ITP 20002-03 Discrete Mathematics, Fall 2021

Homework 2

Your name : Seokjun Kim(김석준), Junhyun Kim(김준현)

Email address : 21600081@handong.edu , 21800179@handong.edu

Anti-king Sudoku

1. Introduction

Anti-king Sudoku is a puzzle that adds one rule to the Sudoku puzzle. Here are the rules of the Sudoku.

- First, only one number can be written in each cell of the Sudoku.

- Second, all numbers in 1 to 9 should be entered one by one in each row and column without overlapping.

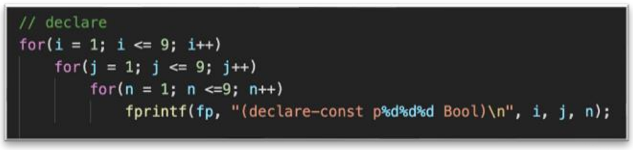
- Third, each cell in the sub-grid box has one of the numbers from 1 to 9 in each cell.

Anti-king Sudoku is a Sudoku with a rule that the same number should not fall within the range of movement of the king of chess around each cell. That is, here, a rule is added in which two cells close to each other vertically, horizontally, and diagonally cannot be the same number. However, it is possible to think a rule above such that two cells that are virtually diagonal close to each other cannot be the same number because single row, and single column cannot have the same number in the Sudoku. We will use SAT solver to solve this Anti-king Sudoku puzzle. SAT solver is a program that checks whether the propositional formula is satisfiable and finds one of solutions(model) for it if it is satisfiable. Using this, we declare that the numbers in each cell of the Sudoku puzzle are propositional variables (Boolean values). If we write logics that fit the rules of the Sudoku puzzle into a propositional formula, put this propositional formula into the Z3 solver input, and the model is the answer to the Sudoku puzzle if we get the satisfiable and model. We will have to declare an appropriate propositional variable so that we can reasonably check this rule, and we will have to write the logic for those propositional variables.

2. Approach

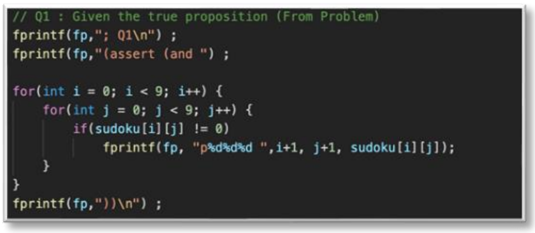
Here are the rules of the Anti-Sudoku. <Predicate declare> *P(i,j,n)* holds when row i and column j and number n.

: In fact, when we solve the Sudoku problem, the answer to each corresponding cell is from 1 to 9. However, a predicate is a propositional function, so we must express the solution as boolean value. For that reason, we declare "The cell in i-th row & j-th column, having n" as a predicate.



```
// declare
for(i = 1; i <= 9; i++)
    for(j = 1; j <= 9; j++)
        for(n = 1; n <=9; n++)
            fprintf(fp, "(declare-const p%d%d%d Bool)\n", i, j, n);
```

0 Given propositions.



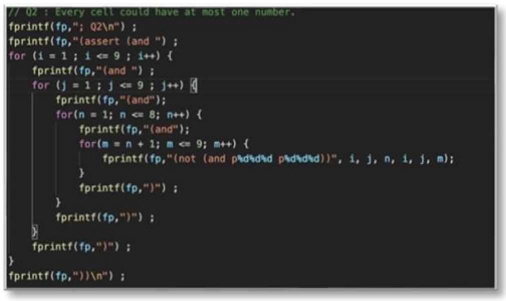
```
// Q1 : Given the true proposition (From Problem)
fprintf(fp,"; Q1\n") ;
fprintf(fp,"(assert (and ") ;

for(int i = 0; i < 9; i++) {
    for(int j = 0; j < 9; j++) {
        if(sudoku[i][j] != 0)
            fprintf(fp, "p%d%d%d ",i+1, j+1, sudoku[i][j]);
    }
}
fprintf(fp,"))\n") ;
```

: int sudoku[9][9]" has a state of given problem.

