# CS 6364 Artificial Intelligence
# Spring 2024
# Assignment 2: Local Search, Encoding, CSPs, and Adversarial Search
# Part II: *Due electronically by Tuesday, March 19, 11:59 p.m.*

In this problem, you will implement a CSP solver that takes exactly three arguments from the command line:

1. A `.var` file that contains the variables in the CSP to be solved and their domains. Each line of the file contains a variable (represented by a single letter), followed by a colon and its possible values, each of which is an integer. For instance, the line "A: 1 2 3" indicates that the possible values for variable $A$ are 1, 2, and 3. Note that there is a space separating the domain values.

2. A `.con` file that contains the constraints. Each line corresponds to exactly one constraint, which involves two variables and has the form `VAR1 OP VAR2`. `VAR1` and `VAR2` are the names of the two variables involved, and `OP` can be one of four binary operators: = (equality), ! (inequality), $>$ (greater than), and $<$ (less than).

3. The consistency-enforcing procedure, which can take one of two values: `none` and `fc`. If `none` is used, no consistency-enforcing procedure is applied, and the solver simply uses backtracking to solve the problem. `fc` indicates that the solver will use forward checking to enforce consistency.

**Note:**

- The first two arguments are the *paths* to the .var file and the .con file. Do **not** assume that the .var file and the .con file are in the same directory as your code files.

- Whenever the solver needs to choose a *variable* during the search process, apply the most constrained variable heuristic, breaking ties using the most constraining variable heuristic. If more than one variable remains after applying these heuristics, break ties alphabetically.

- Whenever the solver needs to choose a *value* during the search process, apply the least constraining value heuristic. If more than one value remains after applying this heuristic, break ties by preferring smaller values.

Your program should allow exactly three arguments to be specified in the command line invocation of your program, in the order in which they are listed above. Sample input files will be available in the assignment page of the course website. Your program should write to `stdout` the branches visited in the search tree (or stop when the solution is reached). By branches we mean an assignment that violates a constraint or that leads to one or more unassigned variables having empty domains (when using forward checking). Write them in text form with each branch on one line. For example, suppose we had variables X and Y with domains $\{0, 1, 2\}$, variable Z with domain $\{1, 2\}$, and constraint Y=Z. The branches visited in the search tree without forward checking should be written as:

1. Z=1, X=0, Y=0 failure
2. Z=1, X=0, Y=1 solution

However, if forward checking is applied, the output should be:

1. Z=1, Y=1, X=0 solution

**Important:** Note that the printed branches should keep a consistent variable ordering. Variables should be printed in the order they are assigned values. Do not output anything else to `stdout`. Any program that does not conform to the above specification will receive no credit.

To implement the program, you should use Python (3.8.18), C++ (g++ 7.5.0), or Java (openjdk 11.0.13 2021-10-19). Do **not** use any non-standard libraries in your code (except for numpy-1.19.5 and pandas-1.1.5), and make sure your program can compile on UTD machines, not just your own machines. If you are using:

- **Python**
  - Make sure that your primary source file is `main.py` and that your code runs successfully after executing `python main.py <path_to_var_file> <path_to_con_file> <none|fc>`.
- **C++**
  - Make sure that your primary source file is `main.cpp` and that your code runs successfully after executing `g++ main.cpp -o a.out -std=c++17` and `./a.out <path_to_var_file> <path_to_con_file> <none|fc>`.
- **Java**
  - Make sure that your primary source file is `Main.java` and that your code runs successfully after executing `javac Main.java` and `java Main <path_to_var_file> <path_to_con_file> <none|fc>`.

## Submission

Once you are done, sign in to gradescope. You will be able to see *Assignment 2 - Part 2* under the Assignments section. Directly submit your files to this submission folder. Please **do not** rename the files or upload the files in a zip file or folder (your homework will not be graded otherwise). If you worked in a group, make sure to add your partner when you submit!

## Example Files

We will provide the corresponding .con and .var files for 3 test cases (ex1, ex2, and ex3), along with the correct output using both forward checking and no consistency enforcing procedure (fc, none). You should use these test cases to verify the basic functionality of your code and output formatting; however, remember that **we will test your code on hidden test cases**; so correctly solving ex1, ex2, and ex3 does not guarantee you will get all points.

## Grading

We will be using an autograder as the main grading mechanism for this submission. Keep in mind that:

- The autograder does not have any non-standard libraries installed (except for numpy-1.19.5 and pandas-1.1.5). So if you encounter the error "The autograder failed to execute correctly. Contact your course staff for help in debugging this issue. Make sure to include a link to this page so that they can help you most effectively", it's likely that you used some non-standard libraries in your code.
- OS: The autograder will be running Linux 22.04 LTS.
- Hardware Limitations: The auto-graders have a limited amount of resources, as these are dynamically allocated by gradescope servers. You can assume you will have at least 2 threads available and at most 3gb of ram; auto-graders will output a timeout error if your code takes longer than 10 minutes to go through the tests. Even if your code does not time out or run out of memory on your machine, it could very well do so when testing, since we may run tests on bigger/diffierent input les. Please take this into account and try to avoid bad code.

We encourage you to submit to Gradescope before the due date and debug your code according to the feedback given by the autograder. More information about the autograder will be available on Piazza shortly.

**Important note:** We will be using MOSS for plagiarism detection on all programming assignments (you can look it up, it's quite powerful). Any students caught cheating will receive a 0 on the corresponding assignment and will be reported accordingly.