ITP 20002-03 Discrete Mathematics, Fall 2021

Homework 2

Your name : Seokjun Kim(김석준)

Email address : 21600081@handong.edu

Gappy

1. Introduction

Gappy is 9×9 puzzle that each cell either black or white according to the rules below.

- No two black cells are adjacent to each other vertically, horizontally, or diagonally.
- There must be exactly two black cells in each row and each column.

The label on each row and column indicates the number of white cells between two black cells on each row and column. Because of Gappy is 9-by-9 grid, a label must be between zero and seven. At the beginning of the puzzle, only labels exist, by which should infer the solution. Therefore, only information on label will be needed in the program. SAT solver will be used to solve this Gappy puzzle. A SAT solver evaluates if a propositional formula is satisfiable and, if it is, finds one of several answers. First, I have to declare propositional variables appropriately to suit the feature of puzzle, and to create a code that writes a propositional formula corresponding to the rules of the puzzle using the propositional variable declared. After that, I will interpret the instruction “z3 formula” to output the model or code it to inform that there is no solution.

2. Approach

Firstly, appropriate propositional variables should be declared to solve the Gappy puzzle. It is sufficient to include information about the index of the row and column, and whether it is a white cell or black in the propositional variables. Since it is 9-by-9 grid, i and j are declared from 1 to 9, and white and black are declared as 0 and 1, respectively. (If p110 true, then left-most and top-most one is white cell) Therefore, I will declare 9*9*2 variables.

```
// i: 1~9   j: 1~9    b: 0,1 (white, black)
for (i = 1 ; i <= 9 ; i++)
    for (j = 1 ; j <= 9 ; j++)
        for(b = 0 ; b <= 1 ; b++)
            fprintf(fp,"(declare-const p%d%d%d Bool)\n", i, j, b) ;
// example p110 p111
//          T    T  = contradiction
//          T    F  = White
//          F    T  = Black
//          F    F  = contradiction
```

1. Pre-assigned conditions (By a label)

There must be at least one pair of black cells in each row and column that match a label. Take the label input by the user or file, and write the propositional formula through that input.

```
scanf("%c", &buffer);
//puzzle input (row)
fprintf(fp,"; pre-assigned-row\n") ;
fprintf(fp,"(assert (and ") ;
for(int i = 1; i <= 9; i++) {
    scanf("%c", &label);
    scanf("%c", &buffer);
    space = (int)label - '0';
    if(space > 7 || space < 0){
        printf("This is not valid input.\n") ;
        return 0;
    }
    fprintf(fp, "(or ") ;
    for(int j = 1; j <= 8 - space ; j++){   // j + space + 1 <= 9 -> j <= 8 - space
        fprintf(fp,"(and p%d%d%d p%d%d%d)", i, j, 1, i, j+space+1, 1) ;
    }
    fprintf(fp, ")") ;
}
fprintf(fp,"))\n") ;

//puzzle input (column)
fprintf(fp,"; pre-assigned-column\n") ;
fprintf(fp,"(assert (and ") ;
for(int j = 1; j <= 9; j++) {
    scanf("%c", &label);
    scanf("%c", &buffer);
    space = (int)label - '0';
    if(space > 7 || space < 0){
        printf("This is not valid input.\n") ;
        return 0;
    }
    fprintf(fp, "(or ") ;
    for(int i = 1; i <= 8 - space ; i++){   // i + space + 1 <= 9 -> i <= 8 - space
        fprintf(fp,"(and p%d%d%d p%d%d%d)", i, j, 1, i+space+1, j, 1) ;
    }
    fprintf(fp, ")") ;
}
fprintf(fp,"))\n") ;
```

2. Each cell must be either a black or white.

To write a propositional formula that satisfies the rules of Gappy mentioned above, I will first make a rule to have either a white cell or a black cell in one cell. This logic is the same as the logic that either must be true and cannot be true at the same time. To satisfy this, I can establish a rule that both in the same row and column are not false or that neither can be true. It is expressed as a propositional logic as follows.

$$\bigwedge_{i=1}^{9}\bigwedge_{j=1}^{9}\{p(i,j,0) \vee p(i,j,1)\} \wedge \neg\{p(i,j,0) \wedge p(i,j,1)\}$$

```
// Each cell must be white or black, not both.  (and (or w b) (not (and w b)))
fprintf(fp,"; Q0\n") ;
fprintf(fp,"(assert (and ") ;
for (i = 1 ; i <= 9 ; i++) {
    fprintf(fp,"(and ") ;
    for (j = 1 ; j <= 9 ; j++){
        fprintf(fp,"(and (or p%d%d%d p%d%d%d) (not (and p%d%d%d p%d%d%d)))", i, j, 0, i, j, 1, i, j, 0, i, j, 1) ;
    }
    fprintf(fp,")") ;
}
fprintf(fp,"))\n") ;
```

3. There must be no two black cells are adjacent to each other.

To generate a code that satisfies this rule, four cases were divided into horizontal, vertical, and two diagonal directions.