1 Each cell can be assigned with at most one number.

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

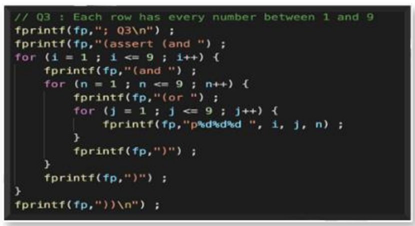


```
// Q2 : Every cell could have at most one number.
fprintf(fp,"; Q2\n") ;
fprintf(fp,"(assert (and ") ;
for (i = 1 ; i <= 9 ; i++) {
    fprintf(fp,"(and ") ;
    for (j = 1 ; j <= 9 ; j++) {
        fprintf(fp,"(and");
        for(n = 1; n <= 8; n++) {
            fprintf(fp,"(and");
            for(m = n + 1; m <= 9; m++) {
                fprintf(fp,"(not (and p%d%d%d p%d%d%d))", i, j, n, i, j, m);
            }
            fprintf(fp,")") ;
        }
        fprintf(fp,")") ;
    }
    fprintf(fp,")") ;
}
fprintf(fp,"))\n") ;
```

2 All numbers in 1 to 9 should be entered one by one in each row and column without overlapping.

- Each row has every number between 1 and 9

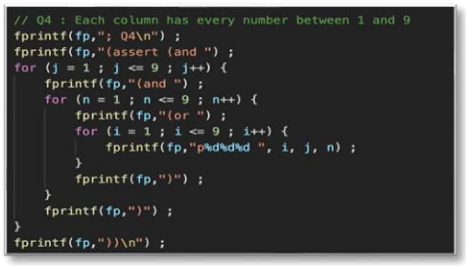
$$\bigwedge_{i=1}^{9}\bigwedge_{n=1}^{9}\bigvee_{j=1}^{9}p(i,j,n)$$



```
// Q3 : Each row has every number between 1 and 9
fprintf(fp,"; Q3\n") ;
fprintf(fp,"(assert (and ") ;
for (i = 1 ; i <= 9 ; i++) {
    fprintf(fp,"(and ") ;
    for (n = 1 ; n <= 9 ; n++) {
        fprintf(fp,"(or ") ;
        for (j = 1 ; j <= 9 ; j++) {
            fprintf(fp,"p%d%d%d ", i, j, n) ;
        }
        fprintf(fp,")") ;
    }
    fprintf(fp,")") ;
}
fprintf(fp,"))\n") ;
```

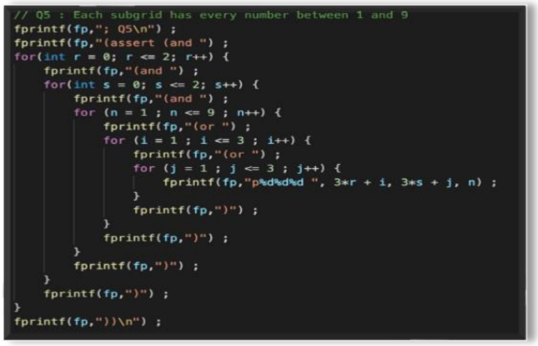
- Each column has every number between 1 and 9

$$\bigwedge_{j=1}^{9}\bigwedge_{n=1}^{9}\bigvee_{i=1}^{9}p(i,j,n)$$


```c
// Q4 : Each column has every number between 1 and 9
fprintf(fp,"; Q4\n") ;
fprintf(fp,"(assert (and ") ;
for (j = 1 ; j <= 9 ; j++) {
    fprintf(fp,"(and ") ;
    for (n = 1 ; n <= 9 ; n++) {
        fprintf(fp,"(or ") ;
        for (i = 1 ; i <= 9 ; i++) {
            fprintf(fp,"p%d%d%d ", i, j, n) ;
        }
        fprintf(fp,")") ;
    }
    fprintf(fp,")") ;
}
fprintf(fp,"))\n") ;
```

3 Each cell in the sub-grid box has one of the numbers from 1 to 9 in each cell.

$$\bigwedge_{r=0}^{2}\bigwedge_{s=0}^{2}\bigwedge_{n=1}^{9}\bigvee_{i=1}^{3}\bigvee_{j=1}^{3}p(3r+i,3s+j,n)$$


```c
// Q5 : Each subgrid has every number between 1 and 9
fprintf(fp,"; Q5\n") ;
fprintf(fp,"(assert (and ") ;
for(int r = 0; r <= 2; r++) {
    fprintf(fp,"(and ") ;
    for(int s = 0; s <= 2; s++) {
        fprintf(fp,"(and ") ;
        for (n = 1 ; n <= 9 ; n++) {
            fprintf(fp,"(or ") ;
            for (i = 1 ; i <= 3 ; i++) {
                fprintf(fp,"(or ") ;
                for (j = 1 ; j <= 3 ; j++) {
                    fprintf(fp,"p%d%d%d ", 3*r + i, 3*s + j, n) ;
                }
                fprintf(fp,")") ;
            }
            fprintf(fp,")") ;
        }
        fprintf(fp,")") ;
    }
    fprintf(fp,")") ;
}
fprintf(fp,"))\n") ;
```

