Graph Theory

SUDOKU AS SAT/SMT PROBLEM SOLVED BY MICROSOFT Z3 SOLVER

王梓帆 | 顏子傑 | 林冠名

Date: 2022 / 06 / 10 Fri.

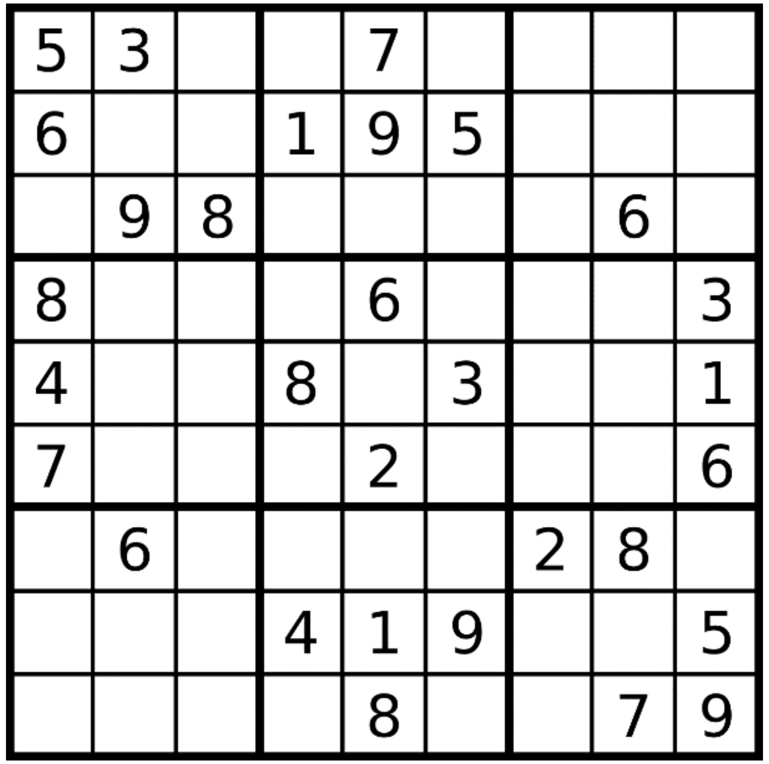
Group: G

Outline

01

章節 PART

Introduction

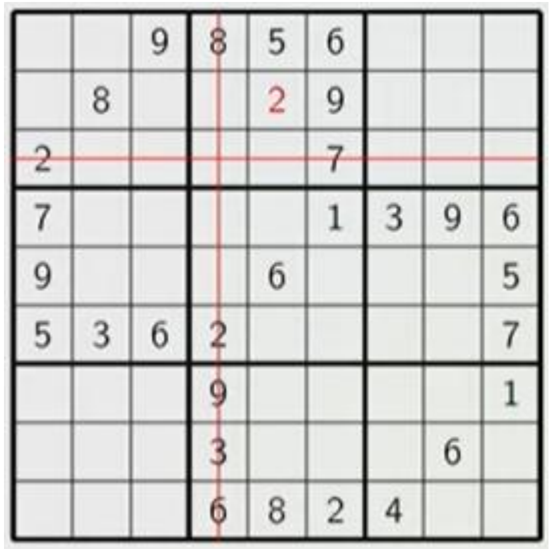


5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

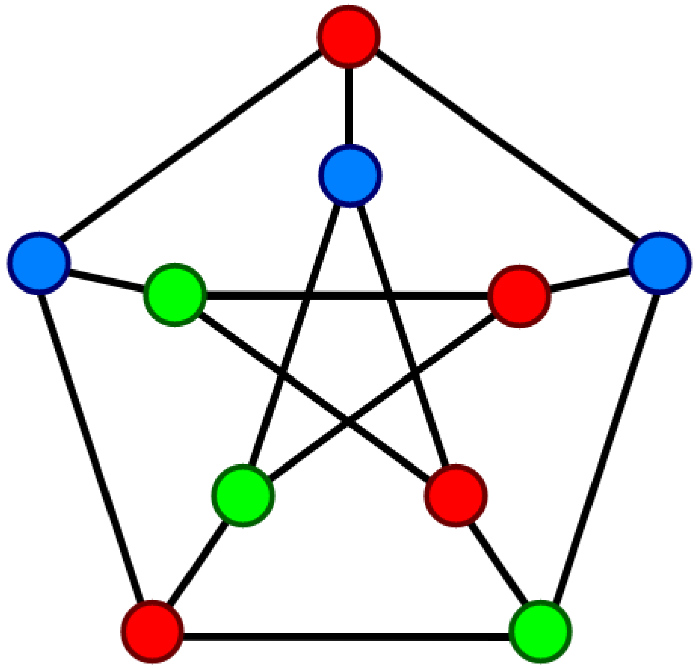
Basic Rules of Sudoku

- ◻ Every entry has to contain a single number
- ◻ Only the numbers from 1 through to 9 can be used
- ◻ Each 3 × 3 box can only contain each number from 1 to 9 once
- ◻ Each horizontal row can only contain each number from 1 to 9 once
- ◻ Each vertical column can only contain each number from 1 to 9 once
- ◻ Game will Terminate when all the entries are filled

- ◻ *Completing Quasigroups or Latin Squares:*
  - ➢ *A Structured Graph Coloring Problem*



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

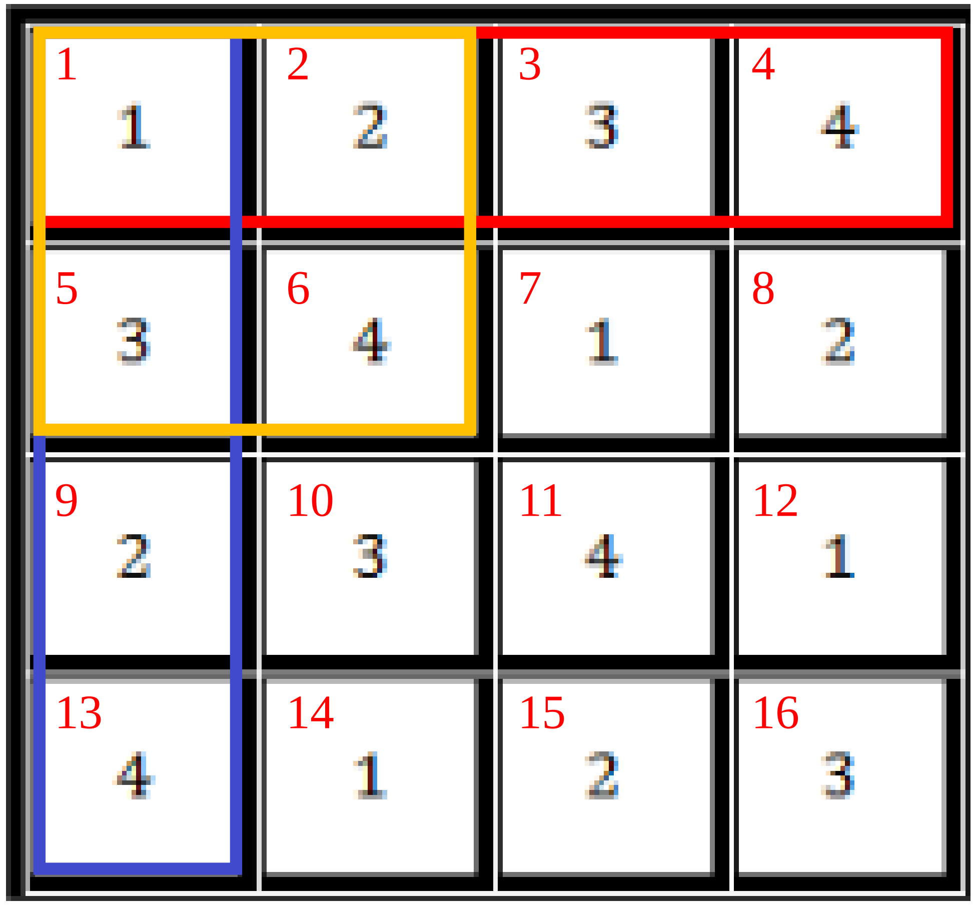


Chromatic Number

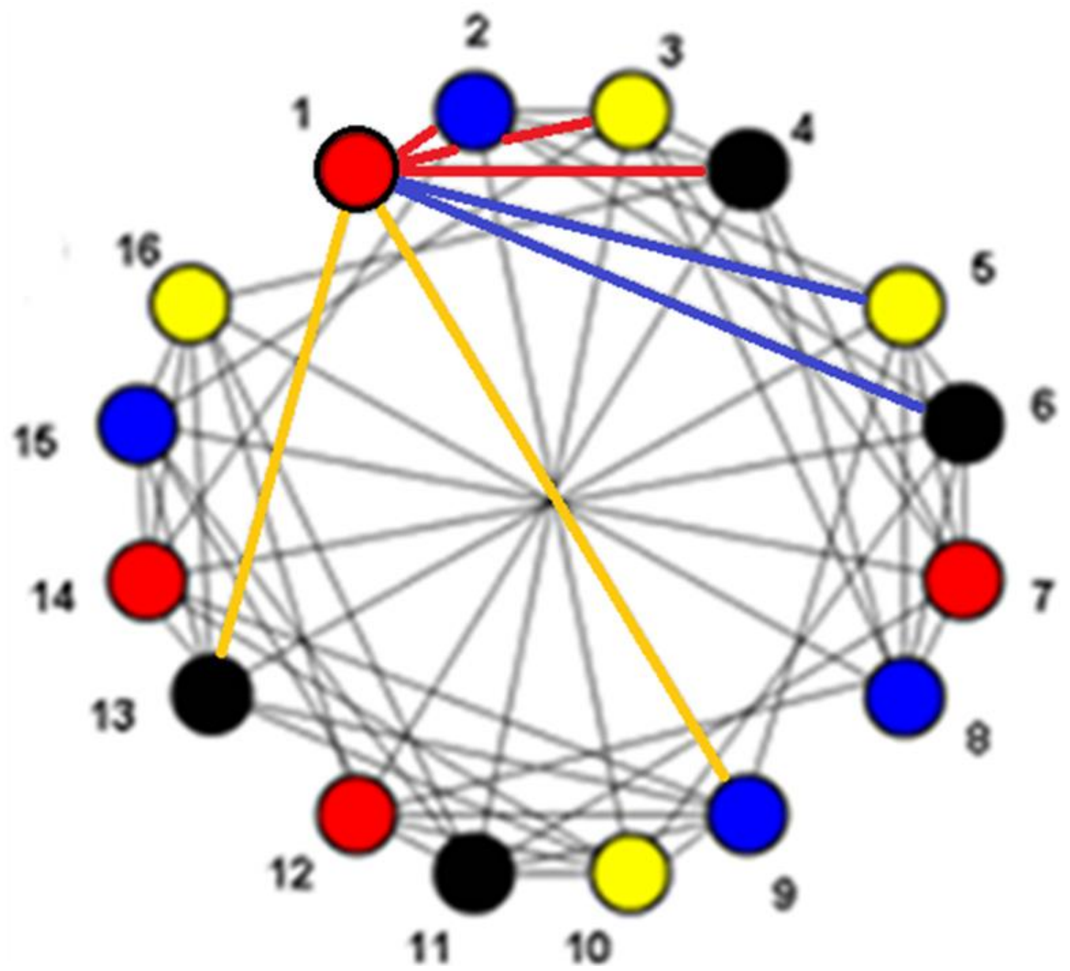
- Chromatic number of a graph G is denoted as X(G)
- The minimum number of colors needed to label the vertices
- Adjacent vertices receive different colors.
- Take an Petersen Graph as Example: X(G) = 3

Mapping Sudoku as Chromatic Number Problem

- Take 4 × 4 Matrix as example



1 1	2 2	3 3	4 4
5 3	6 4	7 1	8 2
9 2	10 3	11 4	12 1
13 4	14 1	15 2	16 3



Algorithm & Constraints

02

章節 PART

Backtracking

- Sudoku can be solved by assigning numbers one by one
- Before assigning a number, check whether it is safe to assign
- Check all numbers are not in the same row, column or 3×3 box
- Assign the number and check recursively.
- If none of the number leads to a solution, return false.

