21700613 임성빈 21800325 박하윤
21800386 손세민 21800409 신지영
21800607 이혜림

Discrete Mathematics Section #1
Programming Assignment #1

< Puzzle Solvers Using SMT-solver >

1. Introduction

SMT-solvers are tools to verify and prove the correctness of programs. Z3, which is one of SMT-solvers, is a theorem prover developed by Microsoft Research. The program is used to check the satisfiability of logical formulas. It is composed of a sequence of commands. In this assignment, a logical formula with Z3 commands is built by using the C programming language. The result text file will be the input (formula) for Z3 which would generate a possible answer for each puzzle as an output.

2. Problem Definition

According to the given instructions, the overall formula must be written in quantifier-free LIA logic. Quantifier-free logic is a logic made with no quantifiers. Since these puzzles have natural numbers as their domain, the law of identity could be applied like the formula below.

$$\forall x[(x+1)(x-1) = x^2 - 1]$$

Then, the formula without the quantifier is always satisfied.

LIA stands for Linear Integer Arithmetic, which is a theory that allows inequalities and equations over real numbers. Therefore, quantifier-free LIA logic is simply a logic that uses formulas with equations and inequalities, but without quantifiers.

Another requirement for our program is to receive the input through standard input. A standard input in Linux (and other Unix-based OS) is the source of data input for command line programs. A standard input is by default plain text entered through the keyboard, but it can also be redirected to come from other sources through arguments and redirectional operators (e.g. '<'). In our program, the redirectional operator, '<', will be used to redirect the standard input from the keyboard to a text file.

3. Solutions for Three Puzzles

A) Puzzle 1 : Asterisk Sudoku

The sudoku is represented as a 9x9 grid with nine 3x3 subgrids.

The constraints with logic formulas of the asterisk sudoku is demonstrated as follows:

p(i,j,k) is equal to 1 if and only if a number k is placed at a(i,j) cell; 0 otherwise

i) All values must be from 1 to 9.

$$Q_1 : \bigwedge_{y=1}^{9} \bigwedge_{x=1}^{9} 1 \le a_{y,x} \le 9$$

ii) Each row must contain every number between 1 and 9.

$$Q_2 : \bigwedge_{k=1}^{9} \bigwedge_{y=1}^{9} \left\{ \sum_{x=1}^{9} p(y,x,k) = 1 \right\}$$

iii) Each column must contain every number between 1 and 9.

$$Q_3 : \bigwedge_{k=1}^{9} \bigwedge_{x=1}^{9} \left\{ \sum_{y=1}^{9} p(y,x,k) = 1 \right\}$$

iv) Each sub-grid must contain every number between 1 and 9.

$$Q_4 : \bigwedge_{m=1}^{3} \bigwedge_{n=1}^{3} \bigwedge_{k=1}^{9} \left\{ \sum_{y=3m-2}^{3m} \sum_{x=3n-2}^{3n} p(y,x,k) = 1 \right\}$$

v) Each cell with an asterisk sign must have distinctive values.

$$Q_5 : \bigwedge_{k=1}^{9} \left\{ \sum_{y=1}^{9} \sum_{x=1}^{9} p(y*, x*, k) = 1 \right\}$$

With these above constraints and logic formulas, the programming is approached as follows:

i) Declare constants as a(i,j) when it represents a value in a row i and column j.
ii) Assign the input into the specific cells with 'scanf" function.
iii) Set the values to be from 1 to 9, which is a constraint 1.
iv) Every row, column, and cell with asterisk sign is asserted to have a distinctive values by exploiting either sum of 'ite'(if then else) or 'distinct' command.

A discussion drawn by the asterisk sudoku is as follows:

- Setting values from 1 to 9 was prone to be ignored since it is common sense. Finding all the constraints takes a significant role in solving the problem.
- Initially thought that by asserting each row and column sum to be 45(sum from 1-9), the puzzle could be solved. However, test results showed that those constraints were not enough, and realized that setting the constraints would be a tricky job if the *distinct* keyword was not used.
- It was also difficult to describe in the z3 logic syntax that there are distinct values in each row, column, and sub-grid. we tried to follow the example of n_queen at first, but we noticed that we used the propositional logic and it needs too many constraints. After that, we started to seek a way to use LIA logic, not a propositional one, and finally found a ite(if then else) statement. Then the logic was completed by combining an example of n_queen_lia with sum of ite.
- Actually, we tried to use the command *(declare-sort..)* that make random type to replace Quantifiers. Although, at first, we were successful in implementing another logic different from the logic we adapted Solution 3, unfortunately, it takes too many times. Simply, I explain the logic we thought. Assume the random type is "Point" in Sudoku, then there are 81 Point in the board. If there are p1, p2, p3 as Point and both p1 and p2 are on the same row and both p2 and p3 on the same row, then, both p1 and p3 are on the same row. In the same way, we adapted the logic on column, block.

