# Assignment 2: CSPs

CSC 384H—Winter 2015

Out: Feb. 23th, 2015
Due: Electronic Submission Friday 13th March, 7:00pm
Late assignments will not be accepted without medical excuse
Worth 15% of your final mark.

**Be sure to include your name and student number as a comment in all submitted documents.**

## Handing in this Assignment

***What to hand in on paper:*** No paper submission required.

***What to hand in electronically:*** You must submit all your answers and code electronically. You must submit the following:

1. A file of python code called **`sudoku_csp.py`**. This file will contain your implementation of two CSP models of Sudoku along with your routine for enforcing GAC.
2. A PDF file called **`a2_answers.pdf`**. Use any word processing software to produce your written answers and then convert to PDF.

To submit these files electronically, use the CDF secure Web site
`https://www.cdf.toronto.edu/students/`
or use the CDF **submit** command. Type **man submit** for more information. The name of the assignment for submit will be "A2"

Since we will test your code electronically, you must

- *make certain that your code runs on CDF using python3 (version 3.4.1* (installed as "python3" on CDF).
- not add any non-standard pythons imports from within the python files you submit (the imports that are already in the template files must remain).
- include all your written answers in the PDF file you create and submit (using the file name `a2_answers.pdf`). **NOTE the space limits specified for each.** If you exceed the space limits marks will be deducted.

(*Warning: marks will be deducted for errors in following the stated guidelines*):

## Introduction

In this assignment you will implement a procedure that given a list of constraints enforces GAC on them by pruning values from the variable domains. You will also implement two different CSP models of the Sudoku puzzle, and compare the effects of enforcing GAC on these models.

**What is supplied.** You will be supplied with python code implementing **Constraint** and **Variable** objects. The file `cspbase.py` contains the class definitions for these objects. The code supports representing constraints as a collection of satisfying tuples—so to specify a constraint in one has to construct the list of satisfying tuples and pass them to the constraint object.

Note that this representation can be space expensive, especially for constraints over many variables. For example, to represent an initially empty Sudoku board in Model-2 requires over 1 GB of RAM. However, for any board that has a typical number of fixed cells the memory requirements are reasonable.

You will also be supplied with a template file `sudoku_csp.py` which you will complete with your implementation.

## Question 1. Implement Enforce GAC: worth 20/100 marks

Implement a routine that takes a list of constraints (constraint objects) and enforces GAC by pruning values from the domains of the variables in its scope. See `sudoku_csp.py` for a more detailed specification of this function.

**To Submit**

1. Submit your python implementation in the file `sudoku_csp.py` (this file will also contain other parts of the assignment).

## Question 2. Implement Model 1: worth 30/100 marks

Implement the function `sudoku_enforce_gac_model_1` which takes as input an initial Sudoku board, and constructs a CSP model where all constraints are binary not-equals constraints. Define a variable for each cell of the board and construct the right collection of not-equals constraints so as to capture the rules of the puzzle. Then enforce GAC and return the reduced variable domains in the format specified.

**To Submit**

1. Submit your python implementation in the file `sudoku_csp.py` along with the rest of your implementation.

## Question 3. Implement Model 2: worth 40/100 marks

Implement the function `sudoku_enforce_gac_model_2` which takes the same input as `sudoku_enforce_gac_model_1`, but this time constructs a CSP model where each row, column, and sub-square has an **all-diff** constraint over its 9 variables. Enforce GAC and return the reduced variable domains in the format specified.

## Question 4. Using your Implementation: worth 10/100 marks

Use your two CSP models to simplify various Sudoku problems. There are many web-sites that have on-line puzzles of different degrees of difficulty. Test to see if Model-2 can do better than Model-1 or vice versa. In `a2_answers.pdf` submit answers to the following questions. Again observe the length limits on your answers and note that marks will be deducted

1. Does one the two models always reduce the variable domains more than the other? If so which one is more effective in pruning the variable domains. (One sentence answer).
2. How many different constraints are needed for Model-1 and for Model-2. (One sentence answer).
3. The most space expensive problem is when the Sudoku board is initially empty. In this case all variables will have a domain of size nine. (Initially filled in cells yield variables with domain sizes of 1). When the board is initially empty, how many satisfying tuples must be stored for each constraint in Model-1, and how many satisfying tuples must be stored for each constraint in Model-2. (One sentence answer).

4. Assuming that each element of each tuple stored in each constraint requires 4 bytes of space, what is the total space needed in bytes needed to represent constraints (only the constraints) arising from the initially empty board in Model-1 and in Model-2. (One sentence answer).