ITP 20002-03 Discrete Mathematics, Fall 2021

Homework 2. Solving Puzzles with SAT Solver

Shin Hong

hongshin@handong.edu

1. Introduction

This homework asks you to construct C programs that use a SAT solver (e.g., Z3) to find solutions of given problems of three puzzle games *Anti-King Sudoku*, *Nondango*, and *Gappy*. Specifically, your program for each puzzle game must generate a propositional formula that represents the winning condition of a given game instance, passes the formula to a SAT solver and then translate the result from the SAT solver into a human-readable game solution. For each puzzle game, you must submit a C program source code file and a report that details your design and implementation, as well as the demonstration of its soundness.

This homework consists of (1) collaborative work and (2) individual work. The collaborative part asks you to work together with your partner on *Anti-king Sudoku* and *Nondango* and submit the results by **October 7** (Thur). At the same time, you are asked to work individually on *Grappy* and submit the result by **October 11** (Mon).

You can find the detailed descriptions of *Anti-king Sudoku*, *Nondango* and *Gappy* at Sections 2, 3 and 4, respectively, including puzzle rules, and input and output data format. Section 5 specifies the requirements and the guidelines on your program structures and your reports. Section 6 gives submission instructions.

2. Anti-king Sudoku

2.1. Rules

Anti-king Sudoku is a variant of Sudoku which inherits all settings (e.g., 9-by-9 grid with nine 3-by-3 sub-grids, pre-assigned cells) and rules from the original version and has a new requirement that the number on a cell is different from all numbers of the adjacent cells in the scope of King's movement. Specifically, the rules of *Anti-king Sudoku* are as follows:

- Each cell can be assigned with one integer between one and nine such that each of the nine integers must appear exactly once in every row, every column, and every sub-grid.
- A player cannot change the numbers on the cell initially assigned by the problem.
- No two cells adjacent to each other vertically, horizontally, or diagonally have the same number.

A player wins a puzzle if he/she finds an assignment to all cells that satisfy all requirements at the same time. Figure 1 is an example of a puzzle problem (left) and one of its solutions (right) [1].

					5		6	
			4	2				1
	7			6			3	
				3				2
	2						9	
6				7				
	4			1			8	
8				4	7			
	1		9					

4	3	1	7	9	5	2	6	8
9	6	8	4	2	3	7	5	1
5	7	2	1	6	8	9	3	4
1	9	5	8	3	4	6	7	2
7	2	4	6	5	1	8	9	3
6	8	3	2	7	9	4	1	5
2	4	9	5	1	6	3	8	7
8	5	6	3	4	7	1	2	9
3	1	7	9	8	2	5	4	6

Figure 1. An example problem (left) and solution (right) of the Anti-king Sudoku puzzle

2.2. Input and output

An input consists of nine lines each of which has nine values separated by one whitespace. A value can be one of the nine integers (from '1' to '9') representing preassigned values and '?' indicating an unassigned cell. The following is an input text for the puzzle instance of Figure 1 (visualize the control character as blue):

```
? ? ? ? ? 5 ? 6 ?
? ? ? 4 2 ? ? ? 1
? 7 ? ? 6 ? ? 3 ?
? ? ? ? 3 ? ? ? 2
? 2 ? ? ? ? ? ? 9 ?
6 ? ? ? 7 ? ? ? ? ?
? 4 ? ? 1 ? ? 8 ?
8 ? ? ? 4 7 ? ? ?
? 1 ? 9 ? ? ? ? ? ?
```

Like the input data, a solution must be represented as nine lines of nine values where all question marks are replaced with one of the nine integer values. For example, the solution of Figure 1 is as follows:

```
4 3 1 7 9 5 2 6 8
9 6 8 4 2 3 7 5 1
5 7 2 1 6 8 9 3 4
1 9 5 8 3 4 6 7 2
7 2 4 6 5 1 8 9 3
6 8 3 2 7 9 4 1 5
2 4 9 5 1 6 3 8 7
8 5 6 3 4 7 1 2 9
3 1 ? 9 8 2 5 4 6
```