B) Puzzle 2 : Fill-a-Pix

Fill-a-pix is a puzzle that has similarity with minesweeper. A N x M matrix containing either integers from 1-9 or a '?' character is given. The number on a cell corresponds to the number of neighboring cells (including the cell containing the number) that are to be filled with black. The solution could be represented through standard output by setting empty cells as '0' and filled cells as '1'.

If the individual cells of the input and output matrix is denoted as input(i,j) and a(i,j) respectively, which both are N x M

matrices. The constraints of the puzzle is as follows:

a(i,j) is equal to 1 if and only if cell (i,j) is filled with black; 0 otherwise.

input(i,j) is the matrix that each cell is filled with given clue.(0-9)

i) Each a(i,j) must have either 0 or 1 as its value.

$$\bigwedge_{y=1}^{N}\bigwedge_{x=1}^{M} 0 \le a_{y,x} \le 1$$

ii) The sum of a(i,j) and its 3 - 9 neighboring cells (depending on their location) must be equal to the corresponding value in input(i,j).

$$\bigwedge_{y=1}^{N}\bigwedge_{x=1}^{M}\left\{\sum_{m=-1}^{1}\sum_{n=-1}^{1} a_{y+m,x+n} = input_{y,x}\right\}$$

How do we get 5 solutions at most?

We designed two ways to do this: one uses Z3 many times, another uses Z3 only once.

i) *In p2.c*, get a solution, and save the solution into a data structure. When getting the next solution, add a constraint that the next solution must not be the same with the previous solution every time.

ex. if solution 1 is :

a(1,1) is 1, a(1,2) is 0, a(1,3) is 1 … and a(n,m) is 0

Then add constraint to get *solution 2*:

Not { b(1,1) is equal to 1, b(1,2) is equal to 0, b(1,3) is equal to 1 … and b(n,m) is equal to 0 }

Same logic is applied every time to get the next solution. We applied it in the real project.

ii) Build Z3 script to get multiple solutions, and run Z3 once.

Solution 1: *a(1,1) , a(1,2), ..., a(n,m)*

Solution 2: *b(1,1), b(1,2), ..., b(n,m)*

Solution 3: *c(1,1), c(1,2), ..., c(n,m)*

Solution 4: *d(1,1), d(1,2), ..., d(n,m)*

Solution 5: *e(1,1), e(1,2), ..., e(n,m)*

Get solution 1: *a(1,1) , a(1,2), ..., a(n,m)*

add new constraint

$$\left(assert\ not \cdot \left(\bigwedge_{i=1}^{N}\bigwedge_{j=1}^{M} (= a(i,j)\ b(i,j))\right)\right.$$

Get solution 2: *b(1,1) , b(1,2), ..., b(n,m)*

add new constraint

$$\left(assert\ not \cdot \left(\bigwedge_{i=1}^{N}\bigwedge_{j=1}^{M} (= a(i,j)\ c(i,j))\right)\right.$$

$$\left(assert\ not \cdot \left(\bigwedge_{i=1}^{N}\bigwedge_{j=1}^{M} (= b(i,j)\ c(i,j))\right)\right.$$

Get solution 3: *c(1,1), c(1,2), ..., c(n,m)*

...

These are the designed logics to obtain multiple solutions in Z3.

With these above constraints and logic formulas, the programming is approached as follows:

i) Get the input matrix from standard input using ‘scanf’ function and store them in a 2-d array, a.

ii) Write the formula with the constraints of the puzzle following the Z3 syntax.

iii) Send the formula file to the Z3 solver using popen().

iv) Parse the output of the Z3 and store them into another 2d array, board.

v) Display the output matrix onto standard output using printf().

A discussion drawn by the Fill-a-Pix is as follows:

- How to identify input N x M?

we had trouble when receive standard input because though the input of *sudoku** must be 9 x 9, the input of *Fill-a-Pix* can be *N* x *M*. We designed two solutions.

i) get N, and M

a) Count N: the number of enter ('\n').
b) Count T: total number of integers.
c) M = T / N

ii) get M, and N

d) Count M: the number of integers before first enter ('\n').
e) Count T: total number of integers.
f) N = T / M

- In the process of writing instructions using y+m and x+n, we realized that some values were outside the given range. For example, when y is 0 and m is -1, y+m is -1 which is less than 0. To solve this problem, we used if (0 < y+m <= N and 0 < x+n <= M), which a statement means "print if the values are in given range in the process of writing formula."

C) Puzzle 3 : Numbrix

Numbrix puzzle consists of an NxM grid where some numbers between 1 and NxM are placed on some cells.

