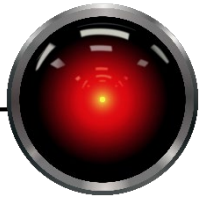
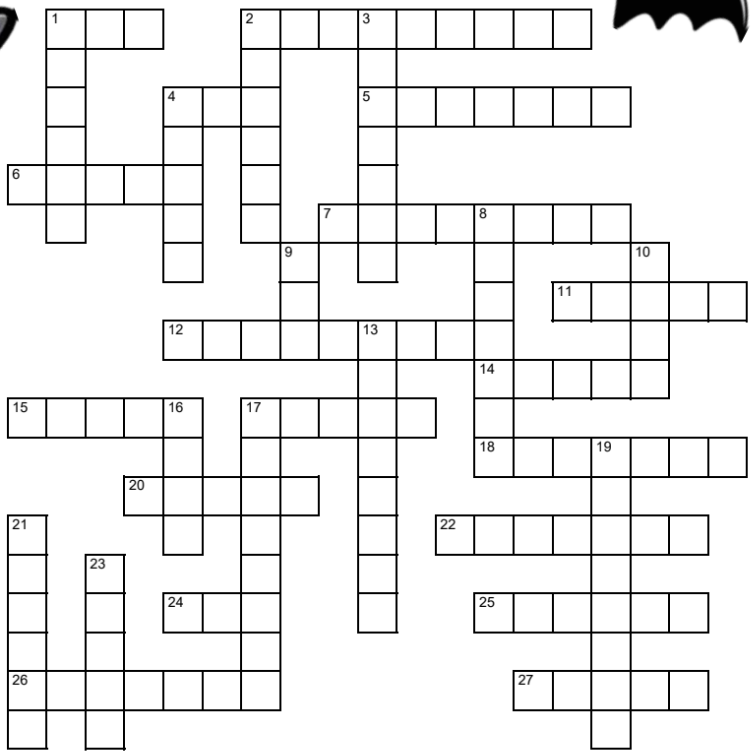


A3

Part I: 20 points each

1. Consider the following children's crossword puzzle from puzzles-to-print.com:

Halloween



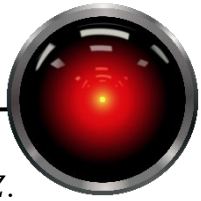
Across

1. He swallowed the canary.
2. Playground for ghosts.
4. What a spider spins.
5. Frankenstein has one.
6. When ghosts come out to play.
7. Scare.
11. What the pot might call the kettle.
12. October 31st.
14. ____ or treat.
15. Witch transportation.
17. Frightening.
18. The Count.
20. A skeleton is just a bunch of these.
22. Disguise.
24. Lives in the belfry.
25. Incey wincey is one of these.
26. Main ingredient in a popular pie.
27. ____ stories.

Down

1. Where a vampire sleeps.
2. Evil or mischievous creature.
3. He hates garlic.
4. Samantha for example.
8. ____ house.
9. Whoo? Whoo?
10. Mr. O'Lantern.
13. Comes out on full moon nights.
16. Might be full, half, or new.
17. A boney sort of fellow.
19. Fire burn, and ____ bubble.
21. When something makes our skin crawl, it's this.
23. Found in Egypt.

Ignoring the constraints introduced by the puzzle hints, explain how you would define a constraint satisfaction problem for this problem.

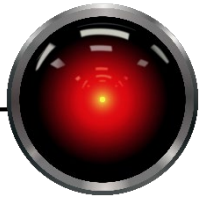


2. A constraint satisfaction problem has a global constraint: $2 \cdot X = Y \cdot Z$. The domains of X, Y, and Z are respectively $\{4, 5, 6\}$, $\{3, 4\}$, and $\{3, 4\}$. Show how you can use encapsulation to write this as a binary constraint between X and an encapsulated variable.

For the next questions, consider a 3-color map problem on Frank Baum's fictional land of Oz introduced in *The Wizard of Oz* in 1900. We wish to color the map with colors G (green), B (blue) and P (purple).

The map below (© MGM 2008, used under educational fair use) shows the five regions: Winkie Land, Gillikin Land, Munchkin Land, Quadling Land, and the Emerald City. Abbreviate the lands as W, G, M, Q, and E.

3. Draw a constraint hypergraph for the land based on territories that physically touch one another (do not use the road system).
4. Suppose that you are given that Q=G and W=P. Show the initial queue for the AC3 algorithm and simulate the algorithm.
5. Show how the assignments of Q=G and W=P restrict the domains of the other variables using forward checking.
6. What would the conflict sets be from problem 5?



