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_d
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

```
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

```
'''
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
```