Sudoku as SAT Problem

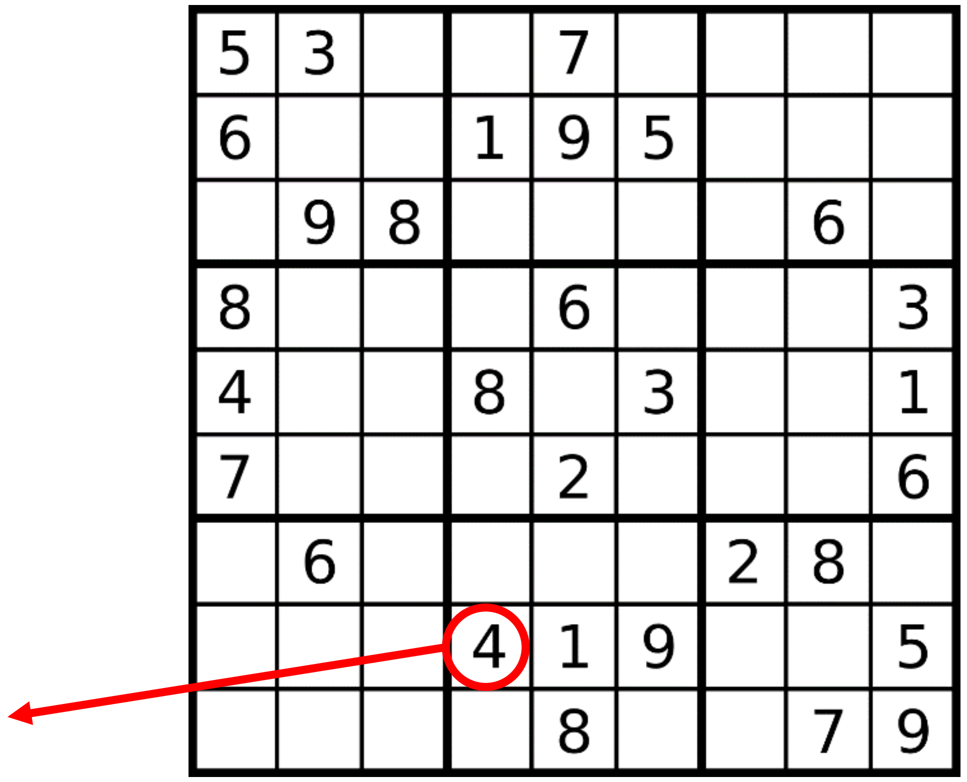
✓ Discussed in *Sudoku as SAT Problem*

☐ Proposition Variables

➢ $9 \times 9 \times 9 =$ 729 Boolean Variables

➢ Defined as s_{xyz}

➢ For Example: $s_{484} = True \leftrightarrow S[4,8] = 4$



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

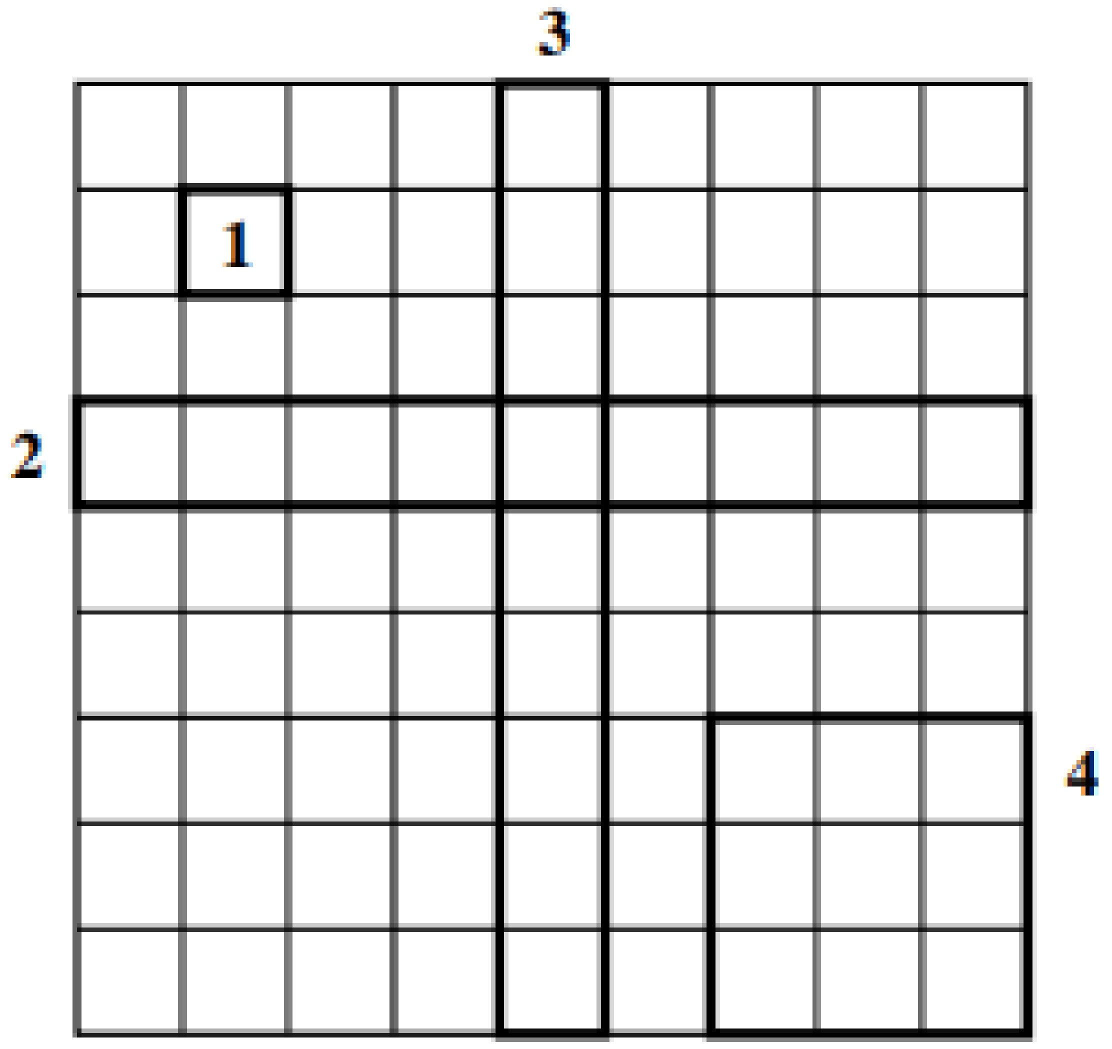
Sudoku as SAT Problem

- Minimum encoding and Extended Encoding

- Two encoding style of Latin square discussed

➢ *Completing Quasigroups or Latin Squares:*

➢ *A Structured Graph Coloring Problem*



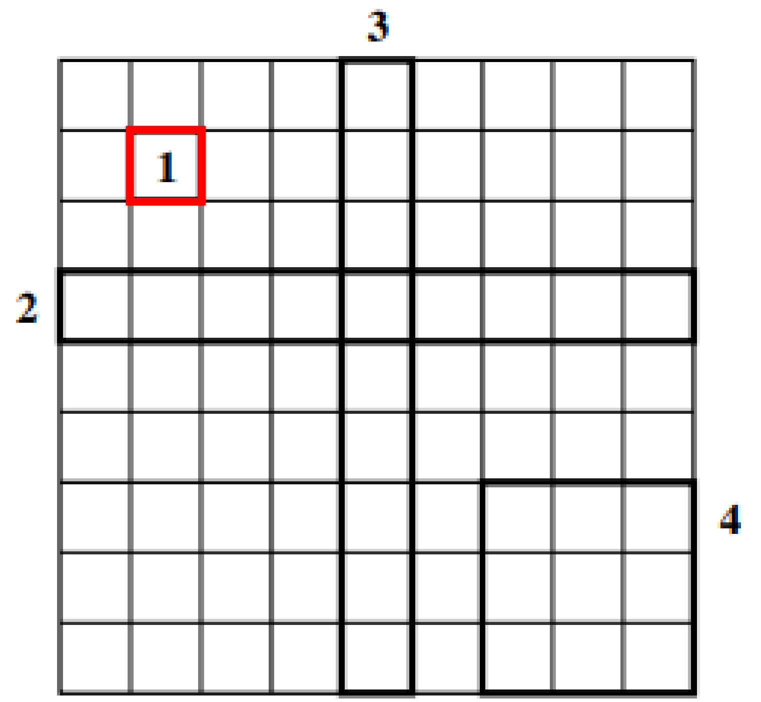
Sudoku as SAT Problem

- Constraints by minimum encoding

1. There is at least one number in each entry

- General Form: $\bigwedge_{x=1}^{9} \bigwedge_{x=1}^{9} \bigvee_{z=1}^{9} s_{xyz}$

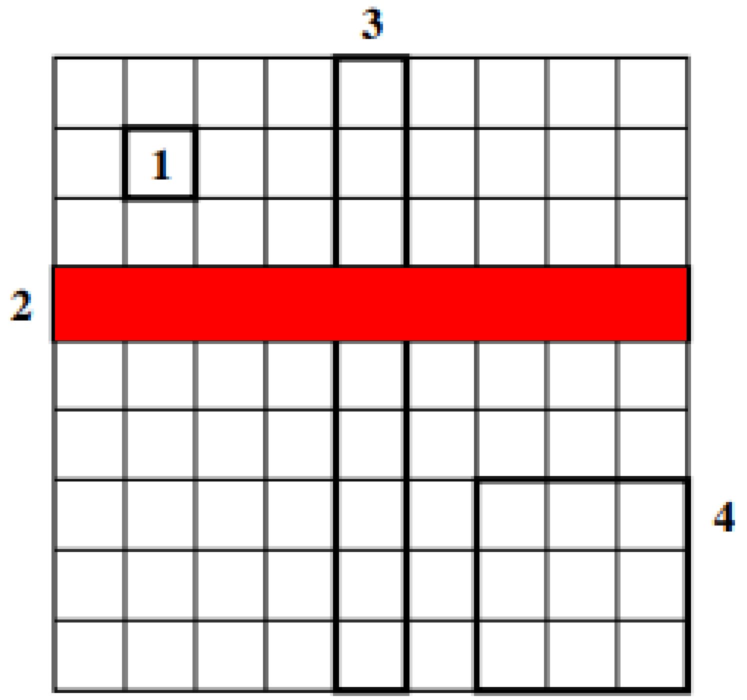
- For example: $(s_{221} \vee s_{222} \vee \ldots \vee s_{229})$



Sudoku as SAT Problem

- Constraints by minimum encoding

2. Each number appears at most once in each row



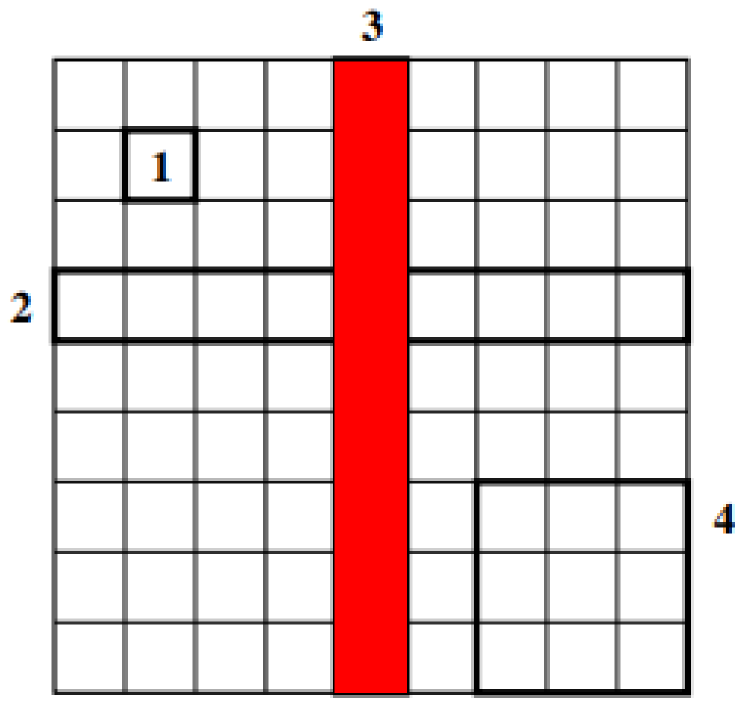
- General Form: $\bigwedge_{y=1}^{9} \bigwedge_{z=1}^{9} \bigwedge_{x=1}^{8} \bigwedge_{i=x+1}^{9} (\neg s_{xyz} \vee \neg s_{iyz})$

- For example:
- $(\neg s_{141} \vee \neg s_{241}) \wedge (\neg s_{241} \vee \neg s_{341}) \wedge \ldots \wedge (\neg s_{841} \vee \neg s_{941})$