4 No two cells adjacent to each other vertically, horizontally, or diagonally have the same number.

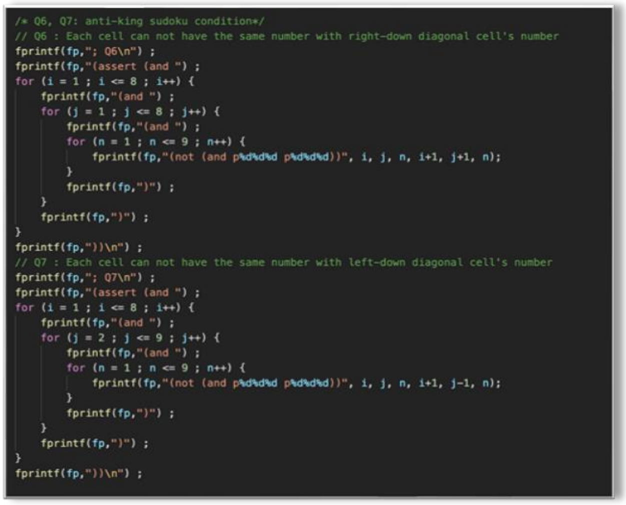
- vertically, horizontally

  : According to the 2nd rule, two cells adjacent to each other vertically or horizontally cannot have the same number.

- diagonally

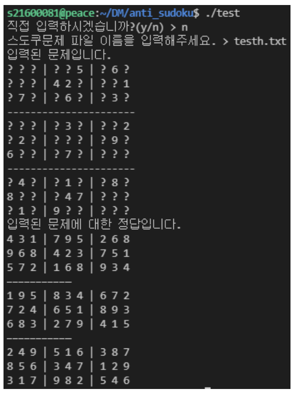
$$\wedge_{i=1}^{8}\wedge_{j=1}^{8}\wedge_{n=1}^{9}\neg(p(i,j,n)\wedge p(i+1,j+1,n))$$

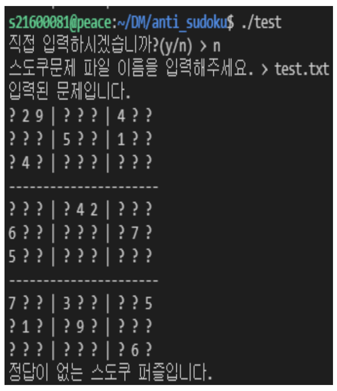
$$\wedge_{i=1}^{8}\wedge_{j=2}^{9}\wedge_{n=1}^{9}\neg(p(i,j,n)\wedge p(i+1,j-1,n))$$


```c
/* Q6, Q7: anti-king sudoku condition*/
// Q6 : Each cell can not have the same number with right-down diagonal cell's number
fprintf(fp,"; Q6\n") ;
fprintf(fp,"(assert (and ") ;
for (i = 1 ; i <= 8 ; i++) {
    fprintf(fp,"(and ") ;
    for (j = 1 ; j <= 8 ; j++) {
        fprintf(fp,"(and ") ;
        for (n = 1 ; n <= 9 ; n++) {
            fprintf(fp,"(not (and p%d%d%d p%d%d%d))", i, j, n, i+1, j+1, n);
        }
        fprintf(fp,")") ;
    }
    fprintf(fp,")") ;
}
fprintf(fp,"))\n") ;
// Q7 : Each cell can not have the same number with left-down diagonal cell's number
fprintf(fp,"; Q7\n") ;
fprintf(fp,"(assert (and ") ;
for (i = 1 ; i <= 8 ; i++) {
    fprintf(fp,"(and ") ;
    for (j = 2 ; j <= 9 ; j++) {
        fprintf(fp,"(and ") ;
        for (n = 1 ; n <= 9 ; n++) {
            fprintf(fp,"(not (and p%d%d%d p%d%d%d))", i, j, n, i+1, j-1, n);
        }
        fprintf(fp,")") ;
    }
    fprintf(fp,")") ;
}
fprintf(fp,"))\n") ;
```