Part II – Sudoku (120 points)

The code package for this assignment contains a partially implemented constraint satisfaction problem for solving Sudoku puzzles. You will need to implement functionality in three modules:

- driver – Solves Sudoku puzzles. Two puzzles are provided for you. When solving these puzzles, you should first run the AC3 algorithm to propagate any constraints from the values that have been set. Print out the puzzle before starting inference and then after inference is completed. If the puzzle is not solved, a backtracking search should be run. The backtracking search should use the minimum remaining value heuristic for variable selection, unordered domain values, and the maintaining arc consistency (MAC) inference algorithm. Implementations for variable selection and ordering can be found in csp_lib.backtrack_util.

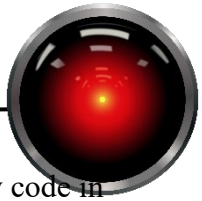
  The easier of the two problems can be solved entirely by inference without any searching.

- backtrack – Implement a backtracking search. Your backtracking algorithm should take the following parameters:
  - csp – An instance of a constraint satisfaction problem (CSP), in this assignment you will pass an instance of CSP subclass csp_lib.sudoku.Sudoku.
  - A handle to a subroutine to select unassigned variables
  - A handle to a subroutine to order values within a variable domain
  - A handled to an inference algorithm
  - A verbose (True/False) flag. When True, the number of inferences and assignments made should be printed (along with any debugging messages).

  and return
  - a dictionary of assignments (variables are keys) if a solution is found or None if no solution is possible
  - the number of variable assignments that were made.

- constraint_prop – implement the AC3 algorithm. It takes the following parameters:
  - csp – An instance of a CSP problem.
  - queue – A list of variable pairs to process. For AC3, this should be set to None and the AC3 function will generate an appropriate list. Allowing users to pass in lists of variables lets this algorithm be used for MAC as well.
  - removals – If not None, expects a list of variables and values. As each value is removed from a domain, the removals list should be updated.

    Hint: You may find the csp.prune method useful.



Several modules are provided in package csp_lib. You do not need to modify code in these modules, but you will need to read them in order to understand how to interact with the Sudoku CSP representation.

The sudoku module implements the class Sudoku that is derived from a CSP class (both in module csp_lib). The code and comments explain the data structures that are used and contain a sample instantiations of two Sudoku puzzles. In general, the CSP class methods will be very useful for this problem. Some of the ones that you might want to pay attention to are (not an exhaustive list):

- display – Show the puzzle with any
- goal_test – Tests whether or not the problem is solved.
- suppose – Given a variable and value, restrict the domain to the value and return a list of removed variables and values; e.g., if we assign val3 to var, we would get back [(var,val1), (var,val2), (var, val4), …] and the csp object would be modified to have val3 the sole member of the domain. Note that this is different from assign which does not return information that will let you restore the state later (don't use assign unless you want to do a lot of extra bookkeeping yourself). Note that the suppose is not the only place that domains are restricted, your inference algorithms can do this
- prune – Given a var name and a value, remove it from the domain. An optional list of (variable, value) tuples that have been removed will be appended to if provided. This is useful for tracking inference that occurred in backtracking.
- infer_assigment – Return variables that can be currently assigned based on their domains.
- restore – Given a removal list (see suppose), restore the values to domains

As with your last assignment, setting breakpoints and stepping through functions can be very helpful for understanding. Module csp.backtrack_util provides a set of utility functions for backtracking. You will not need most of them, but you will need one of the variable ordering routines and the maintaining arc consistency (mac) routine. If you are confused about how the AC3 interface works, you might want to pay attention to mac as it calls AC3 with a smaller queue.

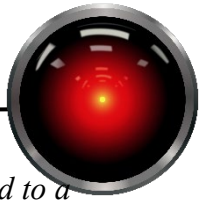
Your driver program should create both the easy and hard sudoku problems and then solve them.

Output: Your code should show for each puzzle:

- Initial sudoku state
- The state of the puzzle after running AC3.
- If the puzzle is not in a goal state (the Sudoku class provides a goal test that can be used on the instances current domains), then run a backtracking search and show the solution.

Part II Submission:

- Modules: backtrack, constraint_prop, and driver (with affidavit)



- Text file with output from your program run. *Use a text file as opposed to a screen shot.*