CS 173 Discrete Structures
Spring 2014
Honors Homework 2
due May 5, 2014

Honors homework must be formatted using the LaTex document formatting package. (Not just the equation mode found in Piazza and Moodle.) See the CS 173 honors web page for help getting started with LaTex. Exception: you do not need to format supporting materials such as program source code, screenshots, and figgures. Your homework should be submitted as hardcopy in the CS 173 honors dropbox in the basement of Siebel. Please submit a hardcopy of your LaTex document both source code and formatted output and hardcopies of any supporting materials. The dropboxes are located just east of the lounge area with the big windows. If you get to the candy/soda machines, youve gone too far east.

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).

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 given. 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.

   Describe the Sudoku constraints logically as a formula over these variables. Argue why it is correct.

   You may want to describe a *minimal* set of constraints (this will help you program it easier as well). If you do this, argue why the set of constraints you choose capture the Sudoku constraints.

2. Formally describe how you will add constraints to the formula in Part 1 to capture a particular Sudoku puzzle instance.

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

Deliverables: The material that you need to submit are:

- The answers to the above questions in LaTeX format, explaining clearly the encoding.
- The source code of the program that converts a Sudoku puzzle to a formula. And the code that converts Z3's output to a solution in the output format. (Once you convert it to a formula, you can use Z3 to solve it. You can use either a downloaded version of Z3 or the online one at `http://rise4fun.com/z3/`. You don't have to have the program call Z3 using Z3's API; writing the formula to a file and calling Z3 on it is good enough. Note that you can copy-paste into and out of the online Z3 version.)

- The solution to the Soduku puzzle contained in `puzzle2.txt` (downloaded from the honors assignment page).
- A set of 2 other Sudoku examples on which your program produces the correct answers. You need to show the 2 Sudoku puzzles and their solutions, and for one of them, give the Z3 input and output. You can find some puzzles here: `http://http://magictour.free.fr/sudoku.htm`.

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).