3. Evaluation

As we mentioned above, through Z3(SAT solver), if there is an answer to the Sudoku puzzle, Z3 will present the model, and if not, the word 'unsat' will be output. To confirm this, we tested two Sudoku puzzles that we already know the correct answer, and do not have the correct answer.


```
s21600081@peace:~/DM/anti_sudoku$ ./test
직접 입력하시겠습니까?(y/n) > n
스도쿠문제 파일 이름을 입력해주세요. > testh.txt
입력된 문제입니다.
? ? ? | ? ? 5 | ? 6 ?
? ? ? | 4 2 ? | ? ? 1
? 7 ? | ? 6 ? | ? 3 ?
---------------------
? ? ? | ? 3 ? | ? ? 2
? 2 ? | ? ? ? | ? 9 ?
6 ? ? | ? 7 ? | ? ? ?
---------------------
? 4 ? | ? 1 ? | ? 8 ?
8 ? ? | ? 4 7 | ? ? ?
? 1 ? | 9 ? ? | ? ? ?
입력된 문제에 대한 정답입니다.
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
```


```
s21600081@peace:~/DM/anti_sudoku$ ./test
직접 입력하시겠습니까?(y/n) > n
스도쿠문제 파일 이름을 입력해주세요. > test.txt
입력된 문제입니다.
? 2 9 | ? ? ? | 4 ? ?
? ? ? | 5 ? ? | 1 ? ?
? 4 ? | ? ? ? | ? ? ?
---------------------
? ? ? | ? 4 2 | ? ? ?
6 ? ? | ? ? ? | ? 7 ?
5 ? ? | ? ? ? | ? ? ?
---------------------
7 ? ? | 3 ? ? | ? ? 5
? 1 ? | ? 9 ? | ? ? ?
? ? ? | ? ? ? | ? 6 ?
정답이 없는 스도쿠 퍼즐입니다.
```

Therefore, it came out as above, so it can be said that the program was well written.

4. Discussion

We constructed a program the Anti-king Sudoku with added rules over the basic Sudoku. At this point, we actually do not use the logic that express "No two cells adjacent to each other vertically, horizontally have the same number". This is because as we mention it in introduction, we cannot have duplicate numbers in a row or column, we do not need to check the two adjacent cells vertically, horizontally.

To express this in propositional logic, if the proposition that "the same number cannot come to two horizontally and vertically adjacent cells" is q and "the duplicate number cannot come from rows and columns" is p, p -> q is tautology, so if p is true, q is true.

5. Conclusion

By writing a program to solve the Anti-king Sudoku puzzle using Z3(SAT solver), we can quickly find out which problem is approached with a propositional logic and the answer to the problem exists. We applied SAT solver directly to relatively easy puzzles, which allowed us to set appropriate propositional variable and set logic accordingly, even if it was a non-puzzle problem, to solve other problems and it was very effective.

Nondango

1. Introduction

Nondango is a puzzle divided into 13 to 50 regions at a given 10*10 board. Each region contains at least two a maximum of eight connected cells, and some cells contain a circle. Also, initially all of these circles are white, and we have to follow the rules below and color them. Exactly one black circle must exist in each region and no circles of the same color must be placed in three successive cells. (Region is irrelevant) This puzzle can eventually be considered a matter of finding where the black circles should be located for a given problem. In other words, when imagining that a black circle is located in a cell, if a situation inevitably violates the conditions, (contradiction) the cell can be seen as a cell containing a white circle.

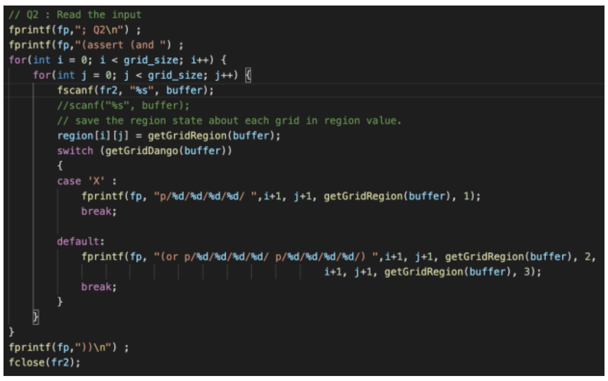
2. Approach

To solve this puzzle, we will use SAT solver in the same way as Anti-king Sudoku. A description of the SAT solver will be omitted. If a propositional variable is declared to solve Nondango and a propositional logic is written that meets the conditions of the puzzle, that is, if there is a value in which the entire logic is true by any evaluation, the model can be the correct answer to Nondango. Unlike Sudoku, the Nondango puzzle has a characteristic called region. So, when we declare the propositional variable, we should also consider the region. We included information about rows, columns, regions, and dango(the information of cells that is empty, or contains circle either white or black) in the propositional variable. In order to output the correct answer, information on rows, columns, and dango is sufficient, but in order to write logics for the region, the region should also be included when declaring a propositional variable.

P(i,j,r,d) holds when row i and column j and region r, and dango(empty or white or black). Therefore, we declare P(i,j,r,d) as up to 10*10*50*3 variables. (I =1~10, j =1~10, r =1~50(max), d =1~3(empty, white, black) Now we will use this propositional variable to write the logic as follows.

Here are the rules of the nondango.

