ITP 20002-03 Discrete Mathematics, Fall 2021

Homework 2

Lee Hyo Ju

22000603@handong.edu

1. Introduction

This second report describes the task of writing c codes that meet the rules of 'Happy' puzzle and finding answers using z3. It explains the rules of this puzzle, how I approached the logic to solve this problem, the results and what I could learn by doing assignments.

To put it simply, "Gappy" puzzle is a game that colors cells to satisfy conditions among 9-by-9 grids. At the start of the game, each row and column have the present numbers between 1 and 7. These numbers are the number of white cells that must exist between the black cells in each row and column. In other words, only a specified number of white cells can be colored between the two black cells, and there are only two black cells present in each row and column. In addition, cells that are directly attached horizontally, vertically, or diagonally cannot be painted black.

In "anti-sudoku," "nondango," and "nqeen," which were covered in the previous task, there were rules that only one cell with a specific value should exist. Since Gappy is a rule that two must exist, it had to be approached thinking about the difference from the previous puzzles.

2. Approach

In other words, Gappy's rules can be expressed in four as follows.

1. Each of the 9x9 cells must have a value of either 'W' or 'B'.

2. Black cells cannot be attached horizontally, vertically, or diagonally.

3. There must be only two test cells in each row and column.

4. There should be as many white cells as input data between black cells.

To satisfy numbers 3 and 4 at the same time, I considered {B1,W1,⋯,Wn,B2} as one set when a label is n, first black cell in one row is B1 and second black sell is B2, and approached as a rule that only one set exists in each row and column. Therefore, the propositional logic formulas to be described in this puzzle can be expressed in three types as follows.

P1. The cells have only one color.

$$\bigwedge_{row=1}^{9}\bigwedge_{col=1}^{9}(\neg p_{row,col,W} \wedge p_{row,col,B}) \vee (p_{row,col,W} \wedge \neg p_{row,col,B})$$

In P1, to paint only one color per cell, the pW and pB variables in one cell must always have different values. In that case, either '¬pW ∧ pB' or 'pW ∧ ¬pB' always returns true, so I wrote the entire logical expression using that the logical expression that binds them to 'or' always returns true.

P2. The black cells are not right next to the other.

In P2, the rule that two adjacent cells cannot be black was expressed using the and-result of those two cells is always false.

P2-1) Row

$$\bigwedge_{row=1}^{9}\bigwedge_{col=1}^{8}\neg(p_{row,col,B} \wedge p_{row,col+1,B})$$

By adding 1 to the col, computer can express a set of two attached cells that can come out of one row.

P2-2) Column

$$\bigwedge_{col=1}^{9}\bigwedge_{row=1}^{8}\neg(p_{row,col,B} \wedge p_{row+1,col,B})$$

By adding 1 to the row, computer can express a set of two attached cells that can come out of one column.

P2-3) Diagonally to the right bottom.

$$\bigwedge_{row=1}^{8}\bigwedge_{col=1}^{8}\neg(p_{row,col,B} \wedge p_{row+1,col+1,B})$$

By adding 1 to the row and the col, computer can express a set of two attached cells that can come out of lower right diagonal.

P2-4) Diagonally to the left bottom.

$$\bigwedge_{row=1}^{8}\bigwedge_{col=2}^{9}\neg\left(p_{row,col,B}\land p_{row+1,col-1,B}\right)$$

By adding 1 to the row and the col, computer can express a set of two attached cells that can come out of lower left diagonal.

P3. The set exists only one.

In P3, the data from each row and column is written 'term'. Looking at the set as a one variable, you can simply understand this logic like 'nqueen' logic, there is a queen in only one cell. '$p_{row,col,B}$' is first black cell that can be exist, and '$p_{row,col+term+1,B}$' is second black cell that is automatically determined according to the location of. he first cell. The white cells between the black cells were all expressed at the location of the first black cell, adding from 1 to term.

P3-1) Existence in row.

$$\bigwedge_{row=1}^{9}\bigvee_{col=1}^{9-term-1} p_{row,col,B} \land \bigwedge_{mid=1}^{term} p_{row,col+mid,W} \land p_{row,col+term+1,B}$$

P3-2) Uniqueness in row.

$$\bigwedge_{row=1}^{9}\bigwedge_{col=1}^{9-term-2}\bigwedge_{c=col+1}^{9-term-1}\neg\Big(\big(p_{row,col,B} \land \bigwedge_{mid=1}^{term} p_{row,col+mid,W} \land p_{row,col+term+1,B}\big) \land \big(p_{row,c,B} \land \bigwedge_{mid=1}^{term} p_{row,c+mid,W} \land p_{row,c+term+1,B}\big)\Big)$$

P3-3) Existence in column.

$$\bigwedge_{col=1}^{9}\bigvee_{row=1}^{9-term-1} p_{row,col,B} \land \bigwedge_{mid=1}^{term} p_{row+mid,col,W} \land p_{row+term+1,col,B}$$

