Computer Science 384 June 12, 2017
St. George Campus University of Toronto

Homework Assignment #2: Constraint Satisfaction
Due: June 26, 2017 by 11:59 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 10% 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 the assignment files from the A2 web page. Modify `propagators.py` and `kenken_csp.py` appropriately so that they solve the problems specified in this document. **Submit your modified files `propagators.py` and `kenken_csp.py`.**

How to submit: If you submit before you have used all of your grace days, 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 submit a new version of any file at any time, though the lateness penalty applies if you submit after the deadline. For the purposes of determining the lateness penalty, the submission time is considered to be the time of your latest submission. More detailed instructions for using Markus are available at:
`http://www.cdf.toronto.edu/~csc384h/summer/markus.html`.

Warning: marks will be deducted for incorrect submissions.

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 3.5.2), 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 3.5.2) 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.

Evaluation Details: The details of the evaluation will be released as a separate document in approximately one week.

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/summer/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/summer2017/csc384/`.

If you have a question of a personal nature, please email the A2 TA, at ckar at cs dot toronto dot edu or the instructor, Sonya Allin, at sonyaa at teach dot cs dot utoronto dot edu placing 384 and A2 in the subject line of your message.

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, along with the variable ordering heuristic of Minimum Remaining Values (MRV),
2. the encoding of a CSP model to solve the logic puzzle, "Kenken", as described below.

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.
- **orderings.py** – starter code for the implementation of the variable ordering heuristic MRV. You will modify this file with the addition of the new procedure ord mrv to realize MRV.
- **kenken_csp.py** – starter code for the Kenken CSP model.
- **Sample test cases** - PENDING

Kenken Formal Description

Kenken puzzle[1] has the following formal description:

- Kenken consists of an *nxn* grid where each cell can be assigned a number 1 to *n*, so that no digit appears more than once in any row or column. Grids range in size from 3*x*3 to 9*x*9. Additionally, Kenken grids are divided into heavily outlined groups of cells often called cages and the numbers in the cells of each cage must produce a certain target number when combined using a specified

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

mathematical operation (either addition, subtraction, multiplication or division). What follows is an example of a Kenken problem instance with a corresponding solution:

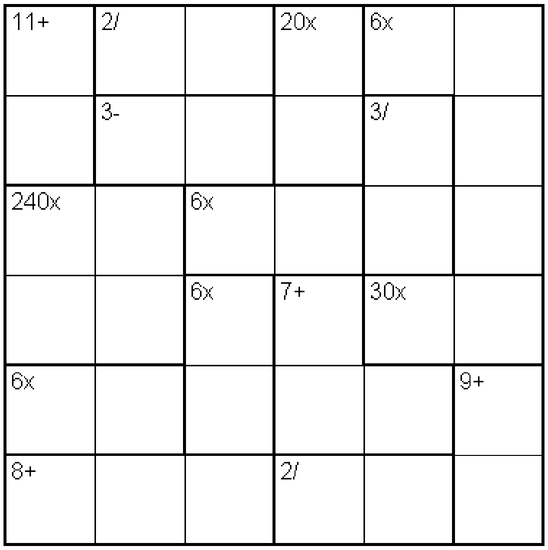

Figure 1: Kenken problem.

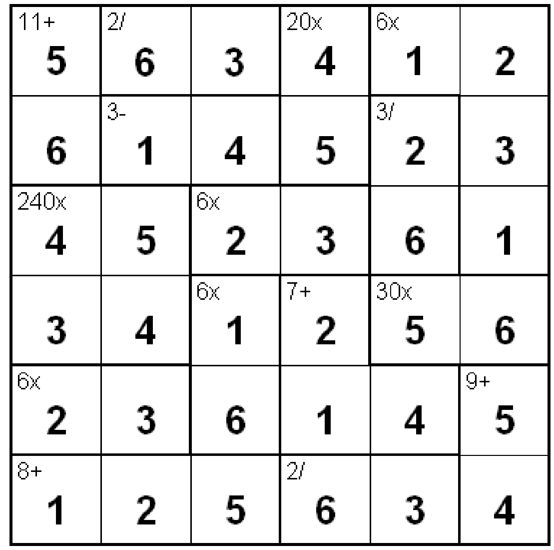

Figure 2: Solution to the Kenken problem.

Question 1: Propagators and Variable ordering (worth 65/100 marks)

You will implement python functions to realize two constraint propagators – a Forward Checking constraint propagator and a Generalized Arc Consistence (GAC) constraint propagator, and the variable ordering heuristic Minimum Remaining Values (MRV). These propagators and the variable ordering heuristic are briefly described below. The files `cspbase.py`, `propagators.py` and `orderings.py` provide the **complete input/output specification** of the three functions you are to implement.

The correct implementation of each function is worth 25/100 marks.

Brief implementation description: A `Propagator Function` takes as input a CSP object `csp` and (optionally) a variable `newVar` representing a newly instantiated variable. 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.
A `Variable Ordering Function` takes as input a CSP object `csp`, and returns a Variable object `var`. The CSP object is used to access variables and constraints of the problem, via methods found in `cspbase.py`. You must implement:

`prop_FC` 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` 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` A variable ordering heuristic that chooses the next variable to be assigned in backtracking search, according to the minimum remaining values (MRV) heuristic. MRV returns the variable with the most constrained current domain (i.e., the variable with the fewest legal values).

Question 2: Kenken Model (worth 35/100 marks)

You will implement a CSP encoding to solve the logic puzzle, Kenken. The CSP model is briefly described below. The file `kenken_csp.py` provides the **complete input/output specification** of the CSP encoding you are to implement.

Brief implementation description: A `Kenken Model` takes as input a Kenken Grid, which is a list of lists, where each list represents a cage in the grid. Cell names are encoded as integers in the range $11, \ldots, nn$ and each inner list contains the numbers of the cells that are included in the corresponding cage, followed by the final value and the operation (0="+", 1="-", 2="/", 3="*"). You must implement:

`kenken_csp_model` A model built using only binary not-equal constraints for the row and coloumn constraints, and n-ary constraints for the operation constraints.

Bonus: (how many marks) more binary constraints - PENDING

HAVE FUN and GOOD LUCK!