1 Pre-assigned propositions.

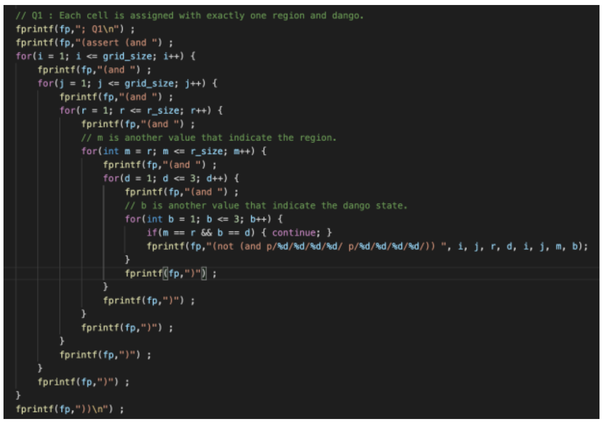


```
// Q2 : Read the input
fprintf(fp,"; Q2\n") ;
fprintf(fp,"(assert (and ") ;
for(int i = 0; i < grid_size; i++) {
    for(int j = 0; j < grid_size; j++) {
        fscanf(fr2, "%s", buffer);
        //scanf("%s", buffer);
        // save the region state about each grid in region value.
        region[i][j] = getGridRegion(buffer);
        switch (getGridDango(buffer))
        {
        case 'X' :
            fprintf(fp, "p/%d/%d/%d/%d/ ",i+1, j+1, getGridRegion(buffer), 1);
            break;

        default:
            fprintf(fp, "(or p/%d/%d/%d/%d/ p/%d/%d/%d/%d/) ",i+1, j+1, getGridRegion(buffer), 2,
                                                            i+1, j+1, getGridRegion(buffer), 3);
            break;
        }
    }
}
fprintf(fp,"))\n") ;
fclose(fr2);
```

2 Each cell is assigned with exactly one region and dango.

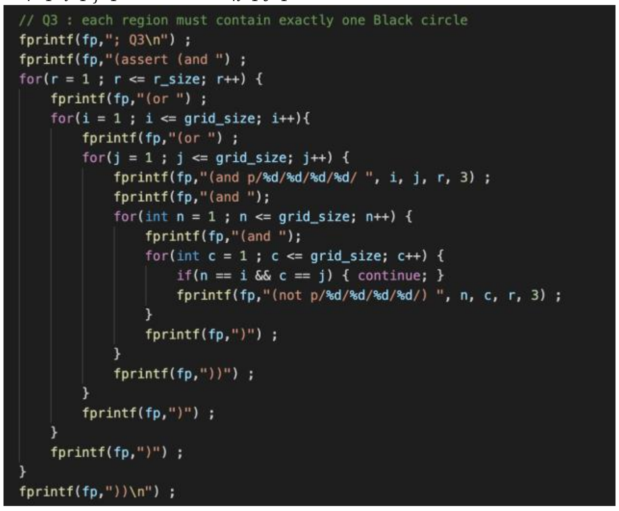
$$\bigwedge_{i=1}^{10}\bigwedge_{j=1}^{10}\bigwedge_{r=1}^{r_{s}ize}\bigwedge_{m=r}^{r\_size}\bigwedge_{d=1}^{3}\bigwedge_{b=1}^{3}[\neg\{(r=m)\wedge(d=b)\}$$
$$\rightarrow\ \neg\{p(i,j,r,d)\wedge p(i,j,m,b)\}]$$



```
// Q1 : Each cell is assigned with exactly one region and dango.
fprintf(fp,"; Q1\n") ;
fprintf(fp,"(assert (and ") ;
for(i = 1; i <= grid_size; i++) {
    fprintf(fp,"(and ") ;
    for(j = 1; j <= grid_size; j++) {
        fprintf(fp,"(and ") ;
        for(r = 1; r <= r_size; r++) {
            fprintf(fp,"(and ") ;
            // m is another value that indicate the region.
            for(int m = r; m <= r_size; m++) {
                fprintf(fp,"(and ") ;
                for(d = 1; d <= 3; d++) {
                    fprintf(fp,"(and ") ;
                    // b is another value that indicate the dango state.
                    for(int b = 1; b <= 3; b++) {
                        if(m == r && b == d) { continue; }
                        fprintf(fp,"(not (and p/%d/%d/%d/%d/ p/%d/%d/%d/%d/)) ", i, j, r, d, i, j, m, b);
                    }
                    fprintf(fp,")") ;
                }
                fprintf(fp,")") ;
            }
            fprintf(fp,")") ;
        }
        fprintf(fp,")") ;
    }
    fprintf(fp,")") ;
}
fprintf(fp,"))\n") ;
```