3.1) horizontal

$$\bigwedge_{i=1}^{9}\bigwedge_{j=1}^{8}\neg\{p(i,j,1)\wedge p(i,j+1,1)\}$$

```
// There must be no two black cells are adjacent to each other
// horizontally
fprintf(fp,"; Q1-row\n") ;
fprintf(fp,"(assert (and ") ;
for (i = 1 ; i <= 9 ; i++) {
    fprintf(fp,"(and ") ;
    for (j = 1 ; j <= 8 ; j++){
        fprintf(fp,"(not (and p%d%d%d p%d%d%d))", i, j, 1, i, j+1, 1) ;
    }
    fprintf(fp,")") ;
}
fprintf(fp,"))\n") ;
```

3.2) vertical

$$\bigwedge_{i=1}^{8}\bigwedge_{j=1}^{9}\neg\{p(i,j,1)\wedge p(i+1,j,1)\}$$

```
// There must be no two black cells are adjacent to each other
// vertically
fprintf(fp,"; Q1-column\n") ;
fprintf(fp,"(assert (and ") ;
for (i = 1 ; i <= 8 ; i++) {
    fprintf(fp,"(and ") ;
    for (j = 1 ; j <= 9 ; j++){
        fprintf(fp,"(not (and p%d%d%d p%d%d%d))", i, j, 1, i+1, j, 1) ;
    }
    fprintf(fp,")") ;
}
fprintf(fp,"))\n") ;
```

3.3) diagonal (right)

$$\bigwedge_{i=1}^{8}\bigwedge_{j=1}^{8}\neg\{p(i,j,1)\wedge p(i+1,j+1,1)\}$$

```
// There must be no two black cells are adjacent to each other
// diagonally to the right
fprintf(fp,"; Q1-diag-right\n") ;
fprintf(fp,"(assert (and ") ;
for (i = 1 ; i <= 8 ; i++) {
    fprintf(fp,"(and ") ;
    for (j = 1 ; j <= 8 ; j++){
        fprintf(fp,"(not (and p%d%d%d p%d%d%d))", i, j, 1, i+1, j+1, 1) ;
    }
    fprintf(fp,")") ;
}
fprintf(fp,"))\n") ;
```

3.4) diagonal (left)

$$\bigwedge_{i=1}^{8}\bigwedge_{j=2}^{9}\neg p(i,j,1)\wedge p(i+1,j-1,1)$$

```
// There must be no two black cells are adjacent to each other
// diagonally to the left
fprintf(fp,"; Q1-diag-left\n") ;
fprintf(fp,"(assert (and ") ;
for (i = 1 ; i <= 8 ; i++) {
    fprintf(fp,"(and ") ;
    for (j = 2 ; j <= 9 ; j++){
        fprintf(fp,"(not (and p%d%d%d p%d%d%d))", i, j, 1, i+1, j-1, 1) ;
    }
    fprintf(fp,")") ;
}
fprintf(fp,"))\n") ;
```

4. There must be exactly two black cells in each row and column.

Since the above rule must have exactly two, it is difficult to write a propositional formula with one logic. Therefore, I would like to divide it into the following detailed rules that are easy to write.

4.1) Each row and column must have at least two black cells.

In fact, if you think carefully, 4.1) satisfies the condition through pre-assigned condition. In order for the solution to satisfy the pre-assigned condition, there must be at least one pair of black cells in each row and column that match a label. Therefore, we do not need to write a propositional formula for this condition.

4.2) Each row and column cannot have more than three black cells.

4.2.1) Each row cannot have more than three black cells.

A propositional formula can be written by confirming that there is no true case for all cases where each row can have three pairs of black cells.

$$\bigwedge_{i=1}^{9}\bigwedge_{j=1}^{7}\bigwedge_{jj=j+1}^{8}\bigwedge_{jjj=jj+1}^{9}\neg\big(p(i,j,1)\wedge p(i,jj,1)\wedge p(i,jjj,1)\big)$$

```
// There must be no more than three black cells in each row
fprintf(fp,"; Q2-row\n") ;
fprintf(fp,"(assert (and ") ;
for (i = 1 ; i <= 9 ; i++) {
    fprintf(fp,"(and ") ;
    for (j = 1 ; j <= 7 ; j++){
        fprintf(fp,"(and ") ;
        for(int jj = j+1; jj <= 8 ; jj++){
            fprintf(fp,"(and ") ;
            for(int jjj = jj+1; jjj <= 9; jjj++){
                fprintf(fp,"(not (and p%d%d%d p%d%d%d p%d%d%d))", i, j, 1, i, jj, 1, i, jjj, 1) ;
            }
            fprintf(fp,")") ;
        }
        fprintf(fp,")") ;
    }
    fprintf(fp,")") ;
}
fprintf(fp,"))\n") ;
```

4.2.2) Each column cannot have more than three black cells.