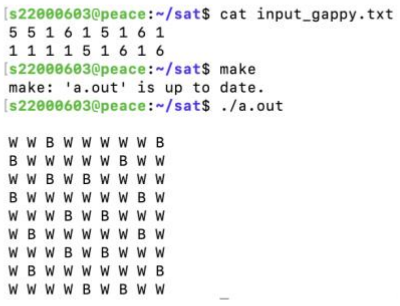
P3-4) Uniqueness in column.

$$\bigwedge_{col=1}^{9}\bigwedge_{row=1}^{9-term-2}\bigwedge_{r=row+1}^{9-term-1}\neg\Big(\big(p_{row,col,B} \land \bigwedge_{mid=1}^{term} p_{row+mid,col,W} \land p_{row+term+1,col,B}\big) \land \big(p_{r,col,B} \land \bigwedge_{mid=1}^{term} p_{r+mid,col,W} \land p_{r+term+1,col,B}\big)\Big)$$

3. Evaluation

When the same data as the input presented in the homework description is put in, the same W-B value is derived, or if all four rules of Gappy described in **Approach** are satisfied, the problem will be evaluated as successful. Also, it will be evaluated that it has successfully implemented Gappy-solver only when the correct output to meet all rules or by outputting No solution, in more than 3 another cases.

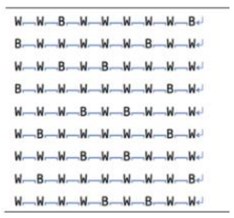
Suggested input)



```
s22000603@peace:~/sat$ cat input_gappy.txt
5 5 1 6 1 5 1 6 1
1 1 1 1 5 1 6 1 6
s22000603@peace:~/sat$ make
make: 'a.out' is up to date.
s22000603@peace:~/sat$ ./a.out

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
```

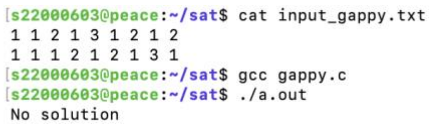
< The image of the suggested output >



```
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
```

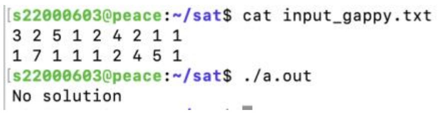
Comparing the two images, the same cells have the same value. Therefore, this code can be said to have solved the Gappy problem well.

Another input 1)



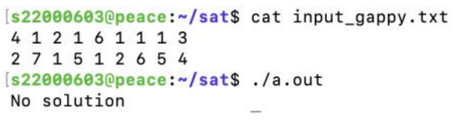
```
s22000603@peace:~/sat$ cat input_gappy.txt
1 1 2 1 3 1 2 1 2
1 1 1 2 1 2 1 3 1
s22000603@peace:~/sat$ gcc gappy.c
s22000603@peace:~/sat$ ./a.out
No solution
```

Another input 2)



```
s22000603@peace:~/sat$ cat input_gappy.txt
3 2 5 1 2 4 2 1 1
1 7 1 1 1 2 4 5 1
s22000603@peace:~/sat$ ./a.out
No solution
```

Another input 3)



```
s22000603@peace:~/sat$ cat input_gappy.txt
 4 1 2 1 6 1 1 1 3
 2 7 1 5 1 2 6 5 4
s22000603@peace:~/sat$ ./a.out
 No solution
```

It is difficult to find a case that is not a solution, so all three tasks derived a no solution, but this solution successfully solved the problem because the correct answer was derived for the presented input.

4. Discussion

The key point of this problem is to solve how to logically express the existence of only two black cells and the existence of a specified number of white cells between them. At first, I thought I should think of the two conditions separately and express all the two combinations of black cells that could be made in each row and column. Therefore, to define the white cells in the middle, the locations of the two black cells must be known, but it has become too complicated to set these locations as variables and store them in the code. When I thought of several cells as a set and expressed them, I was able to solve the problem much easier. Furthermore, expressing black and white cells as a set, code could be more efficient because it did not have to access all the black cell combinations that could come out of each row and column.

5. Conclusion

This task was to write a C language code that derives the correct answer to the Gappy puzzle using SAT solver. To accomplish this task, I first had to understand the rules of the game and create a logical circuit that would return only values that met the rules to true out of all the values that could come to the 9-by-9 grid cell. In addition, it was necessary to run z3-solver in C-language code and print the results derived in an easy-to-see form, and to establish a rule to check whether the correct answer was produced.

This report contains an explanation of how I performed all these processes and talks about how I approached to think about how effective logical circuits are at the heart of solving problems. Rather than looking at the rules one by one, it indicates that the key point of this task was to find a way to express them effectively while looking at the rules.

I tried to achieve more effective results in each process, and in conclusion, I was able to achieve successful results.