3 Each region must contain exactly on black circle

$$\bigwedge_{r=1}^{r\_size}\bigvee_{i=1}^{10}\bigvee_{j=1}^{10}[p(i,j,r,3)\wedge\{\bigwedge_{n=1}^{10}\bigwedge_{c=1}^{10}(\neg((n=i)\wedge(c=j))\rightarrow\neg p(n,c,r,3))\}]$$

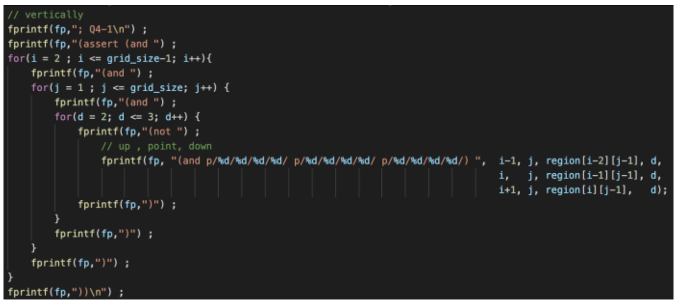


```
// Q3 : each region must contain exactly one Black circle
fprintf(fp,"; Q3\n") ;
fprintf(fp,"(assert (and ") ;
for(r = 1 ; r <= r_size; r++) {
    fprintf(fp,"(or ") ;
    for(i = 1 ; i <= grid_size; i++){
        fprintf(fp,"(or ") ;
        for(j = 1 ; j <= grid_size; j++) {
            fprintf(fp,"(and p/%d/%d/%d/%d/ ", i, j, r, 3) ;
            fprintf(fp,"(and ");
            for(int n = 1 ; n <= grid_size; n++) {
                fprintf(fp,"(and ");
                for(int c = 1 ; c <= grid_size; c++) {
                    if(n == i && c == j) { continue; }
                    fprintf(fp,"(not p/%d/%d/%d/%d/) ", n, c, r, 3) ;
                }
                fprintf(fp,")") ;
            }
            fprintf(fp,"))") ;
        }
        fprintf(fp,")") ;
    }
    fprintf(fp,")") ;
}
fprintf(fp,"))\n") ;
```

4 There must be no three circles of the same color placing on three consecutive cells
r[i][j] has the region number of region about nondango cell[i][j] (i : row, j: column).

4.1 Vertically

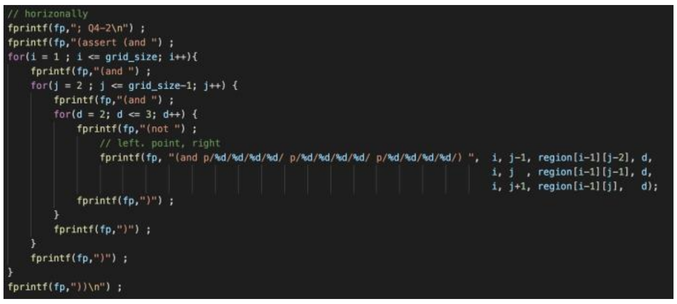
$$\bigwedge_{i=2}^{9}\bigwedge_{j=1}^{10}\bigwedge_{d=2}^{3}\neg\left\{\begin{array}{l}p(i-1,j,r[i-2][j-1],d)\\ \wedge p(i,j,r[i-1][j-1],d)\\ \wedge p(i+1,j,r[i][j-1],d)\end{array}\right\}$$



```
// vertically
fprintf(fp,"; Q4-1\n") ;
fprintf(fp,"(assert (and ") ;
for(i = 2 ; i <= grid_size-1; i++){
    fprintf(fp,"(and ") ;
    for(j = 1 ; j <= grid_size; j++) {
        fprintf(fp,"(and ") ;
        for(d = 2; d <= 3; d++) {
            fprintf(fp,"(not ") ;
            // up , point, down
            fprintf(fp, "(and p/%d/%d/%d/%d/ p/%d/%d/%d/%d/ p/%d/%d/%d/%d/) ",  i-1, j, region[i-2][j-1], d,
                                                                                i,   j, region[i-1][j-1], d,
                                                                                i+1, j, region[i][j-1],   d);
            fprintf(fp,")") ;
        }
        fprintf(fp,")") ;
    }
    fprintf(fp,")") ;
}
fprintf(fp,"))\n") ;
```

4.2 Horizonally

$$\bigwedge_{i=1}^{10}\bigwedge_{j=2}^{9}\bigwedge_{d=2}^{3}\neg\left\{\begin{array}{l}p(i,j-1,r[i-1][j-2],d)\\ \wedge p(i,j,r[i-1][j-1],d)\\ \wedge p(i,j+1,r[i-1][j],d)\end{array}\right\}$$

