Programming Assignment 3: Futoshiki as CSPs

Due Jul 27 by 10pm **Points** 100 **Available** after Jul 5 at 12am

Released: July 5, 2021

Due: July 27, 2021

Table of Contents.

1. Warning
2. Introduction
3. Futoshiki Formal Description
4. Question 1: Propagators
5. Question 2: Futoshiki Models
6. What To Submit

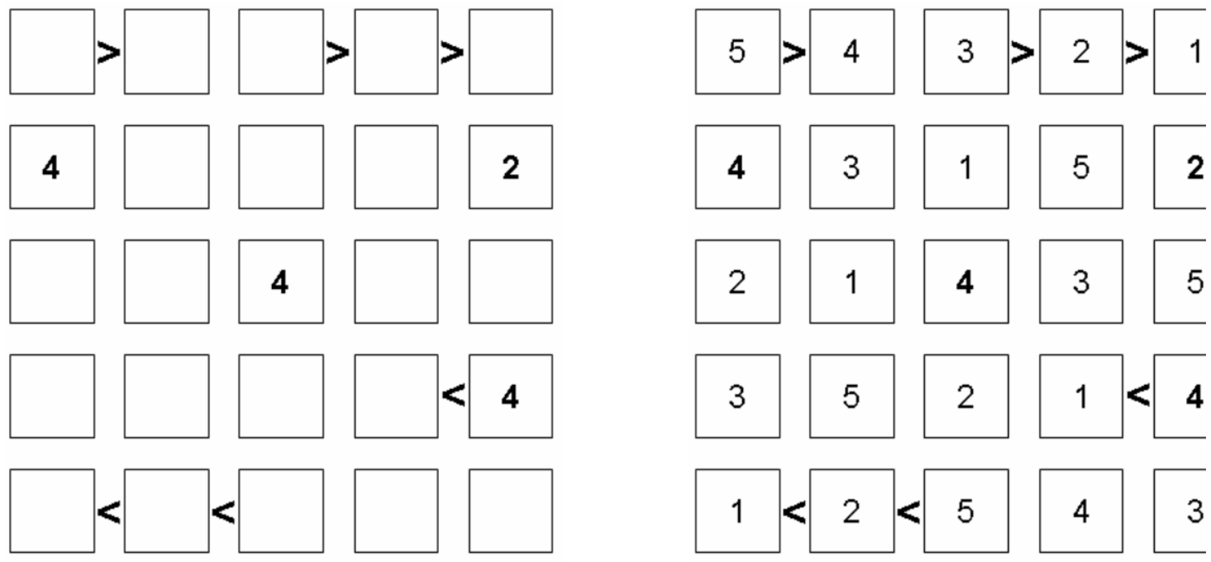

Figure 1: An example of a 5×5 Futoshiki grid in its initial state (left) and solution (right).

Warning (Please read this)

As ever, we are aware that solutions to this problem, or related ones, may exist on the internet. **Do not use these solutions as this would be plagiarism.** To earn marks on this assignment you must develop your own solutions. Also please consider the following points, as you did when implementing prior assignments:

- **Do not add any non-standard imports in the python files you submit** (all imports already in the starter code must remain). All imports that are available on teach.cs are considered to be standard.
- **Do not change any of the supplied files except for `propagators.py` and `futoshiki_csp.py`.**
- **Make certain that your code runs on teach.cs using python3**. You should all have an account on teach.cs and you can log in, download all of your code (including all of the supplied code) to a subdirectory of your home directory, and use the command `python3 autograder.py` and test it there before you submit. Your code will be graded by running it on teach.cs, so the fact that it runs on your

own system but not on teach is not a legitimate reason for a regrade.
- The test cases used in the autograder reflect the test cases we will use during marking. The test cases you have are of similar difficulty, so the grade shown to you by the autograder will be a reasonable predictor of your final grade on the assignment. We will also look for certain things in the assignments (e.g., running them through code plagiarism checkers, looking at assignments that fail all tests, etc.). If we have **good reasons** we will change your grade from that given by the autograder either up or down.

Introduction

There are two parts to this assignment

1. the implementation of two constraint propagators – a Forward Checking constraint propagator, and a Generalized Arc Consistence (GAC) constraint propagator,
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. You will modify this file with the addition of two new procedures `prop_FC` and `prop_GAC`, to realize Forward Checking and GAC, respectively.
- `futoshiki_csp.py` – starter code for the two Futoshiki CSP models.
- Sample test cases (in **automarker.py**) for testing your code.

You can download all of the starter code **at this link. (https://q.utoronto.ca/courses/219334/files/14968455/download?download_frd=1)**

Futoshiki Formal Description

The Futoshiki puzzle 1 has the following formal description: (**from this source (https://en.wikipedia.org/wiki/Futoshiki)**):

- 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.

- Ad ditionally, 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 (at **https://www.futoshiki.org/** (https://www.futoshiki.org/)) but note that column inequality constraints might be included in the games you find here.

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` 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 as input, and returns a Variable object `var`. 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.

What to Submit

You will be using MarkUs to submit your assignment. You will submit two files:

1. Your modified `propagators.py`
2. Your modified `futoshiki_csp.py`

HAVE FUN and GOOD LUCK!