Computer Science 384
St. George Campus

February 3, 2020
University of Toronto

Homework Assignment #2: Constraint Satisfaction
Due: February 25, 2020 by 10:00 PM

Silent Policy: *A silent policy will take effect 24 hours before this assignment is due, i.e. no question about this assignment will be answered, whether it is asked on the discussion board, via email or in person.*

Late Policy: 10% per day after the use of 3 grace days.

Total Marks: This part of the assignment represents 11% of the course grade.

Handing in this Assignment

What to hand in on paper: Nothing.

What to hand in electronically: You must submit your assignment electronically. Download `csp.zip` which contains `propagators.py` and `futoshiki_csp.py` and other related files from the A2 web page. Modify `propagators.py` and `futoshiki_csp.py` so that they solve the problems specified in this document. **Submit your modified files** `propagators.py` **and** `futoshiki_csp.py` **as well as** `acknowledgment_form.pdf` You will submit your assignment using MarkUs. Your login to MarkUs is your teach.cs username and password. It is your responsibility to include all necessary files in your submission. You can make as many submissions as you like while you still have grace days; the number of grace days you use will be determined by the time of your final submission.

We will test your code electronically. You will be supplied with a testing script that will run a **subset** of the tests. If your code fails all of the tests performed by the script (using Python version 3.7), you will receive zero marks. It's up to you to figure out further test cases to further test your code – that's part of the assignment!

When your code is submitted, we will run a more extensive set of tests which will include the tests run in the provided testing script and a number of other tests. You have to pass all of these more elaborate tests to obtain full marks on the assignment.

Your code will not be evaluated for partial correctness, it either works or it doesn't. It is your responsibility to hand in something that passes at least some of the tests in the provided testing script.

- *Make certain that your code runs on teach.cs using python3 (version 3.7) using only standard imports.* This version is installed as "python3" on teach.cs. Your code will be tested using this version and you will receive zero marks if it does not run using this version.
- *Do not add any non-standard imports from within the python file you submit (the imports that are already in the template files must remain).* Once again, non-standard imports will cause your code to fail the testing and you will receive zero marks.
- *Do not change the supplied starter code.* Your code will be tested using the original starter code, and if it relies on changes you made to the starter code, you will receive zero marks.

Clarification Page: Important corrections (hopefully few or none) and clarifications to the assignment will be posted on the Assignment 2 Clarification page, linked from the CSC384 A2 web page, also

found at: `http://www.teach.cs.toronto.edu/~csc384h/winter/Assignments/A2/a2_faq.html`. You are responsible for monitoring the A2 Clarification page.

Questions: Questions about the assignment should be asked on Piazza:

`https://piazza.com/utoronto.ca/winter2020/csc384/home`.

You may also reach out to the Assignment 2 TAs, Chris Karavasilis (ckar at cs.utoronto.ca) and Torin Viger (torin.viger at mail.utoronto.ca), or one of the instructors. Please place "A2" and "CSC384" in the subject line of your email.

Introduction

There are two parts to this assignment

1. the implementation of two constraint propagators and a variable ordering heuristic – a Forward Checking constraint propagator, a Generalized Arc Consistence (GAC) constraint propagator, and the Minimum Remaining Values (MRV) heuristic,

2. the encoding of two different CSP models to solve the logic puzzle, Futoshiki, as described below. In one model you will use only binary not-equal constraints, while in the other model you will use 9-ary all-different constraints in addition to binary not-equal constraints.

What is supplied:

- **cspbase.py** – class definitions for the python objects Constraint, Variable, and BT.
- **propagators.py** – starter code for the implementation of your two propagators and variable ordering heuristic. You will modify this file with the addition of three new procedures `prop_FC`, `prop_GAC`, and `ord_mrv`, to realize Forward Checking, GAC and MRV, respectively.
- **csp_sample_run.py** – This file contains a sample implementation of two CSP problems.
- **futoshiki_csp.py** – starter code for the two Futoshiki CSP models.
- **propagators_test.py** for testing your code for FC and GAC.

Futoshiki Formal Description

The Futoshiki puzzle[1] has the following formal description:

- The puzzle is played on a grid *board* with dimensions n rows by n columns. The board is always a square.
- The board has n^2 spaces on it, where each space may take a value between 1 and n inclusive. You will use a list of lists in order to represent assigned values to the variables on this grid.
- The start state of the puzzle may have some spaces already filled in.

