CSC410

Assignment 2

Due on Monday October 16, 2017 before 11am

Assignment Format and Guidelines on Submission

For each problem, you will write a wrapper code, in your language of choice, such as Python, Perl, C, etc that reads an input text file (as specified for each problem) and generates the output file, and a formula file that contains the exact goal that you submit to z3 for checking (in the same directory where we execute your code). The formula file will serve as a side-sanity-check for us to make sure you are doing things properly. Your wrapper processes the input file, constructs the formula, discharges the satisfiability goal to z3, and processes the output provided by z3 to construct the (well-formatted) output for each problem (as specified in each case).

We will expect a command-line executable (on CDF/CSLab machines) file named:

- `hidato` (for problem 1)
- `grouping` (for problem 2)

We will test your answer on CDF/CSLab machines by placing all your files in a single directory and invoking (for example)

```
hidato input-file
```

from the command-line.

Summary of files you provide and the output files your tool should generate:

- Hidato
  - An executable called `hidato`. This should be executable on CDF (without a need for paths, ...). We will basically type `hidato sample-input-file-name`, and your program should run and produce the desired outputs.
  - All source files for this executable (for inspection purposes) as a zipped directory `hidato.zip`
  - As a result of running your executable on a text input file (as specified), we will expect two output files: `hidato-output.txt` which includes the correct answer (as specified), and `hidato-formula.smt2` which includes the formula you submit to z3 to solve the puzzle.
- Grouping
  - Executables called `groupinga` and `groupingb`. This should be executable on CDF (without a need for paths, ...). We will basically type `groupinga sample-input-file-name`, and your program should run and produce the desired outputs.
  - All source files for the executables (for inspection purposes) as a zipped directory `grouping.zip`
  - As a result of running your executable on a text input file (as specified), we will expect two output files: `groupinga-output.txt` which includes the correct answer (as specified), and `groupinga-formula.smt2` which includes the formula you submit to z3 to solve the puzzle (repeat for the `groupingb` instance).

Evaluation method: We will test each problem with a few different inputs (5–10). For each correct output, you will get an equal fraction of the problem points. In other words, if your code returns the correct output on all tests, you will get a full-mark, and for every wrong answer you will loose (proportionally) some points. Beyond this, **there are no partial marks for any partial efforts**.

Problem 1

[30 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:

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

Use *your theory of choice* to formulate a Hidato puzzle solver: "Given a partially completed grid, is it possible to complete the grid obeying the rules of Hidato?" If the answer is no, then a simple "NO SOLUTION" in the output is sufficient. If the answer is yes, then you need to provide the completed grid as an output.

Input format. The partially completed grid will be provided to you with "-" standing for each blank cell (to be filled by a number) and "*" standing for blocked cells (not to be filled by any numbers) each two symbols 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 - -  * *
```

represents the game (to be played):

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

is an input file for which your output should be:

```
* * 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  * *
```

that corresponds to the solved puzzle:

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

Do not forget: your wrapper code (in your programming language of choice) is meant to only read the text grid, and output a formula F (in the theory of your choice) where a satisfiable assignment to F is a solution to puzzle. You are not allowed to do any other "solving" in your programming language. All puzzle solving needs to be deferred to z3. If you do any solving in your wrapper code (even if it is a little bit), you will get a zero for this problem.

Problem 2

Consider the situation that the students in a class want to team up in groups of (exactly) 2 students for a course project. The instructor asks each student to submit a list of partners that they are willing to work with. These lists are not prioritized, in the sense that it is understood that a student would be equally happy to work with anyone on her/his list. After all lists are in, the instructor is faced with the challenge of making *as many groups of 2 as possible* respecting students' wishes. The goal of this problem is to write a small program to help "her".

Write a wrapper program that takes the lists of student preferences as an input, and prints in the output the groups (of two students) formed, with the understanding that as many students as possible have been grouped together. You will solve this problem in two ways:

(a) (30 points) You can only use the theory of propositional logic (boolean variables only) for the purposes of this subproblem. Solutions that use richer theories such as linear arithmetic or equality logic will get no credit.

(b) (30 points) You can only use a richer theory of your choice. Note that the theory does not have to be theoretically strictly more expressive; therefore, equality logic with uninterpreted functions (reducible to propositional logic) is acceptable for this.

Keep in mind that the encoding should be genuinely different in this case compared to (a), in the sense that you will not add superficial bells and whistles to (a) to get (b). Since this theory subsumes propositional logic, it is certainly possible to do the superficial thing. But, that solution will get zero credit. So, in this case think integer variables instead of booleans, and genuine non-boolean constraints. The purpose of the exercise is for you to contrast two different encodings of the same problem against each other, in terms of (i) the ease of encoding, (ii) the size of encoding, and (iii) the performance of the solver at the end. You will submit a pdf file (`grouping.pdf`) containing a short report to let us know about your conclusions regarding (i–iii). Make clear claims, and back them up by solid evidence. But, don't be verbose. There are 10 points dedicated to this report, which will bring the total for the homework to 100 points.

Do not forget, in each case, like problem 1, you cannot do any solving in the wrapper code. It should be solely for the purpose of generating and dumping a formula to z3, which will then produce the optimal solution (including the optimization part) for you. This is in the spirit of the job shop problem you saw in the tutorial. For (a), you will be using the `assert-soft` command (not covered in the tutorial), but explained in the online Z3 tutorial.

Input format. For simplicity, we identify students with integer identifiers $1 \ldots n$ (where n is the number of students in the class). The input will have one line per student in which a (space separated) list of id's appear indicating the student's choice. For example, in a class with 5 students, the following input:

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

illustrates the partner choices of student "1" as $\{2, 3, 4\}$, and the partner choices of student "5" as just a single student "2". The correct (maximal) output for this case then is:

```
2 groups formed:
1,4
2,5
```

or

```
2 groups formed:
1,3
2,5
```

It is understood that there is only one answer for the optimal *number* of groups that can be formed, but the arrangement of students into groups that would produce that optimal number may not be unique. The number should be correct, but any valid arrangement is fine. You do not need to produce all valid arrangements for the optimal answer.
Note: This is a computationally hard problem. It could be expected, depending on the quality of your encoding, that your code will not scale to large instances. That said, your code (even if correct) should not be so bad in quality that it cannot solve small instances like the example given above. So, like any other code, reasonable performance is part of the specification together with correctness. If a group's code substantially outperforms everyone else's (in solving large instances), then I will consider a bonus prize (in the form of extra marks) for that group.

Repeat Warning

The purpose of these exercises is to solve these problems using a reduction to a satisfiability problem. If you write a program that solves the same problem *algorithmically*, you will not get any credit for it. The reason that we ask for the formula file is to verify that you are indeed doing the reduction. The wrapper program should only process input, generate formulas, and clean up outputs. All the problem solving should be deferred (by way of a correct encoding) to Z3. You have now been warned numerous times in a short document about this single issue. There will be zero leniency regarding mistakes of this type.

Extra

If you are a genuinely interested student, I encourage you to try to separately program direct solutions for either of these problems and compare your solution to the wrapper+solver combination. It would be interesting to see if yours is better or worse, and how long each solution takes. This is just a suggestion for you and has no bearing on the homework or its mark.