Sudoku as SAT Problem

- Constraints by minimum encoding

3. Each number appears at most once in each column



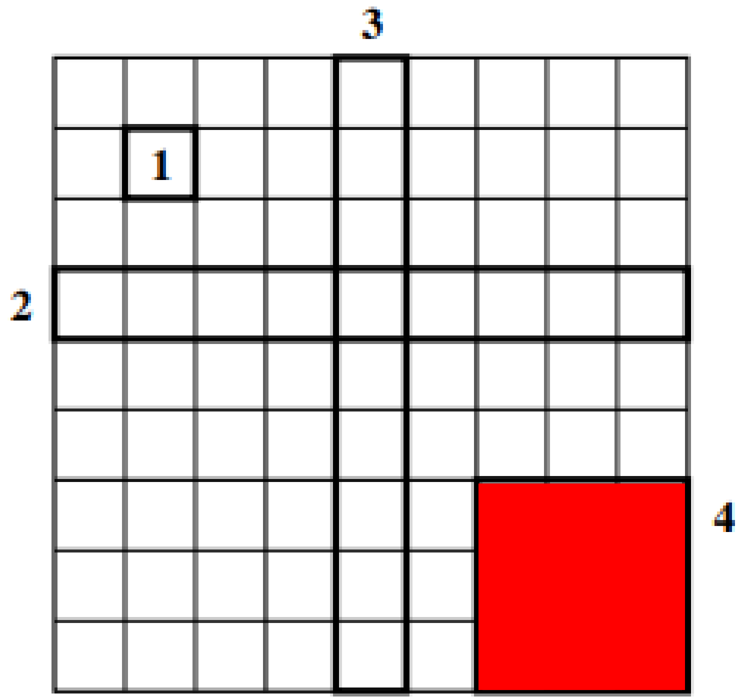
- General Form: $\bigwedge_{x=1}^{9} \bigwedge_{z=1}^{9} \bigwedge_{y=1}^{8} \bigwedge_{i=y+1}^{9} (\neg s_{xyz} \vee \neg s_{xiz})$

- For example:
- $(\neg s_{511} \vee \neg s_{521}) \wedge (\neg s_{521} \vee \neg s_{531}) \wedge \ldots \wedge (\neg s_{581} \vee \neg s_{591})$

Sudoku as SAT Problem

- Constraints by minimum encoding

4. Each number appears at most once in each box

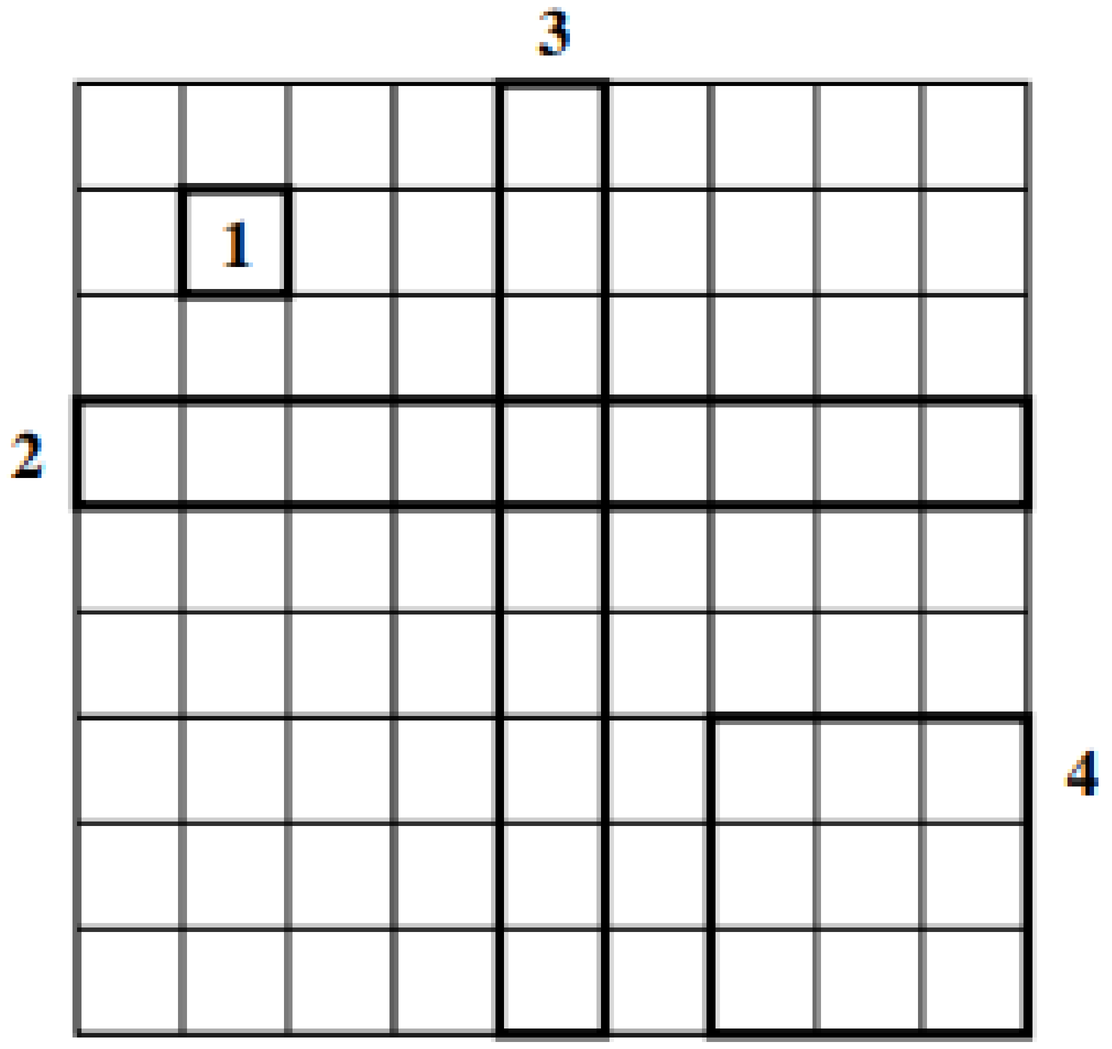


- General Form:
- $\bigwedge_{z=1}^{9} \bigwedge_{i=0}^{2} \bigwedge_{j=0}^{2} \bigwedge_{x=1}^{3} \bigwedge_{y=1}^{2} \bigwedge_{k=y+1}^{3} (\neg s_{(3i+x)(3j+y)z} \vee \neg s_{(3i+x)(3j+k)z})$

- For example:
- $(\neg s_{771} \vee \neg s_{781}) \wedge (\neg s_{781} \vee \neg s_{791}) \wedge \dots \wedge (\neg s_{891} \vee \neg s_{991})$

Sudoku as SAT Problem

- Extended Encoding
- Based on minimum encoding
- More intuitive when solving Sudoku problems



Sudoku as SAT Problem

- Constraints by extended encoding

1. There is at most one number in each entry

- General Form: $\bigwedge_{x=1}^{9} \bigwedge_{y=1}^{9} \bigwedge_{z=1}^{8} \bigwedge_{i=z+1}^{9} (\neg s_{xyz} \vee \neg s_{xyi})$

- For example:
- $(\neg s_{111} \vee \neg s_{112}) \wedge (\neg s_{112} \vee \neg s_{113}) \wedge \ldots \wedge (\neg s_{118} \vee \neg s_{119})$

Sudoku as SAT Problem

- Constraints by extended encoding

2. Each number appears at least once in each row

- General Form: $\bigwedge_{y=1}^{9} \bigwedge_{z=1}^{9} \bigvee_{x=1}^{9} s_{xyz}$

- For example: $(s_{111} \vee s_{211} \vee \ldots \vee s_{911})$

Sudoku as SAT Problem

- Constraints by extended encoding

3. Each number appears at least once in each column

- General Form: $\bigwedge_{y=1}^{9} \bigwedge_{z=1}^{9} \bigvee_{x=1}^{9} s_{xyz}$

- For example: $(s_{111} \vee s_{121} \vee \ldots \vee s_{191})$

Sudoku as SAT Problem

- Constraints by extended encoding

4. Each number appears at least once in each 3×3 box

- General Form:
- $\bigvee_{z=1}^{9} \bigwedge_{i=0}^{2} \bigwedge_{j=0}^{2} \bigwedge_{x=1}^{3} \bigwedge_{y=1}^{3} (s_{(3i+x)(3j+y)z})$

- For example: $(s_{111} \vee s_{121} \vee \dots \vee s_{331})$

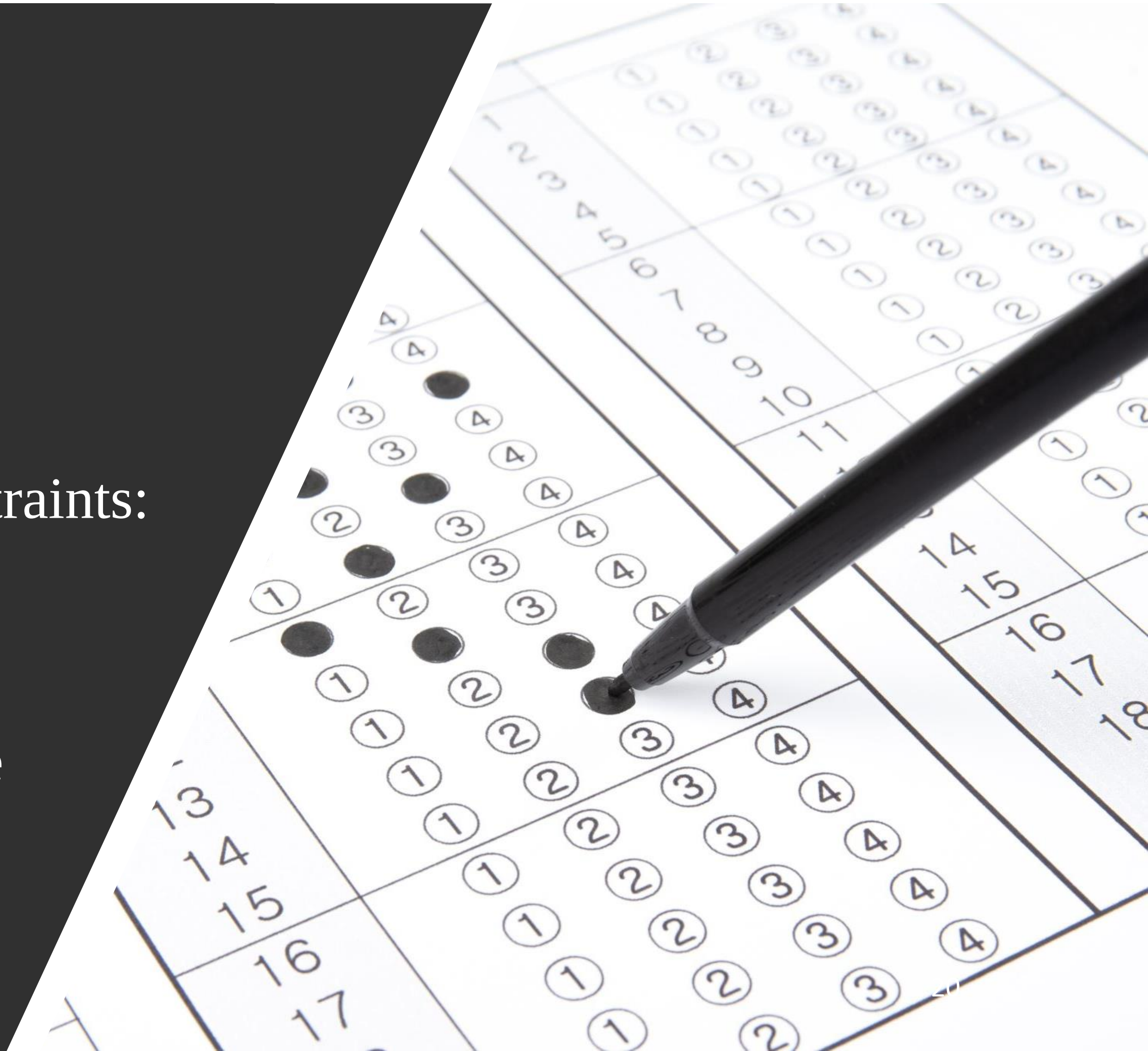
Sudoku as SAT Problem

- Constraints by minimum encoding

- Totally 8,829 clauses

- One Nine-ary clauses from At-least-one constraints:
  - $9 \times 9 = 81$

- Three sets of binary clauses from at-most-one constraints:
  - $3 \times 9 \times 9 \times C_{9,2} = 8748$



