CSC410

Assignment 4

Due on October 30, 2020 at 11:59pm

Assignment Format and Guidelines on Submission

This assignment is worth 10% of the total course mark. Submit on Markus and follow these rules:

- This is a group assignment, designed for groups of 4 students. The volume of work is designed so that it would be reasonable for a group of 4. You may choose to work in a group of 3. Groups of fewer than 3 students will be not be accepted. You cannot submit individually. You can use Piazza, Quercus, or the old-fashioned face-to-face to form groups.
- Each group should submit three files named `q1a.py`, `q1b.py` and `q2.py`. Note that Markus has been setup to accept exactly those 3 files. All the necessary helper functions should be included in each file for each problem. You should not change any of the names or signatures of the functions already provided in the files, and you are not allowed to import any other package than the ones already imported.
- In this assignment, you should not solve the problems algorithmically. This means that your Python code should not do any type of search of the solution. It should only construct the encoding and call the solver to obtain a satisfying assignment.

Note that your assignment will be automatically graded. Your functions will be called from another script. If you mess with the signature, the call will fail and the autograder will give you a 0 mark.

The submission system will remain open for 12 hours after the deadline, but there is a penalty deduction formula set in Markus that deducts 4% for every hour of late submission up to 12 hours.

Instructions about Z3 and Python

- You have to use the Python API of Z3 to produce the encoding and call the solver. All the necessary functionality has been presented in Tutorials 2 and 3.
- You are not allowed to import any other package than the ones already imported in your python files.

Word of Advice

The goal of this assignment is to design encodings for some puzzles. The interface to the solver is minimal and you will need to use only a few functions, so using the solver is not the difficult part of this assignment. The majority of your effort goes into devising a correct encoding of the problem.

Problem 1(a) (40 points)

You will solve the stable marriage problem using an SMT solver. The problem has been traditionally stated as follows [1].

> *Given n men and n women, where each person has ranked all members of the opposite sex in order of preference, marry the men and women together such that there are no two people of opposite sex who would both rather have each other than their current partners. When there are no such pairs of people, the set of marriages is deemed stable.*

[1] Yes, the problem statement is culturally outdated! It is a famous old computer science problem though, and known as it was originally formulated. See `https://en.wikipedia.org/wiki/Stable_marriage_problem`

We can formulate the general problem as follows. We have two sets A and B of same size, and to each element in each set we associate an ordering of the elements of the other set (the preferences). A matching between sets A and B is a set of disjoint pairs (x, y) $(x \in A, y \in B)$. Since A and B are of the same size, a matching associates an element of B to every element of A and vice-versa. A stable marriage is a matching between A and B such that there does not exist any two elements $x \in A$ and $y \in B$ such that x is not matched with y but x prefers y over their current matching and y prefers x over their current matching.

Input format The input is a text file where each line represents the list of preferences. The file has $2n$ lines, where n is the size of set A and of set B. The elements of A are indexed from 1 to n, and the elements of set B from $n+1$ to $2n$. Line i is a list of n integers separated by spaces representing the preferences of i in decreasing order.
For example, the following input defines a problem where $A = \{1, 2, 3\}$ and $B = \{4, 5, 6\}$ where 4 is the most preferred element of both 1 and 2.

```
4 5 6
4 6 5
5 6 4
1 2 3
2 3 1
3 2 1
```

The functions parsing the input are already part of the starter code in `q1a.py` and `q1b.py`.

Task

Encode the problem into a SMT problem with the theory of your choice. You are **not** allowed to use the optimization solver. You must complete the function `solve(A,B)` in `q1a.py` such that the function returns a stable matching, as a list of pairs of integers. `A` and `B` are the two sets where each integer element is accompanied with its list of preferences in decreasing order. To test your encoding, a few input examples have been provided in `test_inputs/`.
Remark that the code forces you to separate your constraints into two sets. First, a set of constraints `constraints_matching` that ensure your output is a matching, but not necessarily stable. Second, a set of constraints `constraints_stability` that ensure the matching is stable. You should not modify the piece of code that calls the solver in `solve(A,B)` nor the name of the lists of constraints. We will verify that you correctly separated the two sets of constraints.

Problem 1(b) (20 points)

In the previous version of the problem, you encoded the problem using clauses ensuring that the output matching is stable. Now your task is to solve the same problem again using a different encoding technique. You will replace the stability clauses with optimization objectives. You should think of an optimization objective that ensures the matching is stable. We will check your code to ensure that you do not reuse the stability constraints of part (a), which would get zero marks for this part.

Task

Complete the function `solve(A,B)` in `q1b.py` such that the function returns a stable matching as a list of pairs of integers. You have to use the optimization solver and add the optimization objectives ensuring the matching is stable.

Note that the starter code now forces you to separate the problem into a set of constraints `constraints_matching` that ensure your output is a matching, and a set of minimization objectives `minimization_objectives` that ensure the matching is stable. You are not allowed to change the names of these lists and the code that calls the solver. If you want to maximize Q, you can always add $-Q$ to the list of minimization objectives.

Problem 2 (40 points)

Hidato is a famous puzzle game. You may have already played this game on your phone. If not, the time has finally come for you to know about it:

`https://en.wikipedia.org/wiki/Hidato`

Input format The partially completed grid is provided to you with `-` standing for each blank cell to be filled with a number, and `*` for blocked cells (that are not to be filled with a number). Each symbol two symbols is separated by a single space. Each line of the puzzle is written in a separate line of the input file. For example, the input file:

```
* * 7 6 - * *
* * 5 - - * *
31 - - 4 - - 18
- 33 2 12 15 19 -
29 28 1 14 - - -
* * - 24 22 * *
* * 25 - - * *
```

Has the following solution, represented in the same format:

```
* * 7 6 9 * *
* * 5 8 10 * *
31 32 3 4 11 16 18
30 33 2 12 15 19 17
29 28 1 14 13 21 20
* * 27 24 22 * *
* * 25 26 23 * *
```

The function parsing the input grid is provided in `q2.py`.

Task

Use the theory of your choice to formulate a Hidato puzzle solver using Z3's Python interface. You will complete the function `solve(grid)` in `q2.py` such that the function answers:

Given a partially completed grid, is it possible to complete the grid obeying the rules of Hidato?

If the answer is no, then simply return `None`. If the answer is yes, then you need to return the completed grid.