Same as the above except row and column.

$$\bigwedge_{j=1}^{9}\bigwedge_{i=1}^{7}\bigwedge_{ii=i+1}^{8}\bigwedge_{iii=ii+1}^{9}\neg\big(p(i,j,1)\wedge p(ii,j,1)\wedge p(iii,j,1)\big)$$

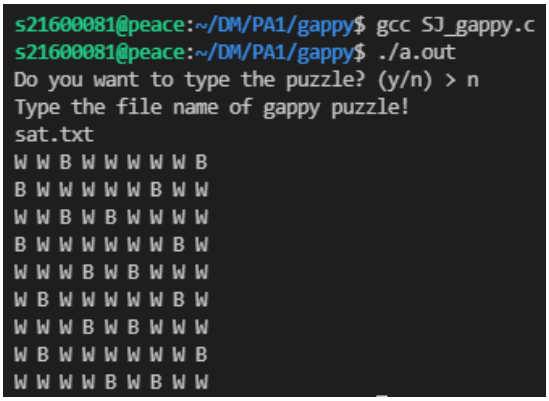
```
// There must be no more than three black cells in each column
fprintf(fp,"; Q2-column\n") ;
fprintf(fp,"(assert (and ") ;
for (j = 1 ; j <= 9 ; j++) {
    fprintf(fp,"(and ") ;
    for (i = 1 ; i <= 7 ; i++){
        fprintf(fp,"(and ") ;
        for(int ii = i+1; ii <= 8 ; ii++){
            fprintf(fp,"(and ") ;
            for(int iii = ii+1; iii <= 9; iii++){
                fprintf(fp,"(not (and p%d%d%d p%d%d%d p%d%d%d))", i, j, 1, ii, j, 1, iii, j, 1) ;
            }
            fprintf(fp,")") ;
        }
        fprintf(fp,")") ;
    }
    fprintf(fp,")") ;
}
fprintf(fp,"))\n") ;
```

3. Evaluation

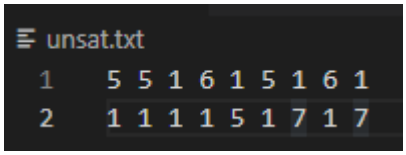
As we mentioned above, through Z3(SAT solver), if there is an answer to the Gappy puzzle, Z3 will present the model, and if not, the word 'unsat' will be output. If any number less than 0 or greater than 7 is included in the input of label, the phrase "This is not valid input" is printed, and if there is a solution, the model is printed, and if not, the phrase "This puzzle has no solution".

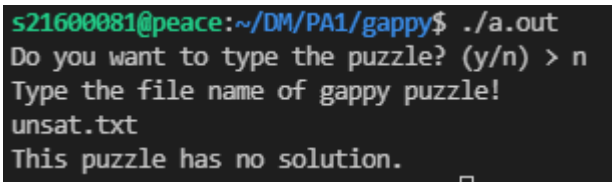
To confirm this, I tested three cases that we already know the correct answer, and do not have the correct answer, and one of the input of label is 8.

1) satisfiable

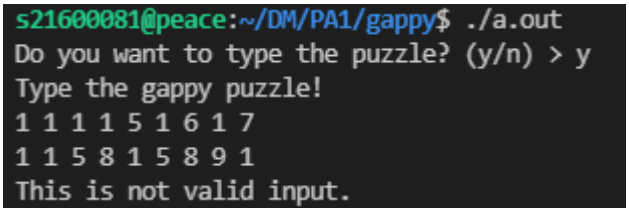


2) unsatisfiable





3) invalid input



Through the above, it can be seen that the program runs well without exception and an appropriate solution is output.

4. Discussion

What is interesting about solving Gappy puzzle is that even if p implies q is not a tautology, if only q is true when p is true, we do not have to write a logic q, just write p. From the above, let us set p and q as follows.

p : There is one pair of black cells in each row and column that match a label.

q : There are at least two black cells in each row and column.

If p is true, then q is true because every row and column have a pair of black cells. However, if p is not true, then we cannot have to say q is not true. This is because even if p is not true, pairs that do not match the label can exist in some rows and columns. Interestingly, however, we do not have a model anyway unless p is true in SAT solver, so the result does not change whether q is true or not. Therefore, we don not need to check q.

To put this broadly, we can see from this that if a true proposition leads to another true proposition, the latter case does not need to be considered when we use SAT solver.

5. Conclusion

We can quickly obtain the solution of some problems by creating a program using SAT solver like this program. How to declare propositional variables in programming was a very powerful aspect in the process of addressing the problem using SAT solver and investing a lot of that time is important for effective programming.

Through three puzzle homeworks, I was familiar with the ability to utilize SAT solver and the ability to solve problems through logic writing. I had considerable fun writing and programming logics so that the variables could be properly assigned when properly declaring and solving problems according to the characteristics of the problems.