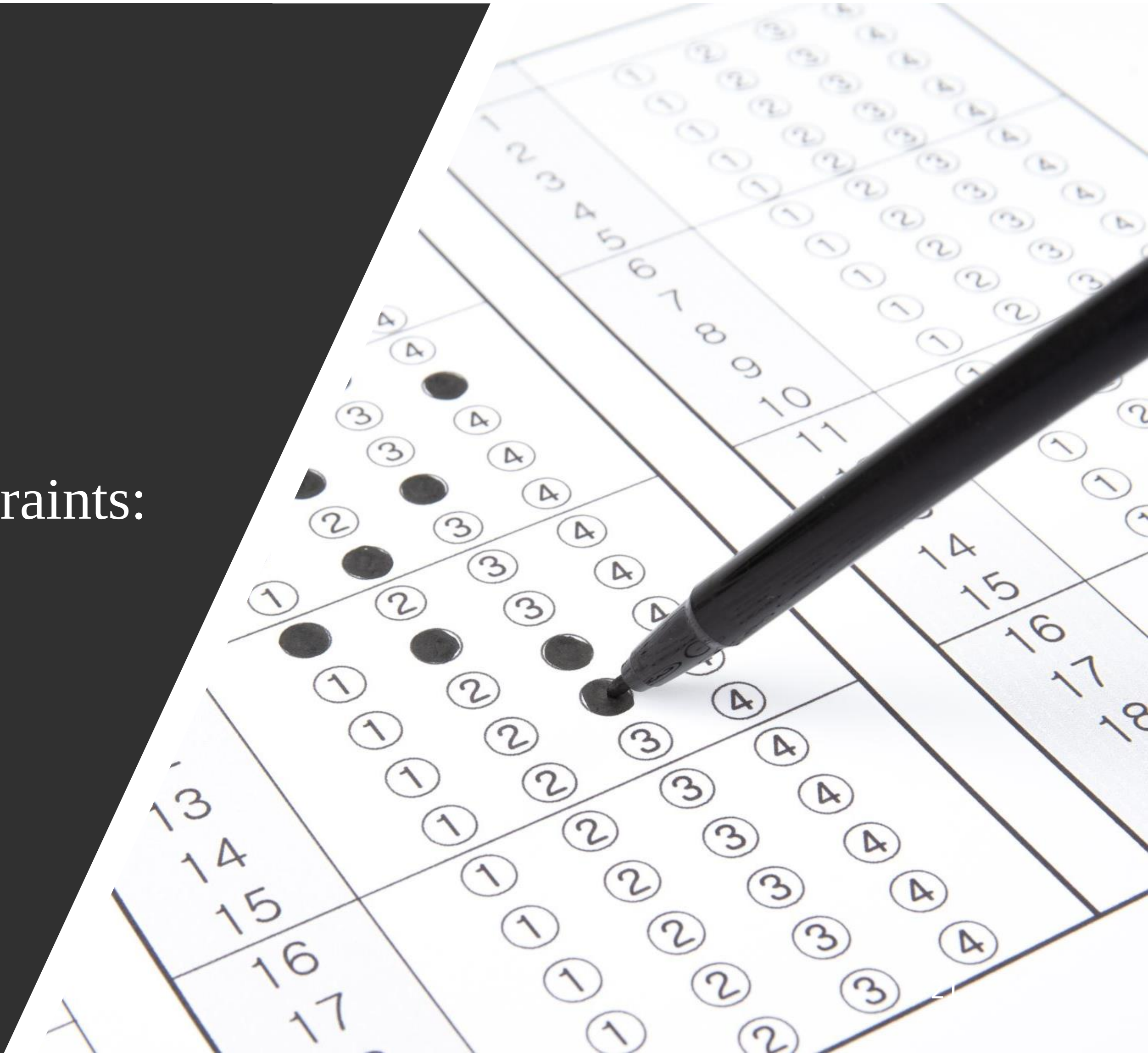
Sudoku as SAT Problem

- Constraints by extended encoding

- Totally 11,988 clauses

- One Nine-ary clauses from At-least-one constraints:
  - $4 \times 9 \times 9 = 324$

- Three sets of binary clauses from at-most-one constraints:
  - $4 \times 9 \times 9 \times C_{9,2} = 11664$

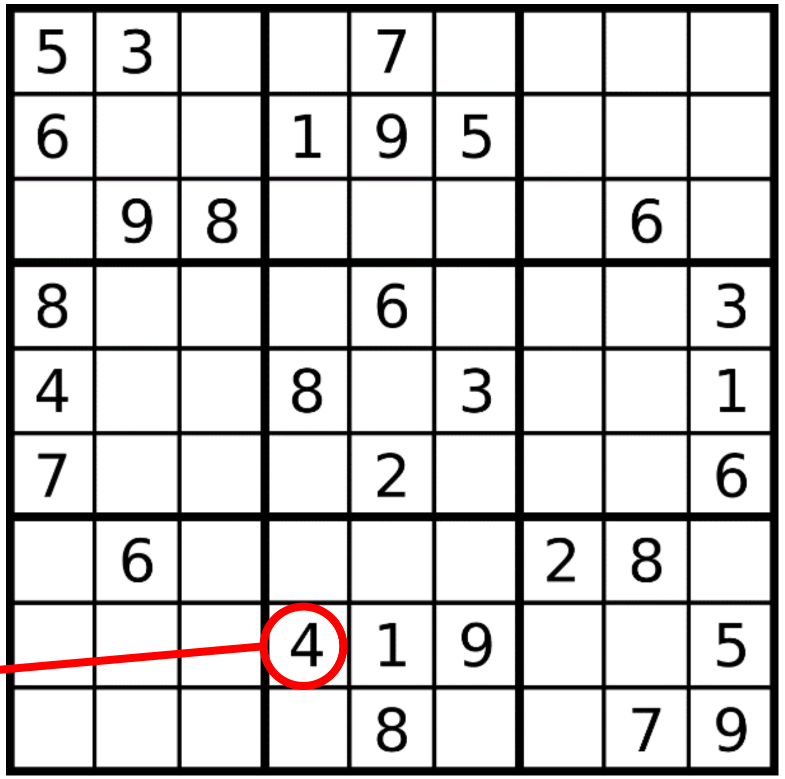


Sudoku as SMT Problem

✓ Discussed in *An Alldifferent Constraint Solver in SMT*

☐ Proposition Variables

- 9 × 9 = 81 Integer Variables
- Defined as S_{xy}
- For Example: $S_{48} = 4 \leftrightarrow S[4,8] = 4$



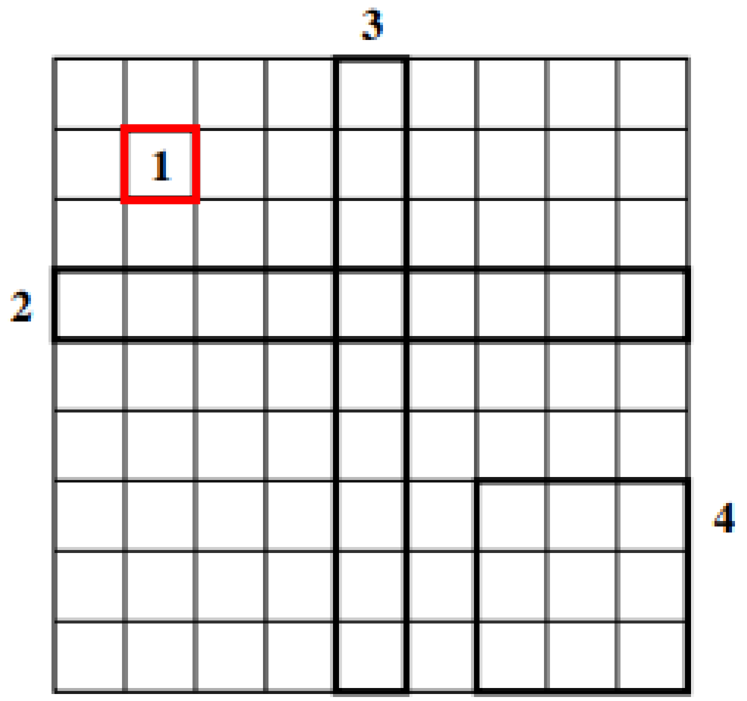
5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Sudoku as SMT Problem

1. Fill in a number from one to nine in each entry

- General Form: $\bigwedge_{x=1}^{9} \bigwedge_{y=1}^{9} (S_{xy} > 0 \wedge S_{xy} < 10)$

- For example: $(S_{22} > 0) \wedge (S_{22} < 10)$

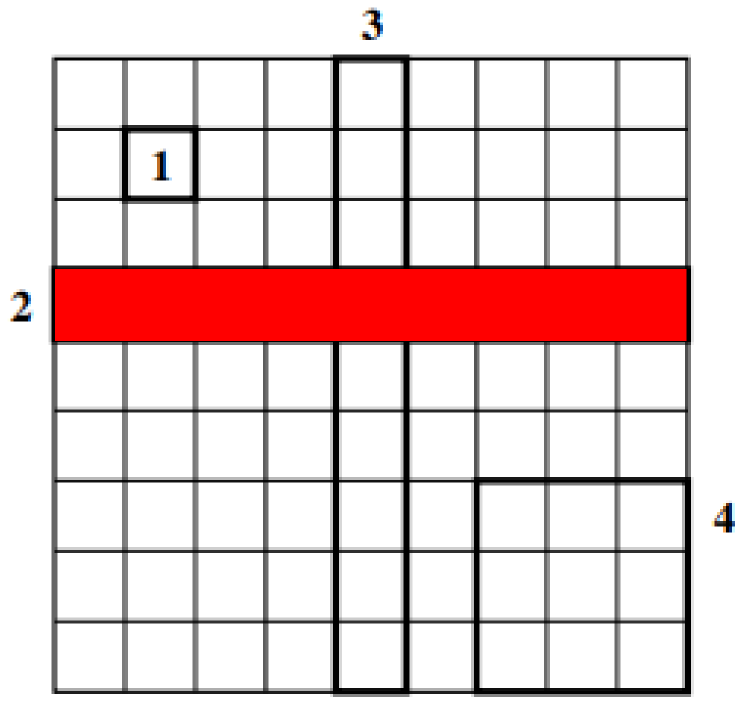


Sudoku as SMT Problem

2. Numbers in each row are all different

- General Form: $\bigwedge_{x=1}^{9} \bigwedge_{i=1}^{9} alldifferent(\, S_{iy})$

- For example: $alldifferent(\, S_{14}, S_{24}, \ldots, S_{94})$

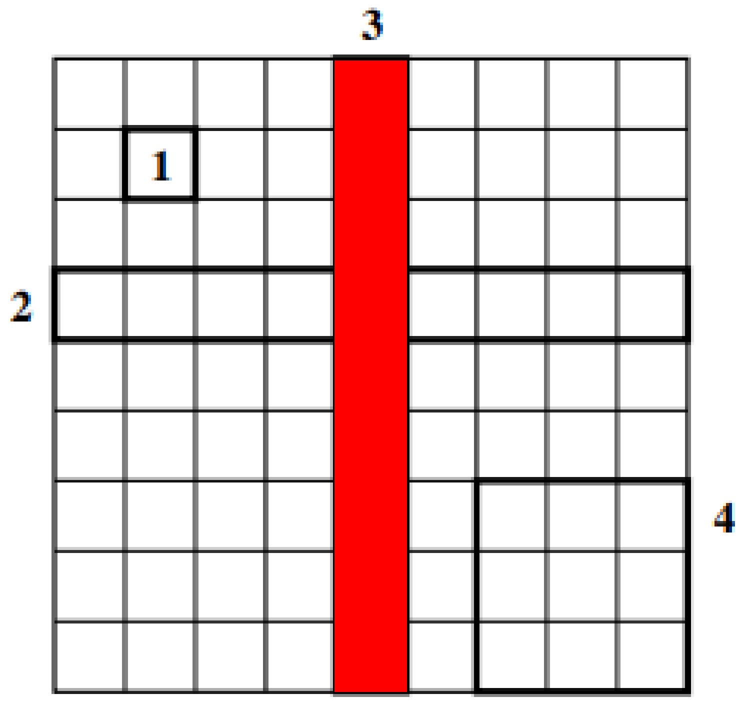


Sudoku as SMT Problem

3. Numbers in each column are all different

➢ General Form: $\bigwedge_{y=1}^{9} \bigwedge_{i=1}^{9} alldifferent(\, S_{xi})$