The constraints with logic formulas of the Numbrix is demonstrated as follows:

p(x,y,k) is equal to 1 if and only if a number k is placed at a(i,j) cell; 0 otherwise

i) Every value must be from 1 to N x M

$$Q_1 : \bigwedge_{y=1}^{N} \bigwedge_{x=1}^{M} \{1 \le a_{y,x} \le N \cdot M\}$$

ii,iii) Two numbers x and x+1 must be placed at vertically or horizontally adjacent cells.

$$Q_2 : \bigwedge_{y=1}^{N} \bigwedge_{x=1}^{M} \{(a_{y,x} - 1 = a_{y-1,x}) \vee (a_{y,x} - 1 = a_{y+1,x}) \vee (a_{y,x} - 1 = a_{y,x-1}) \vee (a_{y,x} - 1 = a_{y,x+1})\}$$

$$Q_3 : \bigwedge_{y=1}^{N} \bigwedge_{x=1}^{M} \{(a_{y,x} + 1 = a_{y-1,x}) \vee (a_{y,x} + 1 = a_{y+1,x}) \vee (a_{y,x} + 1 = a_{y,x-1}) \vee (a_{y,x} + 1 = a_{y,x+1})\}$$

iv) Every row and column must have only one every value.

$$Q_4 : \bigwedge_{k=1}^{N \cdot M} \left\{ \sum_{y=1}^{N} \sum_{x=1}^{M} p(y, x, k) = 1 \right\}$$

With these above constraints and logic formulas, the programming is approached as follows:

i) Declare constants as a(y,x) when it represents a value in a row y and column x.
ii) Assign the input into the specific cells with 'scanf' function.
iii) Set the values to be from 1 to NxM, which is a constraint 1.
iv) Two numbers *x* and *x*+1 are placed at vertically or horizontally adjacent cells.

A discussion drawn by the Numbrix is as follows:

- The first thing to do for the puzzle was to find a rule. The key rule was that among four neighbors of a cell must contain two cells that could be

connected. Therefore, one of the four needs to be +1 while another one should be -1 compared to the value of cell.

- It was important, but easy to forget, to set up the margin around the checked areas. Since the values should be within the NxM metrics, cells at the edges contain only two neighbors. In order to exclude the values beyond the margin, writing the Z3 commands only and only if the margin is kept within the range.

4. Demonstration of Puzzles

In order to check the correctness of each program, screenshots of the execution of each puzzle are listed below:

A) Asterisk Sudoku

Satisfiable case	
Input	Output
? 2 ? 5 ? * ? 9 ? 8 ? ? 2 ? 3 ? ? 6 ? 3 ? ? 6 ? * 7 ? * ? ? ? * ? 6 ? ? 5 4 ? ? ? ? ? 1 9 ? ? 2 ? ? ? 7 ? ? ? 9 * ? 3 ? ? 8 ? 2 ? ? 8 ? 4 ? * 7 ? 1 ? 9 ? 7 ? 6 ?	4 2 6 5 7 1 3 9 8 8 5 7 2 9 3 1 4 6 1 3 9 4 6 8 2 7 5 9 7 1 3 8 5 6 2 4 5 4 3 7 2 6 8 1 9 6 8 2 1 4 9 7 5 3 7 9 4 6 3 2 5 8 1 2 6 5 8 1 4 9 3 7 3 1 8 9 5 7 4 6 2
Unsatisfiable case	
5 2 ? 5 ? * ? 9 ? 8 ? ? 2 ? 3 ? ? 6 ? 3 ? ? 6 ? * 7 ? * ? ? ? * ? 6 ? ? 5 4 ? ? ? ? ? 1 9 ? ? 2 ? ? ? 7 ? ? ? 9 * ? 3 ? ? 8 ? 2 ? ? 8 ? 4 ? * 7 ? 1 ? 9 ? 7 ? 6 ?	No Solution!

B) Fill-a-pix

Case of solutions ≥ 5	
Input	Output
? ? ? ? ? ? 9 ? ? ? ? 8 8 ? ? ? ? ? ? 4 4 ? 5 ? 2	1 1 1 0 0 1 1 1 1 0 1 1 1 1 1 1 1 0 1 0 1 1 1 1 0 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 1 1 1 0 1 0 1 1 1 1 0 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 0
Case of solutions < 5	
? ? ? ? 3 ? 9 ? ? ? ? 8 8 ? ? ? ? ? ? 4 4 ? 5 ? 2	1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 0

C) Numbrix

Input	Output
? ? ? ? ? ? ? ? 20 13 ? ? ? 26 ? ? 9 ? ? 25 ? ? 10 ? ? ? 23 36 ? ? ? ? ? ? ? ?	17 16 15 14 7 6 18 19 20 13 8 5 27 26 21 12 9 4 28 25 22 11 10 3 29 24 23 36 35 2 30 31 32 33 34 1
? ?	20 19 16 15 12 11 21 18 17 14 13 10 22 23 24 25 26 9 33 32 29 28 27 8 34 31 30 3 4 7 35 36 1 2 5 6