[1] `https://en.wikipedia.org/wiki/Futoshiki`

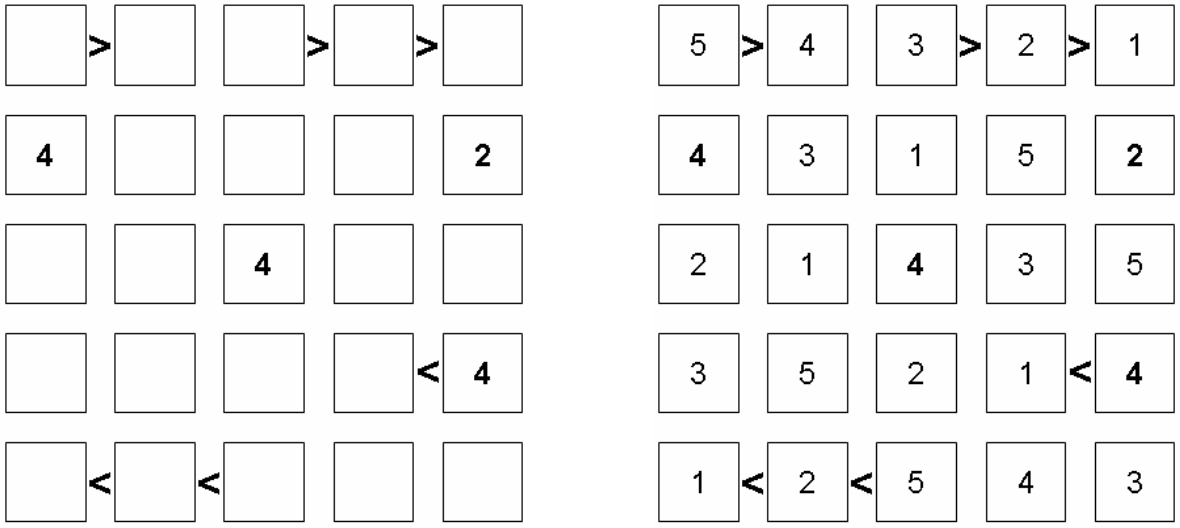

Figure 1: An example of a 5×5 Futoshiki grid in its initial state (left) and solution (right).

- Additionally, the starting board may have some *inequality constraints* specified between some of the spaces. These inequality constraints will *only* apply to two cells that are horizontally adjacent to one another, i.e. the constraints will only apply to rows and not columns. If there is *no* inequality constraint between two adjacent cells, this will be represented with a dot (i.e., with a '.'). For example, consider this 2 x 2 board: $[[0,<,0,],[0,.,0]]$. The assignments $[[1,<,2,],[2,.,1]]$ would satisfy the inequality constraint.
- A puzzle is *solved* if:
  - Every space on the board is given one value between 1 and *n* inclusive.
  - All specified inequality constraints are satisfied.
  - No row contains more than one of the same number.
  - No column contains more than one of the same number.

An example of a Futoshiki instance and its solution are depicted in figure 1. You can also play Futoshiki online[2] but column inequality constraints might be included.

Question 1: Propagators **(worth 60/100 marks)**

You will implement python functions to realize two constraint propagators – a Forward Checking constraint propagator and a Generalized Arc Consistence (GAC) constraint propagator. These propagators are briefly described below. The files `cspbase.py` and `propagators.py` provide the **complete input/output specification** of the functions you are to implement. In all cases, the CSP object is used to access variables and constraints of the problem, via methods found in cspbase.py.

Brief implementation description: A `Propagator Function` takes as input a CSP object `csp` and (optionally) a variable `newVar`. The CSP object is used to access the variables and constraints of the problem (via methods found in `cspbase.py`). A propagator function returns a tuple of `(bool,list)` where `bool`

[2] https://www.futoshiki.org/

is False if and only if a dead-end is found, and `list` is a list of `(Variable, value)` tuples that have been pruned by the propagator. `ord_mrv` takes a CSP object csp as input, and returns a Variable object var. You must implement: **You must implement:**.

`prop_FC` (worth 25/100 marks)
: A propagator function that propagates according to the Forward Checking (FC) algorithm that check constraints that have *exactly one uninstantiated variable in their scope*, and prune appropriately.

  If `newVar` is `None`, forward check all constraints. Else, if `newVar=var` only check constraints containing `newVar`.

`prop_GAC` (worth 25/100 marks)
: A propagator function that propagates according to the Generalized Arc Consistency (GAC) algorithm, as covered in lecture. If `newVar` is `None`, run GAC on all constraints. Else, if `newVar=var` only check constraints containing `newVar`.

`ord_mrv` (worth 10/100 marks)
: A variable ordering heuristic that chooses the next variable to be assigned according to the Minimum Remaining Values (MRV) heuristic. ord mrv returns the variable with the most constrained current domain (i.e., the variable with the fewest legal values).

Question 2: Futoshiki Models **(worth 40/100 marks)**

You will implement two different CSP encodings to solve the logic puzzle, Futoshiki. In one model you will use only binary not-equal constraints, while in the other model you will use n-ary all-different constraints in addition to binary not-equal constraints. These CSP models are briefly described below. The file `futoshiki_csp.py` provides the **complete input/output specification** for the two CSP encodings you are to implement.

The correct implementation of each encoding is worth 20/100 marks.

Brief implementation description: A `Futoshiki Model` takes as input a Futoshiki board, and returns a `CSP object`, consisting of a variable corresponding to each cell of the board. The variable domain of that cell is $\{1,...,n\}$ if the board is unfilled at that position, and equal to i if the board has a fixed number i at that cell. All appropriate constraints will be added to the board as well. You must implement:

`futoshiki_csp_model_1` A model built using only binary not equal constraints for the row and column constraints, and binary inequality constraints.

`futoshiki_csp_model_2` A model built using n-ary all-different constraints for the row and column constraints, and binary inequality constraints.

Caveat: The Futoshiki CSP models you will construct can be space expensive, especially for constraints over many variables, (e.g., those contained in the second Futoshiki CSP model). *HINT: Also be mindful of the **time** complexity of your methods for identifying satisfying tuples, especially when coding the second Futoshiki CSP model.*

HAVE FUN and GOOD LUCK!