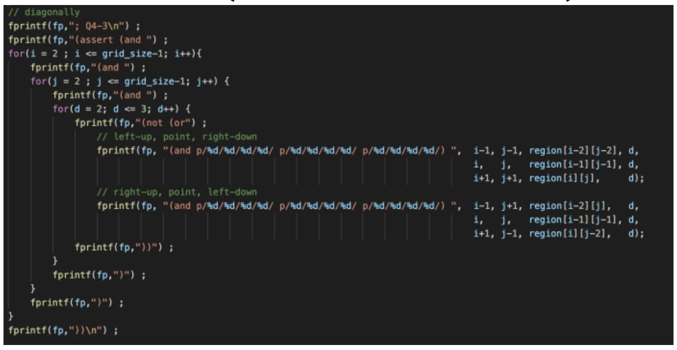


```
// horizonally
fprintf(fp,"; Q4-2\n") ;
fprintf(fp,"(assert (and ") ;
for(i = 1 ; i <= grid_size; i++){
    fprintf(fp,"(and ") ;
    for(j = 2 ; j <= grid_size-1; j++) {
        fprintf(fp,"(and ") ;
        for(d = 2; d <= 3; d++) {
            fprintf(fp,"(not ") ;
                // left. point, right
                fprintf(fp, "(and p/%d/%d/%d/%d/ p/%d/%d/%d/%d/ p/%d/%d/%d/%d/) ",  i, j-1, region[i-1][j-2], d,
                                                                                    i, j  , region[i-1][j-1], d,
                                                                                    i, j+1, region[i-1][j],   d);
            fprintf(fp,")") ;
        }
        fprintf(fp,")") ;
    }
    fprintf(fp,")") ;
}
fprintf(fp,"))\n") ;
```

4.3 Diagonally

$$\bigwedge_{i=2}^{9}\bigwedge_{j=2}^{9}\bigwedge_{d=2}^{3}\neg\begin{Bmatrix} p(i-1,j-1,r[i-2][j-2],d) \\ \wedge p(i,j,r[i-1][j-1],d) \\ \wedge p(i+1,j+1,r[i][j],d) \end{Bmatrix}$$

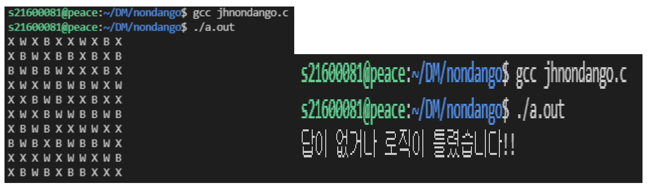
$$\bigwedge_{i=2}^{9}\bigwedge_{j=2}^{9}\bigwedge_{d=2}^{3}\neg\begin{Bmatrix} p(i-1,j+1,r[i-2][j],d) \\ \wedge p(i,j,r[i-1][j-1],d) \\ \wedge p(i+1,j-1,r[i][j-2],d) \end{Bmatrix}$$



```
// diagonally
fprintf(fp,"; Q4-3\n") ;
fprintf(fp,"(assert (and ") ;
for(i = 2 ; i <= grid_size-1; i++){
    fprintf(fp,"(and ") ;
    for(j = 2 ; j <= grid_size-1; j++) {
        fprintf(fp,"(and ") ;
        for(d = 2; d <= 3; d++) {
            fprintf(fp,"(not (or") ;
                // left-up, point, right-down
                fprintf(fp, "(and p/%d/%d/%d/%d/ p/%d/%d/%d/%d/ p/%d/%d/%d/%d/) ",  i-1, j-1, region[i-2][j-2], d,
                                                                                    i,   j,   region[i-1][j-1], d,
                                                                                    i+1, j+1, region[i][j],     d);
                // right-up, point, left-down
                fprintf(fp, "(and p/%d/%d/%d/%d/ p/%d/%d/%d/%d/ p/%d/%d/%d/%d/) ",  i-1, j+1, region[i-2][j],   d,
                                                                                    i,   j,   region[i-1][j-1], d,
                                                                                    i+1, j-1, region[i][j-2],   d);
            fprintf(fp,"))") ;
        }
        fprintf(fp,")") ;
    }
    fprintf(fp,")") ;
}
fprintf(fp,"))\n") ;
```

3. Evaluation

Nondango can have multiple solutions, so it is necessary to make sure that the model resulting from the result satisfies the condition exactly.



