CS 173 Discrete Structures
Spring 2016
Honors Homework 3
due Monday,May 2nd

As with the second honors homework, your answers must be formatted using the Latex document formatting package. (Not just the equation mode found in Piazza and Moodle.) Exception: you do not need to format supporting materials such as program source code, screenshots, and figures. Please submit only your Latex-formatted document, plus hard-copies of supporting materials such as program code, in the CS 173 honors dropbox in the basement of Siebel. (Latex source code is not required.)

In this set of exercises, we will build a (fast) Sudoku solver using *constraint solvers*, in particular SMT solvers

`(see http://en.wikipedia.org/wiki/Satisfiability_Modulo_Theories )`

We will use only the quantifier-free theory of linear arithmetic. In this logic, we are allowed to use variables ranging over integers, use integer constants, use the addition function $+$, normal arithmetic relations ($=$, $\leq$, $<$, etc.), and Boolean logic ($\wedge, \vee, \neg$, etc.) to build formulas.

For example, we can write the following formula:

$$x>1 \wedge (y<6 \vee x+y<5) \wedge y>7 \tag{1}$$

An SMT solver will take such a formula and solve the satisfiability problem for it (is there a valuation/interpretation of the integer variables such that the formula evaluates to true). Moreover, if the formula is satisfiable, then the solver will give you a valuation under which the formula is true.

For example, the above formula (1) is unsatisfiable.

We are going to use Z3, which is an SMT solver, to solve Sudoku puzzles. Given a Sudoku puzzle P, we want to generate a linear arithmetic formula φ such that a satisfiable valuation for φ will solve the puzzle. See `http://en.wikipedia.org/wiki/Sudoku` for rules on Sudoku puzzles if you are not familiar with them.

We want you to build this solver by developing a program (in any language that you wish) that:

- Takes a Sudoku puzzle (in the input format described below) and converts it to a quantifier-free linear arithmetic formula φ expressed in the syntax of Z3.
- Run Z3 on this formula, and obtain a satisfiable valuation.
- Output the solution to the Sudoku puzzle (format below)

Moreover, we want you to test your program using several Sudoku puzzles (draw them from somewhere, say from the internet).

Input/output format: The input format will be a file with 9 lines, each line 9 characters long, such as:

```
25..3.9.1
.1...4...
4.7...2.8
..52.....
....981..
.4...3...
...36..72
.7......3
9.3...6.4
```

This encodes a Sudoku puzzle, where some concrete elements of the grid are given, and the rest, which are unknown, are represented by ".".

Your output must be in a similar format:

```
258736941
619824357
437915268
395271486
762498135
841653729
184369572
576142893
923587614
```

with all grid points containing numbers that extend the input to meet the Sudoku constraints.

Z3 formula format: See `http://rise4fun.com/z3/tutorial` for a tutorial on Z3, and read the section on "Arithmetic". You don't need reals or non-linear arithmetic for this problem.

Encoding a formula into Z3 is simple: you must first declare the variables to be integers, and then assert the formula as a bracketed expression (with operators being the first argument, and the arguments to the operator following it, in LISP-like notation). You finally give commands check-sat to check for satisfiability and the command get-model to find a satisfiable valuation. The tutorial comes with an online version of Z3 that you can play with to understand the syntax.

For example, the sample formula (1) on the previous page can be written in Z3 as:

```
(declare-fun x () Int)
(declare-fun y () Int)
(assert (> x 1))
(assert (or (< y 6) (< (+ x y) 5)))
(assert (> y 7))
(check-sat)
(get-model)
```

You can feed the above to Z3 (say online on `http://rise4fun.com/z3/` and Z3 will return `unsat`. If given a satisfiable formula, Z3 will return a model that is fairly easy to understand (for each variable, it will give you a valuation).

In addition to the commands shown in the Z3 tutorial, there is also a feature for declaring that a group of variables must have distinct values:

```
(assert (distinct x y z))
```

Encoding: Before you encode the problem and write the program, answer the following questions. The answers to these questions should be submitted in addition to the program.

1. Let us forget for moment the particular Sudoku problem we have been given and try to model the constraints common to any Sudoku problem. Or, if you like, a Sudoku puzzle where all of the squares have been left blank.

   Let us model a generic Sudoku solution as a set of 81 integer variables, one variable for each grid-position, where we expect the variable to range over the domain $\{1, ...9\}$ encoding the number at the grid-position corresponding to the variable. Give a precise description of the constraints that a correct solution must satisfy and explain why it is correct.

   Each of your basic constraints should be a logical formula whose variables are the 81 grid-position variables. To describe a set of similar constraints, use either logical connectives or precise mathematical English to show how to enumerate the constraints in the set.

   Figuring out how to answer this question succinctly will help you write a short and straightforward program.

2. To encode a specific Sudoku puzzle, you need to add another group of constraints to the ones in Part 1. Explain what these are.

3. Describe how you could check whether a given Sudoku puzzle has precisely one solution. You can make multiple calls to the SMT solver. Describe your solution clearly, being precise about what queries you will make to the solver. You do not have to implement this part.

Code design: Your Sudoku solver will have three steps:

1. Read a Sudoku puzzle in the file format described above and convert it to a formula suitable for giving to Z3.
2. Call Z3 on that formula.
3. Convert Z3's output into our output format.

To perform the second step, you can cut-and-paste formulas to and from the online Z3 solver. In that case the first and third steps might be separate programs. Alternatively, you can download Z3 to your machine and write one program that uses the Z3 API. Both approaches are fine.

Deliverables: The material that you need to submit are:

- The answers to the above questions in LaTeX format, explaining clearly the encoding.
- Your source code.
- The solution to the Soduku puzzle contained in `puzzle2.txt` (downloaded from the honors assignment page).
- Two other Sudoku puzzles and the solutions produced by your program. You can find some puzzles here: `http://magictour.free.fr/sudoku.htm`.
- For one of the example puzzles, give the Z3 input and output formulas (i.e. the input and output of step 2).

Why this exercise? This exercise is meant to teach you to use logic in a programmable fashion. Furthermore, SAT/SMT solvers are powerful tools that can solve theoretically hard problems reasonably fast in practice, at least in some domains. This exercise will teach you how to use these solvers.

Finally, if you are interested, you could implement the above solver using *bit-vectors* (see Z3 documentation), and get a faster implementation (this is not mandatory, of course).