➢ For example: $alldifferent(\, S_{11}, S_{12}, \ldots, S_{19})$

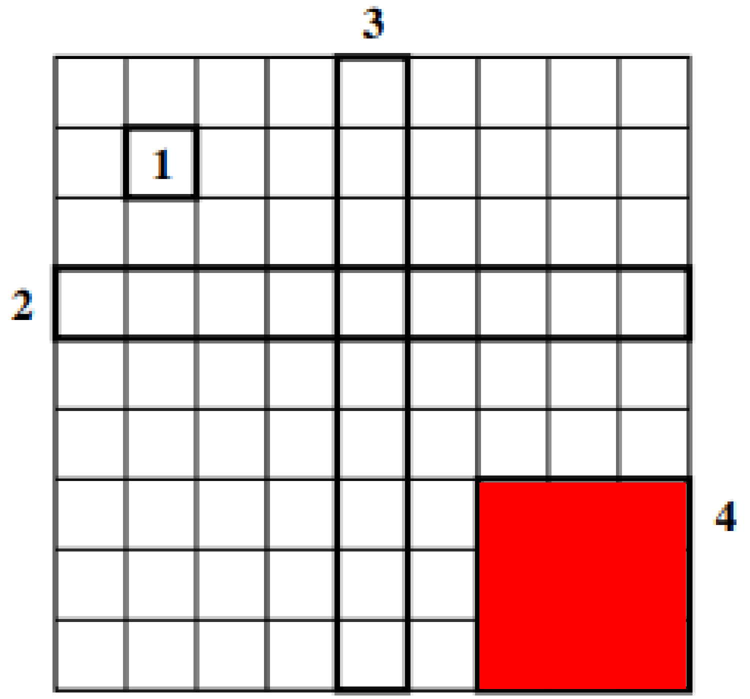


Sudoku as SMT Problem

4. Numbers in each 3 × 3 box are all different

- General Form: $\bigwedge_{i=0}^{2} \bigwedge_{i=0}^{2} \bigwedge_{x=1}^{3} \bigwedge_{i=1}^{3} alldifferent(\, S_{(3i+x)(3i+y)})$

- For example: $alldifferent(\, S_{11}, S_{12}, \ldots, S_{33})$

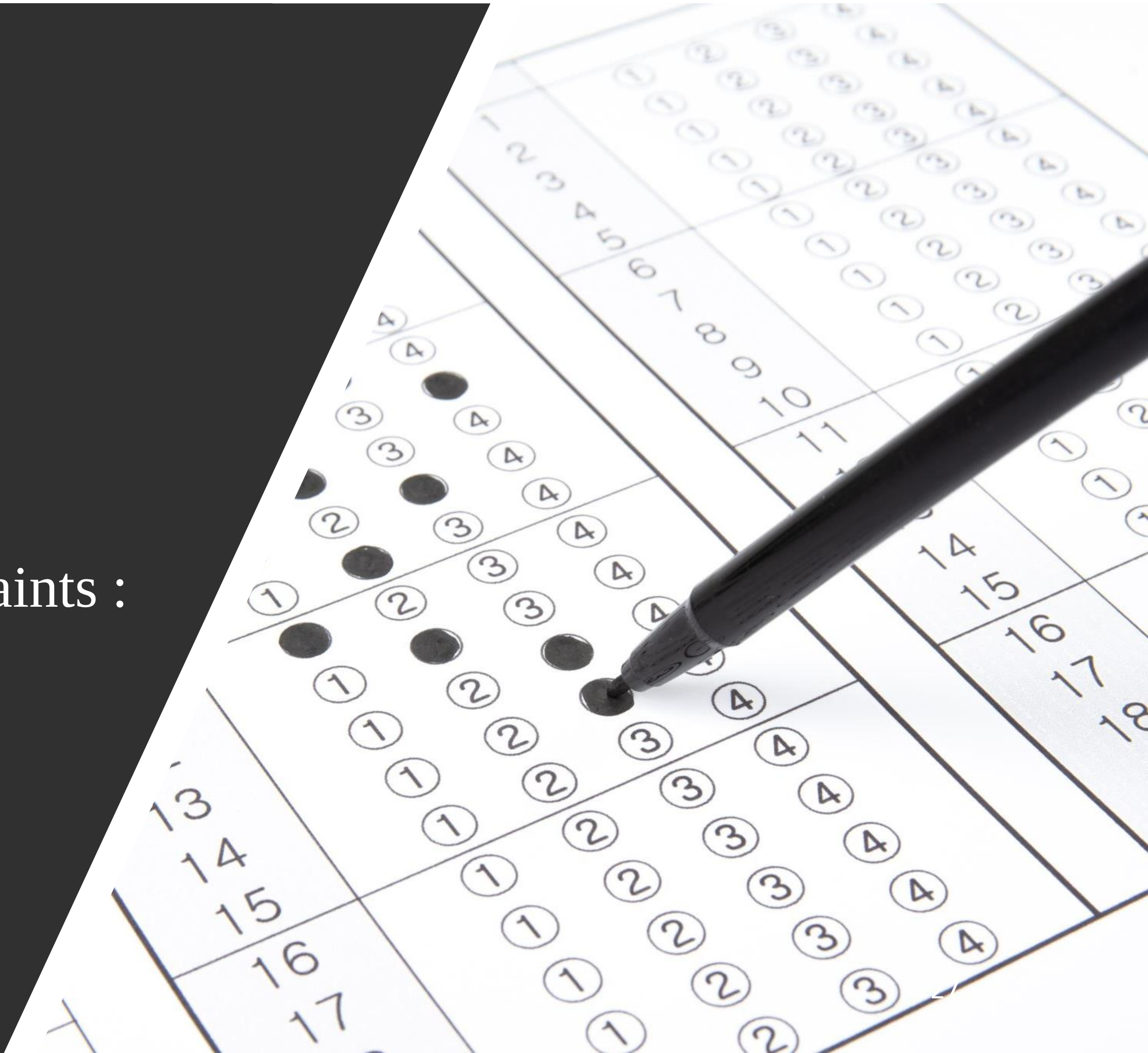


Sudoku as SMT Problem

- ☐ Totally 108 clauses

- ☐ One clause from fill-in-number constraints:
  - ➢ $9 \times 9 = 81$

- ☐ Three sets of clauses from Alldifferent constraints :
  - ➢ $3 \times 9 = 27$



Common Constraints in SAT/SMT Problem

- Equal Constraints

- Constraints are according to the puzzle given

- For example:
- There is already filled in number 4 in [4, 8]