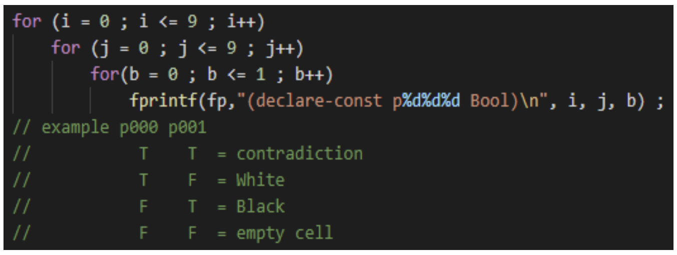
```
s21600081@peace:~/DM/nondango$ gcc jhnondango.c
s21600081@peace:~/DM/nondango$ ./a.out
X W X B X X W X B X
X B W X B B X B X B
B W B B W X X X B X
X W X W B W B W X W
X X B W B X X B X X
X W X B W W B B W B
X B W B X X W W X X
B W B X B W B B W X
X X X W X W W X W B
X B W B X B B X X X
```

```
s21600081@peace:~/DM/nondango$ gcc jhnondango.c
s21600081@peace:~/DM/nondango$ ./a.out
답이 없거나 로직이 틀렸습니다!!
```

Therefore, results came out as above, and if you look at it, you can see that the model came out well according to the conditions, and it can be seen that the Nondango problem, which has no answer, is being printed well. It can be said that the program was well written.

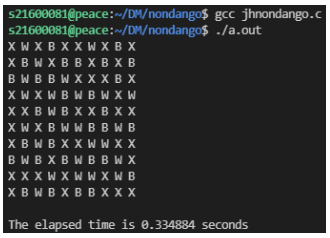
4. Discussion

In the above problem, each of us considered for a long time how to declare propositional variables. When region is included, up to 15,000 propositional variables are declared. Constructing logics using these variables takes considerable time. Therefore, by excepting for region and reducing dango's domain, the number of variables was reduced to 200(10*10*2) from 15,000. Propositional variables were declared p(i,j,d). [d=0 is true(white), d=1 is true(black), both are false(empty)]



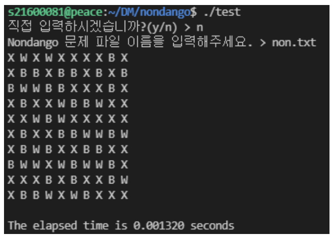
```
for (i = 0 ; i <= 9 ; i++)
    for (j = 0 ; j <= 9 ; j++)
        for(b = 0 ; b <= 1 ; b++)
            fprintf(fp,"(declare-const p%d%d%d Bool)\n", i, j, b) ;
// example p000 p001
//          T    T  = contradiction
//          T    F  = White
//          F    T  = Black
//          F    F  = empty cell
```

The figure above is another nondango program. In this case, when declaring logic, information about the region does not be included in propositional variables. Therefore, the program should check separately the regions. For this reason, we declared an array that has sorted regions in order.



```
s21600081@peace:~/DM/nondango$ gcc jhnondango.c
s21600081@peace:~/DM/nondango$ ./a.out
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

The elapsed time is 0.334884 seconds
```



```
s21600081@peace:~/DM/nondango$ ./test
직접 입력하시겠습니까?(y/n) > n
Nondango 문제 파일 이름을 입력해주세요. > non.txt
X W X W X X X X B X
X B B X B B X B X B
B W W B B X X X B X
X B X X W B B W X X
X X W B W X X X X X
X B X X B B W W B W
X B W B X X B B X X
B W W X W B W W B X
X X X B X B X X B W
X B B W X W B X X X

The elapsed time is 0.001320 seconds
```

The first is to include the region in the propositional variable, and the second is not. The first result is 0.33489sec, and the second result is 0.00132sec. This is about 254 times the result. Through this, it was found that the program efficiency differed significantly depending on how the propositional variable was declared and how the logic was written, so it was very important to declare the propositional variable. In addition, you can see these solutions are different, both solutions meet conditions. Since SAT solver only informs satisfactory models, one of the assignments, different models are bound to appear for each program, which can also be confirmed through the above. In fact, condition cannot be accurately written as a propositional formula by a method declaring propositional variables except region. However, from a programming point of view, it is possible to efficiently code to satisfy its condition. If we include region in propositional variables, we can easily get result by repeating all rows, columns, and regions, but it can take a long time, we have also thought of other perspectives.

5. Conclusion

Similarly, like Anti-king Sudoku, we were able to quickly obtain the solution to this puzzle by creating a program to solve Nondango. As mentioned in discussion part, in the process of solving the problem using SAT solver, we found that how to declare propositional variables in programming was a very strong feature, and we found that investing a plenty of that time is essential for good programming. In addition, we were able to understand the use of SAT solver and the logic of the program well.