3. Nondango

3.1. Rules

Nondango is a board-based puzzle with 10-by-10 grid divided into 13 to 50 regions, where each region consists of two to eight connected cells. Some cells in a grid contains

[1] http://www.cross-plus-a.com/sudoku.htm#Anti-King%20Sudoku

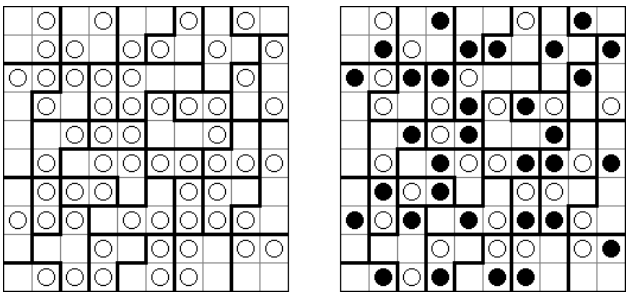

Figure 2. An example problem (left) and solution (right) of the Nondango puzzle

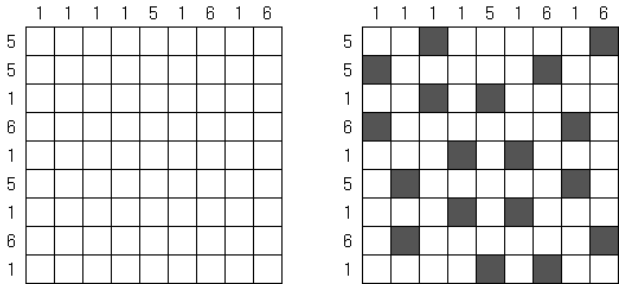

Figure 3. An example problem (left) and solution (right) of the Gappy puzzle

circles, and initially all circles are colored in white. The player of this puzzle is asked to change the color of a circle into black to satisfy all following rules:

- each region must contain exactly one Black circle, and
- there must be no three circles of the same color placing on three consecutive cells (including the cases of three consecutive cells across region boundaries).

Figure 2 shows an example of a Nondango puzzle problem (left side) and one of its solutions (right side). Note that there may be multiple solutions for a puzzle[2].

3.2. Input and output

An input data is given as a text file with ten lines. Each line contains ten words separated by a whitespace. The *j*-th word at the *i*-th line represents the cell at the i-th row (from the top) and the j-th column in the grid. A word contains a positive integer indicating the region identifier and optionally 'W' if a circle is put on the cell. The following is the text file representing the input data of Figure 2:

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

The solution is displayed as text of nine lines indicating each cell contains nothing ('X'), or a white circle ('W'), or black circle ('B'). Two characters in the same line must be separated by a whitespace. The following is the text for the solution of Figure 2:

```
X W X B X X W X B X
X B W X B B X B X B
B W B B W X X X B X
X W X W B W B W X W
X X B W B X X B X X
X W X B W W B B W B
X B W B X X W W X X
B W B X B W B B W X
X X X W X W W X W B
X B W B X B B X X X
```

4. Gappy

4.1. Rules

Gappy is a puzzle for a player to color each cell of a 9-by-9 grid as white or black while following all posed rules. A puzzle problem is defined by the labels given for each row and column. A label is an integer number between 0 and 7. A player wins a game when each cell is properly colored according to the following rules:

- no two black cells are adjacent to each other vertically, horizontally, or diagonally, and
- there must be exactly two black cells in each row, and
- there must be exactly two black cells in each column, and
- the label on each row indicates the number of the white cells between two black cells in the row, and
- the label on each column indicates the number of the white cells between two black cells in the column.

Initially, no color is assigned to any cell in the grid. Note that there may be multiple solutions for the same problem. Figure 3 is the example of a puzzle problem and its solution[3].