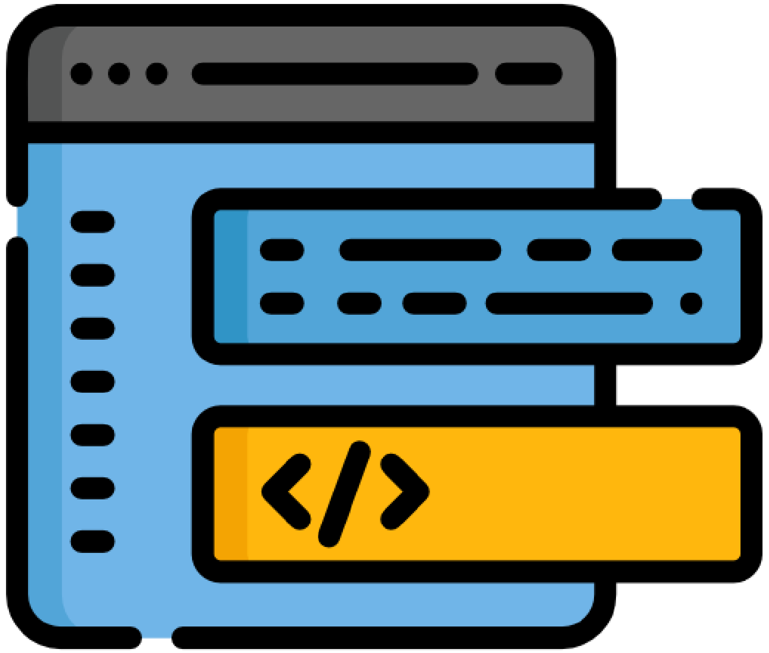
- In SAT Problems: $s_{484} = true$

- In SMT Problems: $s_{48} = 4$

Code Explanation

03

章節 PART



Backtracking

Function and Struct Defining

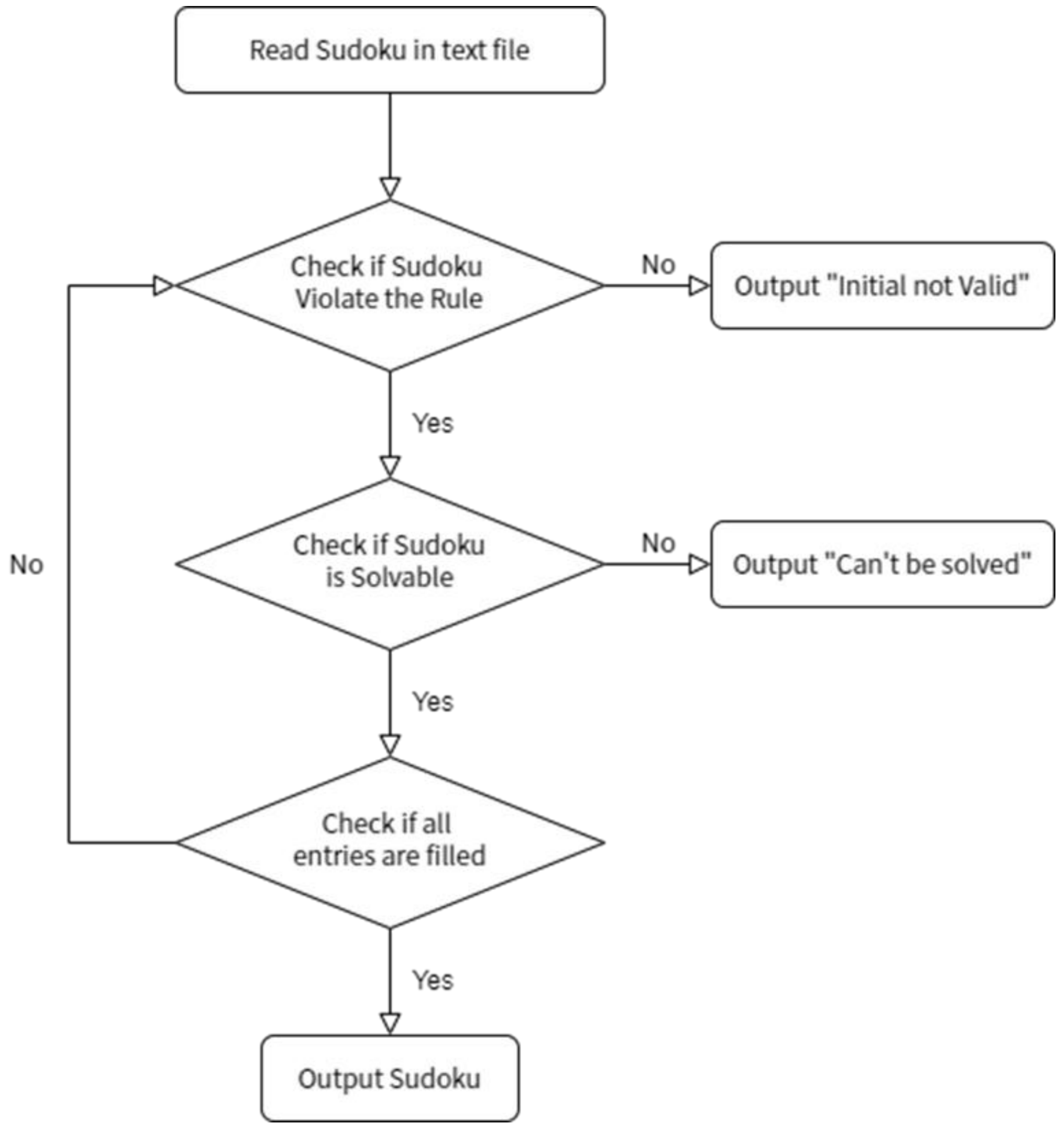
Struct

- ❑ EntryData:
  - ➢ Storing data of x, y coordinates and digits

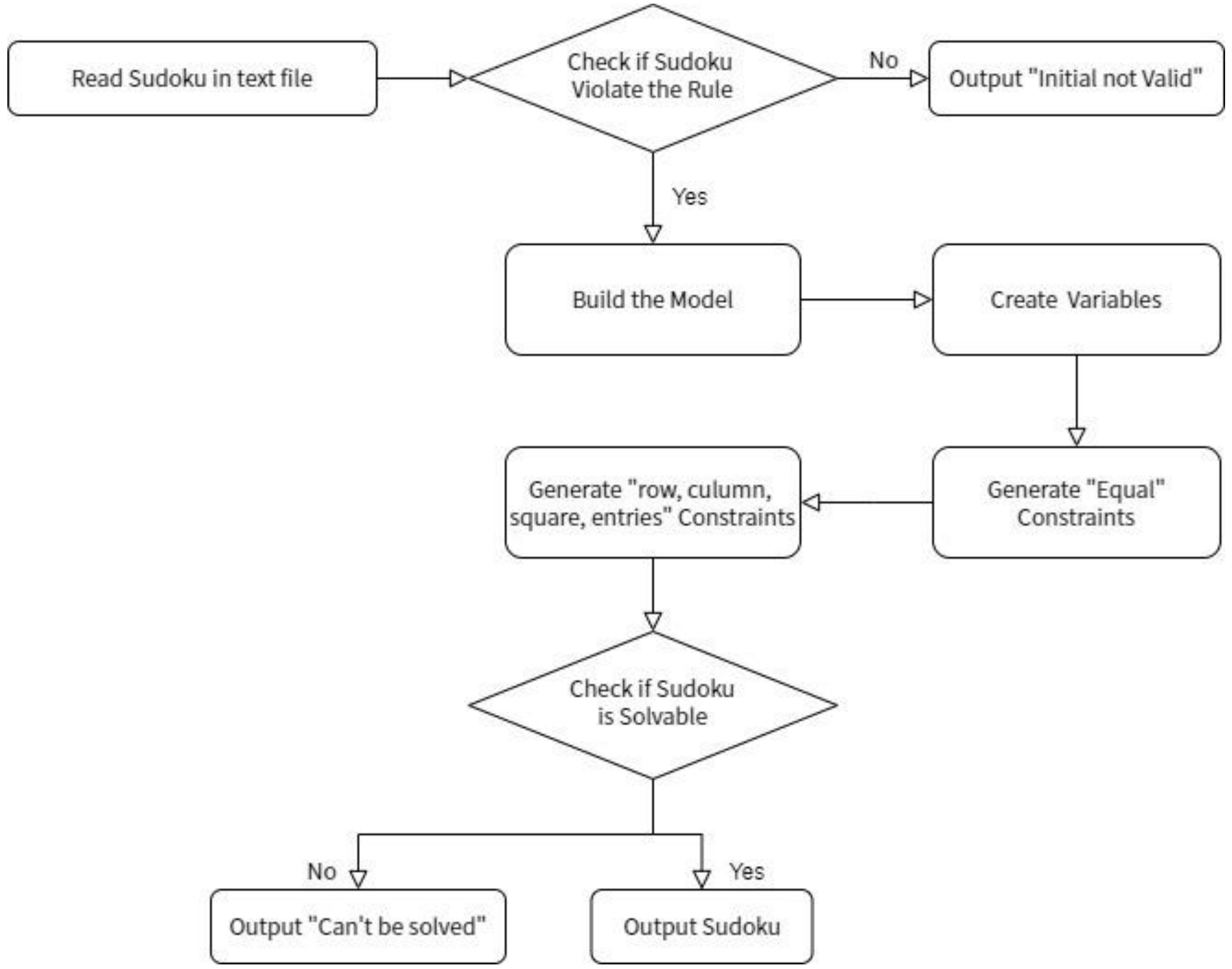
Functions

- ❑ fileInput: used to read the text file of the input
- ❑ reset: used to reset the Sudoku matrix
- ❑ print: output the 9×9 matrix of Sudoku
- ❑ notValid: Check the input is valid or not (check row, column and box)
- ❑ checkEmpty: Find blanks that haven’t filled
- ❑ checkPlacement: Check the entry can be filled
- ❑ solve: Check if we had found a solution.

Backtracking


Read Sudoku in text file
Check if Sudoku Violate the Rule
No
Output "Initial not Valid"
Yes
Check if Sudoku is Solvable
No
Output "Can't be solved"
Yes
Check if all entries are filled
No
Yes
Output Sudoku

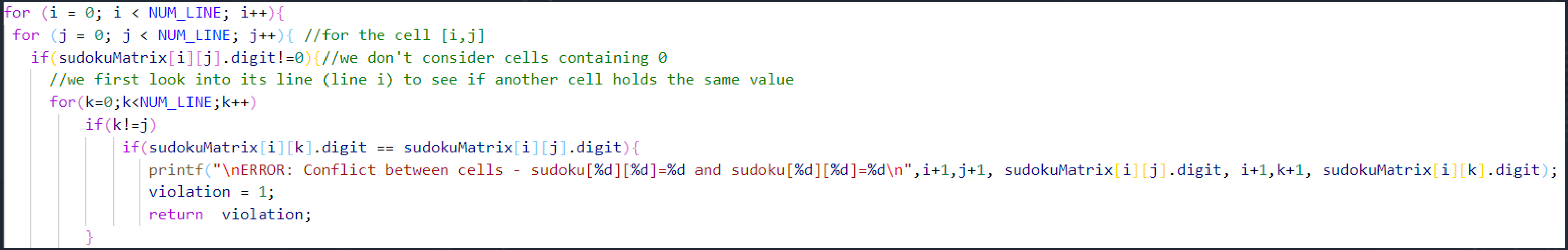
SAT/SMT


Read Sudoku in text file
Check if Sudoku Violate the Rule
No
Output "Initial not Valid"
Yes
Build the Model
Create Variables
Generate "Equal" Constraints
Generate "row, culumn, square, entries" Constraints
Check if Sudoku is Solvable
No
Yes
Output "Can't be solved"
Output Sudoku

Boolean Satisfiability Problem

Check Violation

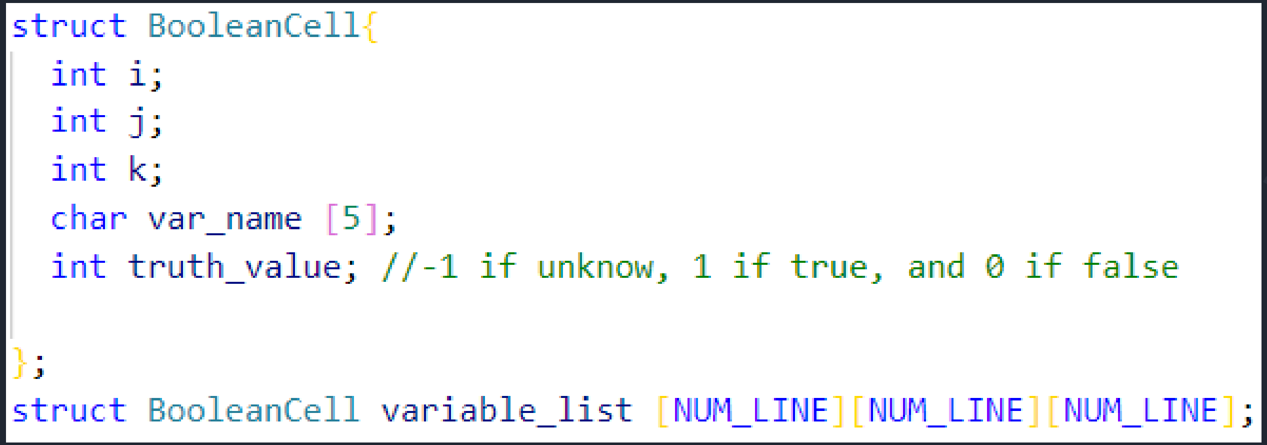
- Check its row to see if another cell holds the same value.
- Check its column to see if another cell holds the same value.
- Check each square of 9 digits to see if a value appears more than once.



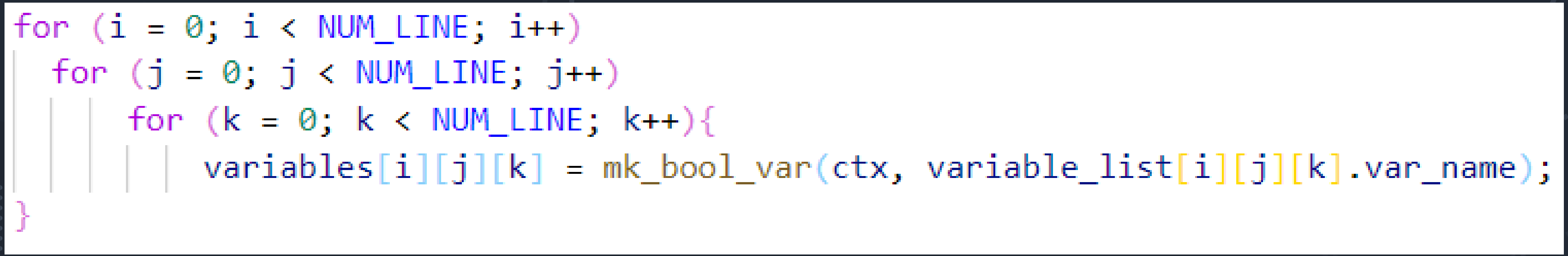
```c
for (i = 0; i < NUM_LINE; i++){
 for (j = 0; j < NUM_LINE; j++){ //for the cell [i,j]
   if(sudokuMatrix[i][j].digit!=0){//we don't consider cells containing 0
     //we first look into its line (line i) to see if another cell holds the same value
     for(k=0;k<NUM_LINE;k++)
         if(k!=j)
             if(sudokuMatrix[i][k].digit == sudokuMatrix[i][j].digit){
                printf("\nERROR: Conflict between cells - sudoku[%d][%d]=%d and sudoku[%d][%d]=%d\n",i+1,j+1, sudokuMatrix[i][j].digit, i+1,k+1, sudokuMatrix[i][k].digit);
                violation = 1;
                return  violation;
         }
```

Create Boolean Variable

- Create 9*9*9=729 Boolean variables



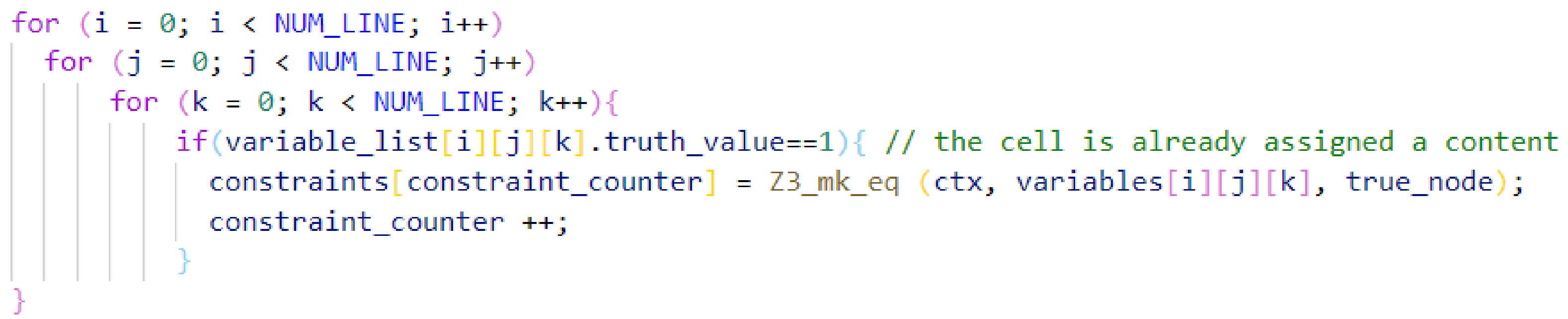
```c
struct BooleanCell{
  int i;
  int j;
  int k;
  char var_name [5];
  int truth_value; //-1 if unknow, 1 if true, and 0 if false

};
struct BooleanCell variable_list [NUM_LINE][NUM_LINE][NUM_LINE];
```



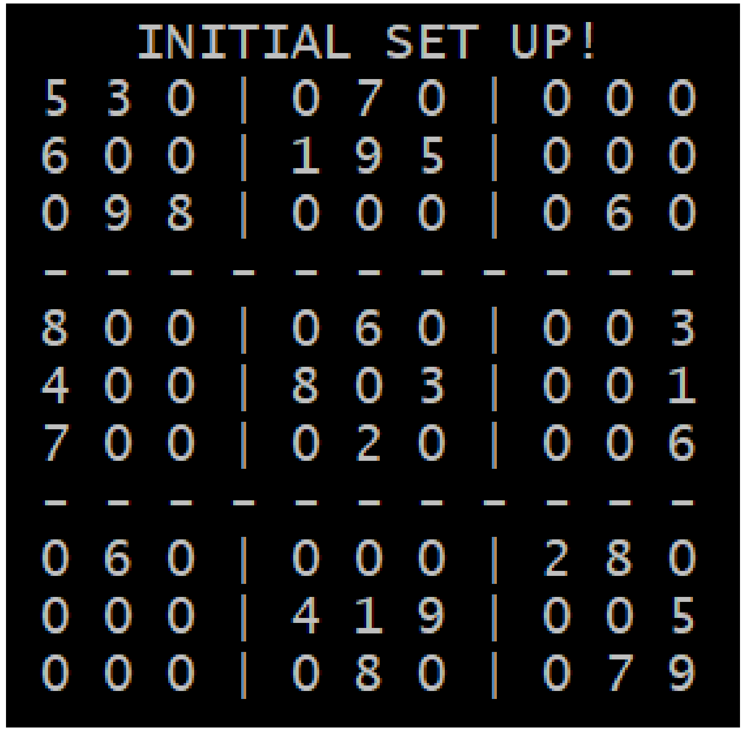
```c
for (i = 0; i < NUM_LINE; i++)
  for (j = 0; j < NUM_LINE; j++)
    for (k = 0; k < NUM_LINE; k++){
      variables[i][j][k] = mk_bool_var(ctx, variable_list[i][j][k].var_name);
}
```

