're having scope problems
        temp_set_original_size = len(csproblem.neighbors[i])
        for element in range(temp_set_original_size):
            temp_constraint = temp_set.pop()
            temp_tuple = (temp_variables_value, temp_constraint)
            temp_queue.put(temp_tuple)
        return temp_queue
        '''
        temp_queue = que.Queue()
        if queue is None:
            for var in csproblem.variables:
                for neighbor in csproblem.neighbors[var]:  # note this will not remove set items
                    temp_queue.put((var, neighbor))
        else:
            for item in queue:
                #assert type(item) is TupleType
                assert isinstance(item, tuple), "%r is not a tuple" % item
                temp_queue.put(item)
        return temp_queue

    ############################################################
    # Global Variables
    ############################################################
    consistency = None
    if queue is None:
        q = set_up_queue(csp, queue)
    else:
        q = set_up_queue(csp, queue)

    csp.support_pruning()

    while not q.empty():
        (X_i, X_j) = q.get()
```

```
72          if revise(csp, X_i, X_j, removals):
73              if len(csp.curr_domains[X_i]) ==  :
74                  return False
75              for x_k in csp.neighbors[X_i]:
76                  if x_k != X_j:
77                      q.put((x_k, X_i))
78              # consistency = True , could be issue
79          consistency = True
80      return consistency
81
82  def revise(csp, X_i, X_j, removals):  #if removals is none?
83      """Return true if we remove a value, False otherwise.
84      Given a pair of variables Xi, Xj, check for each value x in Xi's domain
85      if there is some value y in Xj's domain that does not violate the constraints then remove that x
    value from csp.curr_domain[xi].
86
87      csp - constraint satisfaction problem Xi, Xj - Variable pair to check
88      removals - list of removed (variable, value) pairs. When value i is pruned from Xi, the constraint
    satisfaction
89                  problem needs to know about it and possibly updated the removed list (if we are
    maintaining one)
90      """
91      revised = False
92      temp_curr_domain_X_i = tuple(csp.curr_domains[X_i])  # todo - ask if this is needed
93      for x in temp_curr_domain_X_i:  # Error: csp.curr_dom[x_i] is changing; Solution: make copy of (csp
    .curr_domains[X_i]) i.e. temp_curr_domain_X_i = csp.curr_domains[X_i]
94              if not any([csp.constraints(X_i, x, X_j, y) for y in csp.curr_domains[X_j]]):
95                  # note we can utilize (y in csp.curr_domains[X_j]) because we are not manipulating (csp
    .curr_domains[X_j]) in code below
96                  #csp.curr_domains[X_i].remove(x)  This doesn't handle removals so use prune
97                  csp.prune(X_i, x, removals)
98                  revised = True
99      return revised
```