4.2. Input and output

An input data is given as a text file with two lines. The first line contains the nine numbers of the row-wise labels, from the top-most one to the bottom-most one. The second line contains the nine numbers of the column-wise labels, from the left-most one to the right-most one. Two numbers in a line are separated by a white space. The following is the input data file of Figure 3:

```
5 5 1 6 1 5 1 8 1
1 1 1 1 5 1 6 1 6
```

A solution must be generated as a text data of nine lines each of which contains nine characters from 'B' and 'W' indicating whether the corresponding cell is colored in black ('B') or white ('W'). Each character in the same line must be separated by a whitespace. For example, the solution shown in Figure 3 must be represented as follows:

[2] http://www.cross-plus-a.com/puzzles.htm#Nondango

[3] http://www.cross-plus-a.com/puzzles.htm#Gappy

```
W W B W W W W W B
B W W W W W B W W
W W B W B W W W W
B W W W W W W B W
W W W B W B W W W
W B W W W W W B W
W W W B W B W W W
W B W W W W W W B
W W W W B W B W W
```

5. Requirements and Guidelines

You are asked to model a puzzle problem as a propositional logic formula such that an assignment that satisfies the formula represents a solution of the puzzle problem (the formula is unsatisfiable if there is no solution). Based on this modelling, you must construct a program that uses a SAT solver Z3 to find a solution of a puzzle problem given as input. More specifically, for each of the three puzzles, you need to accomplish the following tasks:

Step 1. devise a scheme to construct a propositional logic formula for a given puzzle problem

Step 2. construct a puzzle solving program. The program must have two components, the formula constructor and the Z3 output interpreter

Step 3. test your program

Step 4. write a report on Steps 1 to 3

The following subsections give details of these four steps.

5.1. Puzzle modeling (Step 1)

You must devise a scheme for constructing a propositional logic formula from a given puzzle problem (note that it must be a purely propositional formula without quantification aspects). Your scheme must define what kinds of propositional variables should be introduced for modelling a puzzle instance and how to translate all posed constraints into propositional formula over these propositional variables. Your report must explain this scheme in detail and show how the satisfiability of a translated formula is related with a puzzle solution.

5.2. Program construction (Step 2)

Based on your puzzle modeling, you should construct a C program that returns a solution for a received puzzle problem. Your program must have the following two components:

- Formula constructor

  Given puzzle problem instance, this module represents the corresponding constraints as a Z3 input string. The resulting string should be passed to Z3 by the main driver.

- Z3 output interpreter

  Given propositional constraints for a puzzle problem, Z3 returns a solution (either a satisfying assignment or a determination that the formula is unsatisfiable) as a formatted text. This module translates this Z3 output into a puzzle solution.

For each puzzle, you must create a program as a single C source code file and a build script (e.g., Makefile). Your program must receive the text data of a puzzle problem via the standard input, and then display a solution via the standard output. If there is no feasible solution, your program must print out "`No solution`". You can assume that always valid input is given to your program.

For evaluation, your program will be be built and tested on the peace server. It is highly recommended to check there is no problem in building and running your program on the peace server before submission.

5.3. Testing (Step 3)

You must present in your report the process and the result of how you have confirmed that your program is working correctly. You must show how you test your programs with different input cases.

5.4. Step 4. Report writing

You must write two reports, one on the collaborative work (*Anti-king Sudoku* and *Nondango*) and the other on the individual work (*Gappy*). Your writing must include the following aspects of your solution in detail:

(1) your idea and details on modeling a puzzle with a propositional logic

(2) structure of your puzzle-solving program, and challenges in implementation

(3) test results

(4) open discussion, for instances, lessons learned, observations, further investigation, suggestions of possible extension or improvements.

Submissions will be evaluated primary based on their reports (not implementations). Thus, your report is expected to deliver all results with best presentations.

You must use a given template to write a report. Each report must not exceed 4 pages. A report must be submitted as a PDF file (not a source file such as MS word file) to avoid formatting errors and other compatibility issues.

6. Submission Instructions

You must submit the results of the collaborative part and the individual part separately.

(1) collaborative work: upload a Zip file archiving all results on *Anti-king Sudoku* and *Nondango* (e.g., two source code files, build scripts, and a PDF file of your report) to Hisnet (not HDLMS). The submission deadline is **11:59 PM, Oct 7 (Thur)**. One submission per team is sufficient.

(2) individual work: upload a Zip file archiving all results on Gappy (e.g., a source code file, a build script and a PDF file of your report) to Hisnet (not HDLMS). The submission deadline is **11:59 PM, Oct 11 (Mon)**.

The two deadlines are strict. No late submissions will be accepted.