Generate Constraints

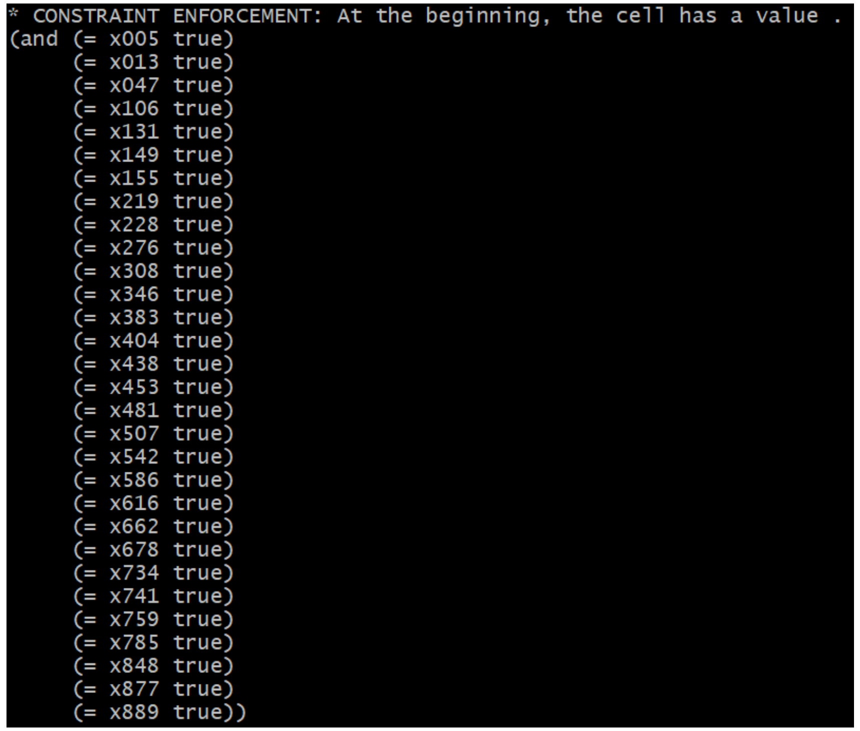
First Equal



```
for (i = 0; i < NUM_LINE; i++)
  for (j = 0; j < NUM_LINE; j++)
      for (k = 0; k < NUM_LINE; k++){
        if(variable_list[i][j][k].truth_value==1){ // the cell is already assigned a content
          constraints[constraint_counter] = Z3_mk_eq (ctx, variables[i][j][k], true_node);
          constraint_counter ++;
        }
}
```



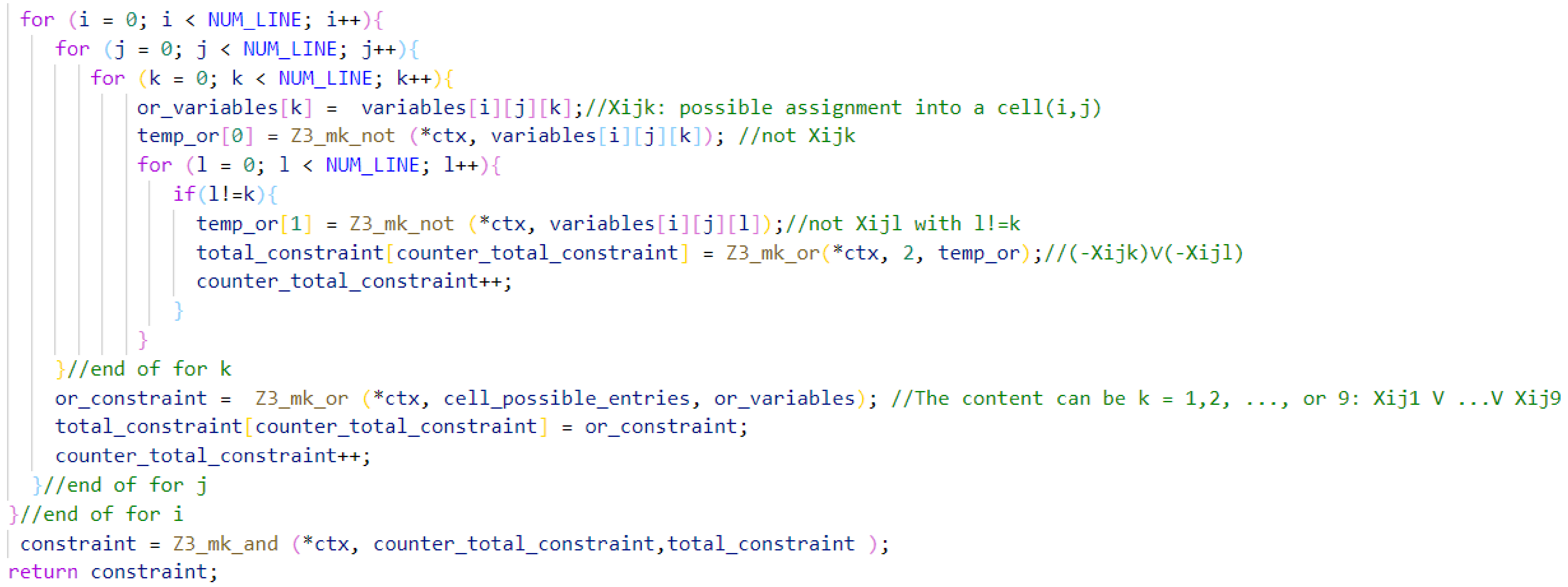
```
    INITIAL SET UP!
5 3 0 | 0 7 0 | 0 0 0
6 0 0 | 1 9 5 | 0 0 0
0 9 8 | 0 0 0 | 0 6 0
- - - - - - - - - - -
8 0 0 | 0 6 0 | 0 0 3
4 0 0 | 8 0 3 | 0 0 1
7 0 0 | 0 2 0 | 0 0 6
- - - - - - - - - - -
0 6 0 | 0 0 0 | 2 8 0
0 0 0 | 4 1 9 | 0 0 5
0 0 0 | 0 8 0 | 0 7 9
```



```
* CONSTRAINT ENFORCEMENT: At the beginning, the cell has a value .
(and (= x005 true)
     (= x013 true)
     (= x047 true)
     (= x106 true)
     (= x131 true)
     (= x149 true)
     (= x155 true)
     (= x219 true)
     (= x228 true)
     (= x276 true)
     (= x308 true)
     (= x346 true)
     (= x383 true)
     (= x404 true)
     (= x438 true)
     (= x453 true)
     (= x481 true)
     (= x507 true)
     (= x542 true)
     (= x586 true)
     (= x616 true)
     (= x662 true)
     (= x678 true)
     (= x734 true)
     (= x741 true)
     (= x759 true)
     (= x785 true)
     (= x848 true)
     (= x877 true)
     (= x889 true))
```

Generate Constraints

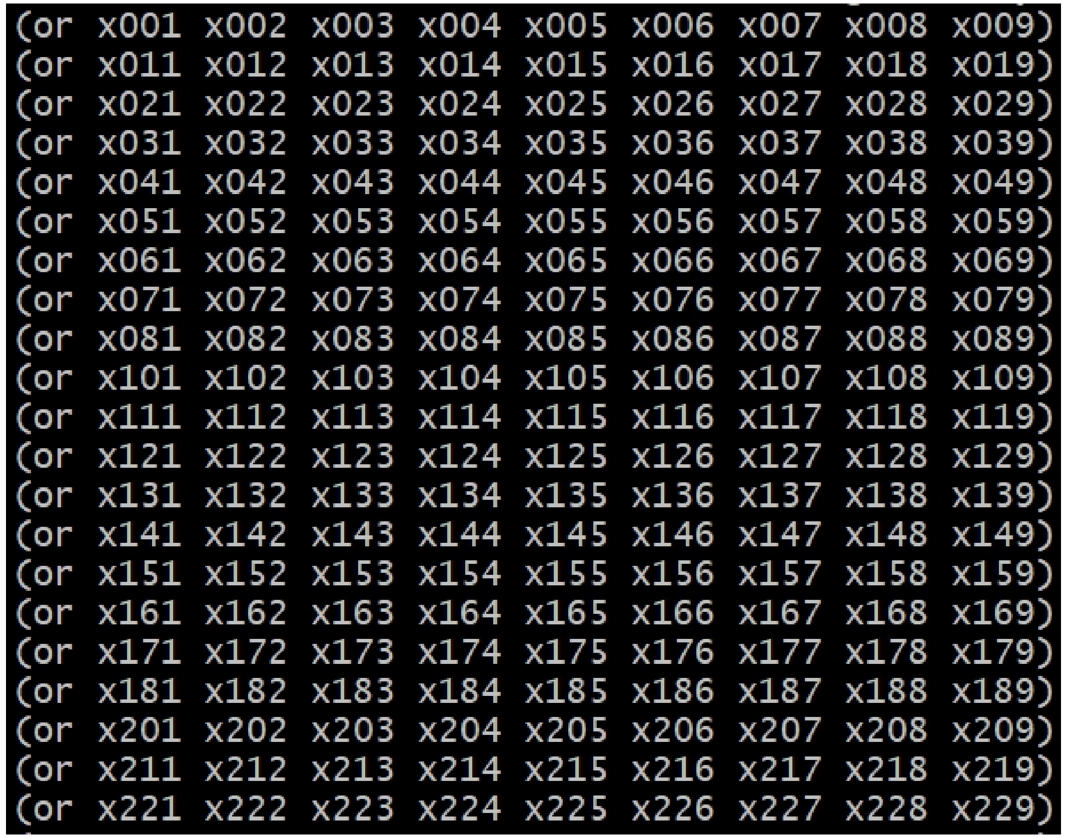
- Only One Label Per Cell



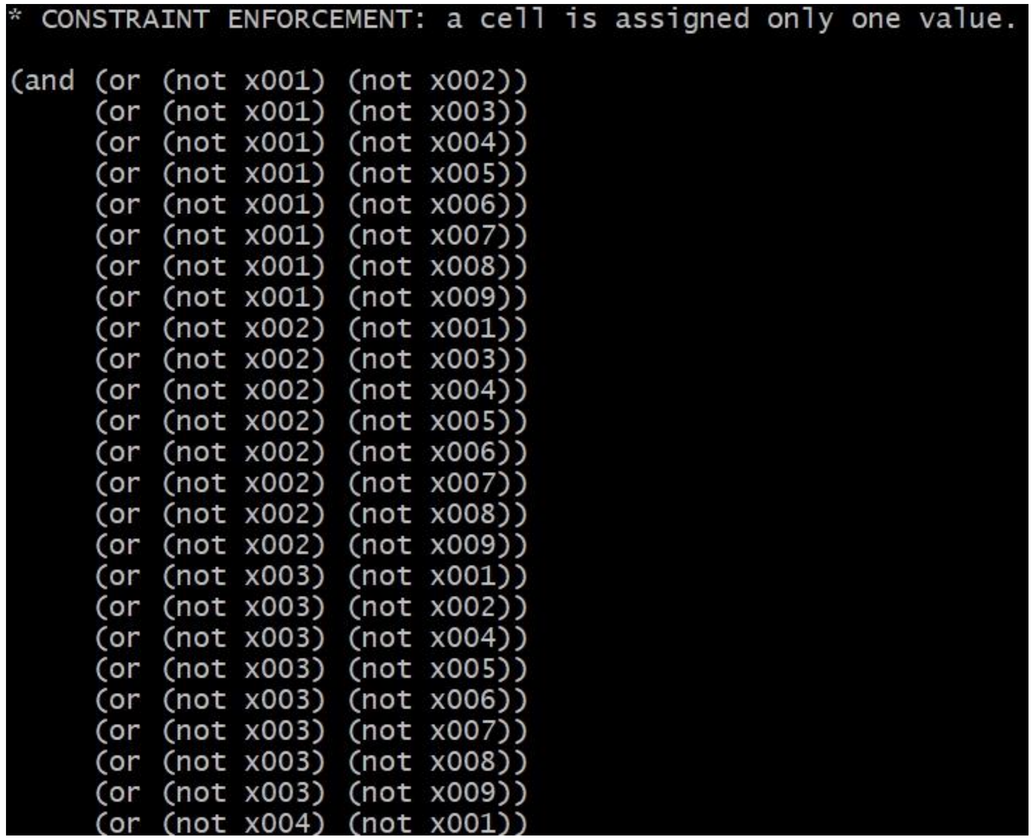
```
for (i = 0; i < NUM_LINE; i++){
    for (j = 0; j < NUM_LINE; j++){
        for (k = 0; k < NUM_LINE; k++){
            or_variables[k] =  variables[i][j][k];//Xijk: possible assignment into a cell(i,j)
            temp_or[0] = Z3_mk_not (*ctx, variables[i][j][k]); //not Xijk
            for (l = 0; l < NUM_LINE; l++){
                if(l!=k){
                    temp_or[1] = Z3_mk_not (*ctx, variables[i][j][l]);//not Xijl with l!=k
                    total_constraint[counter_total_constraint] = Z3_mk_or(*ctx, 2, temp_or);//(-Xijk)V(-Xijl)
                    counter_total_constraint++;
                }
            }
    }//end of for k
    or_constraint =  Z3_mk_or (*ctx, cell_possible_entries, or_variables); //The content can be k = 1,2, ..., or 9: Xij1 V ...V Xij9
    total_constraint[counter_total_constraint] = or_constraint;
    counter_total_constraint++;
  }//end of for j
}//end of for i
 constraint = Z3_mk_and (*ctx, counter_total_constraint,total_constraint );
return constraint;
```

Generate Constraints

- Only One Label Per Cell



```
(or x001 x002 x003 x004 x005 x006 x007 x008 x009)
(or x011 x012 x013 x014 x015 x016 x017 x018 x019)
(or x021 x022 x023 x024 x025 x026 x027 x028 x029)
(or x031 x032 x033 x034 x035 x036 x037 x038 x039)
(or x041 x042 x043 x044 x045 x046 x047 x048 x049)
(or x051 x052 x053 x054 x055 x056 x057 x058 x059)
(or x061 x062 x063 x064 x065 x066 x067 x068 x069)
(or x071 x072 x073 x074 x075 x076 x077 x078 x079)
(or x081 x082 x083 x084 x085 x086 x087 x088 x089)
(or x101 x102 x103 x104 x105 x106 x107 x108 x109)
(or x111 x112 x113 x114 x115 x116 x117 x118 x119)
(or x121 x122 x123 x124 x125 x126 x127 x128 x129)
(or x131 x132 x133 x134 x135 x136 x137 x138 x139)
(or x141 x142 x143 x144 x145 x146 x147 x148 x149)
(or x151 x152 x153 x154 x155 x156 x157 x158 x159)
(or x161 x162 x163 x164 x165 x166 x167 x168 x169)
(or x171 x172 x173 x174 x175 x176 x177 x178 x179)
(or x181 x182 x183 x184 x185 x186 x187 x188 x189)
(or x201 x202 x203 x204 x205 x206 x207 x208 x209)
(or x211 x212 x213 x214 x215 x216 x217 x218 x219)
(or x221 x222 x223 x224 x225 x226 x227 x228 x229)
```



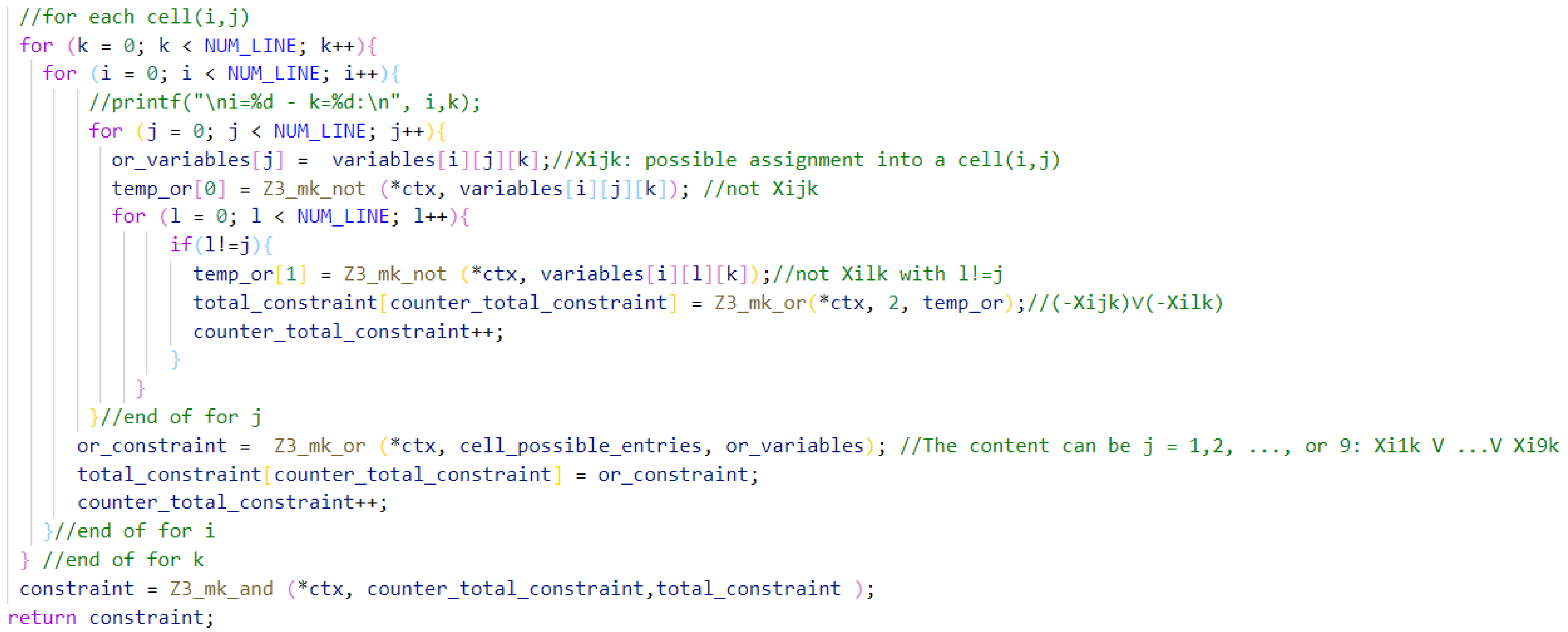
```
* CONSTRAINT ENFORCEMENT: a cell is assigned only one value.

(and (or (not x001) (not x002))
     (or (not x001) (not x003))
     (or (not x001) (not x004))
     (or (not x001) (not x005))
     (or (not x001) (not x006))
     (or (not x001) (not x007))
     (or (not x001) (not x008))
     (or (not x001) (not x009))
     (or (not x002) (not x001))
     (or (not x002) (not x003))
     (or (not x002) (not x004))
     (or (not x002) (not x005))
     (or (not x002) (not x006))
     (or (not x002) (not x007))
     (or (not x002) (not x008))
     (or (not x002) (not x009))
     (or (not x003) (not x001))
     (or (not x003) (not x002))
     (or (not x003) (not x004))
     (or (not x003) (not x005))
     (or (not x003) (not x006))
     (or (not x003) (not x007))
     (or (not x003) (not x008))
     (or (not x003) (not x009))
     (or (not x004) (not x001))
```

Generate Constraints

AND	$\boxed{-x001}$ ∨-x002	$\boxed{-x001}$ ∨-x003	$\boxed{-x001}$ ∨-x004	$\boxed{-x001}$ ∨-x005	$\boxed{-x001}$ ∨-x006	$\boxed{-x001}$ ∨-x007	$\boxed{-x001}$ ∨-x008	$\boxed{-x001}$ ∨-x009
	-x002 ∨$\boxed{-x001}$	-x002 ∨-x003	-x002 ∨-x004	-x002 ∨-x005	-x002 ∨-x006	-x002 ∨-x007	-x002 ∨-x008	-x002 ∨-x009
	-x003 ∨$\boxed{-x001}$	-x003 ∨-x002	-x003 ∨-x004	-x003 ∨-x005	-x003 ∨-x006	-x003 ∨-x007	-x003 ∨-x008	-x003 ∨-x009
	-x004 ∨$\boxed{-x001}$	-x004 ∨-x002	-x004 ∨-x003	-x004 ∨-x005	-x004 ∨-x006	-x004 ∨-x007	-x004 ∨-x008	-x004 ∨-x009
	-x005 ∨$\boxed{-x001}$	-x005 ∨-x002	-x005 ∨-x003	-x005 ∨-x004	-x005 ∨-x006	-x005 ∨-x007	-x005 ∨-x008	-x005 ∨-x009
	-x006 ∨$\boxed{-x001}$	-x006 ∨-x002	-x006 ∨-x003	-x006 ∨-x004	-x006 ∨-x005	-x006 ∨-x007	-x006 ∨-x008	-x006 ∨-x009
	-x007 ∨$\boxed{-x001}$	-x007 ∨-x002	-x007 ∨-x003	-x007 ∨-x004	-x007 ∨-x005	-x007 ∨-x006	-x007 ∨-x008	-x007 ∨-x009
	-x008 ∨$\boxed{-x001}$	-x008 ∨-x002	-x008 ∨-x003	-x008 ∨-x004	-x008 ∨-x005	-x008 ∨-x006	-x008 ∨-x008	-x008 ∨-x009
	-x009 ∨$\boxed{-x001}$	-x009 ∨-x002	-x009 ∨-x003	-x009 ∨-x004	-x009 ∨-x005	-x009 ∨-x006	-x009 ∨-x007	-x009 ∨-x008

Generate Constraints

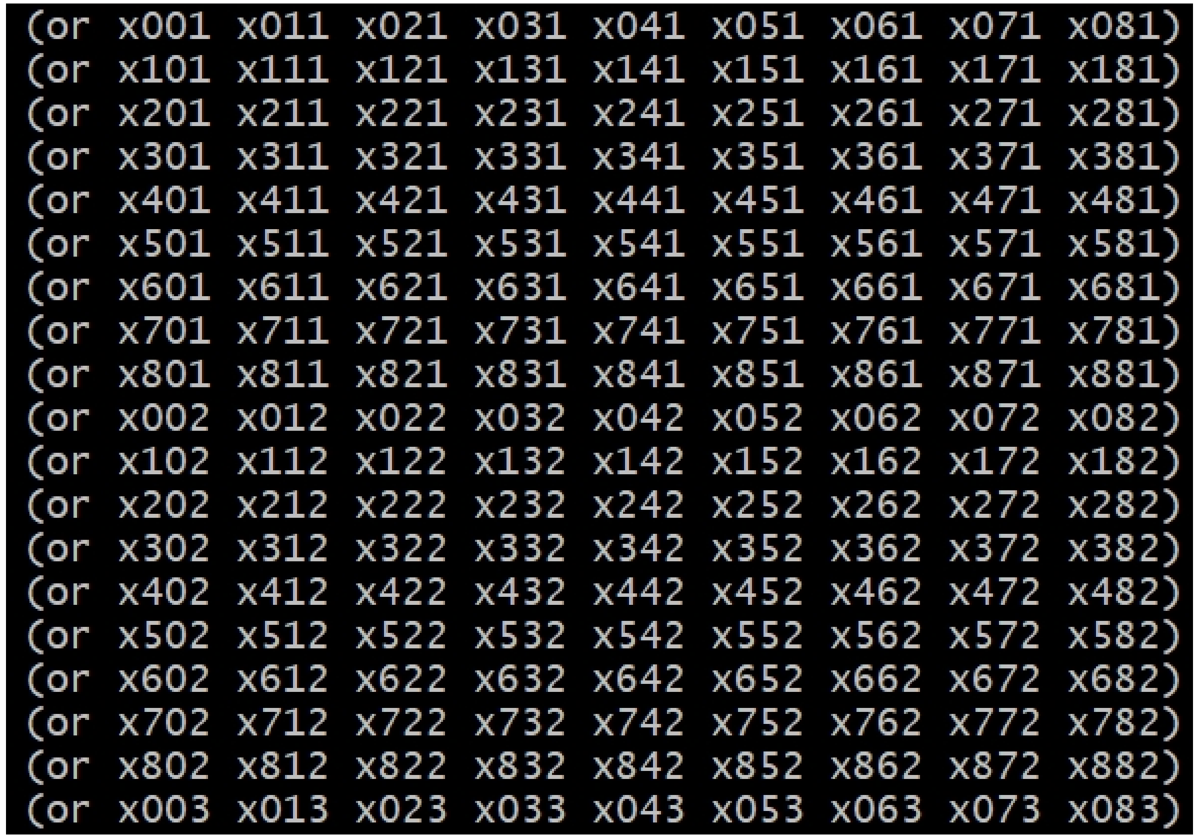
- Must Appear Once On Each Row



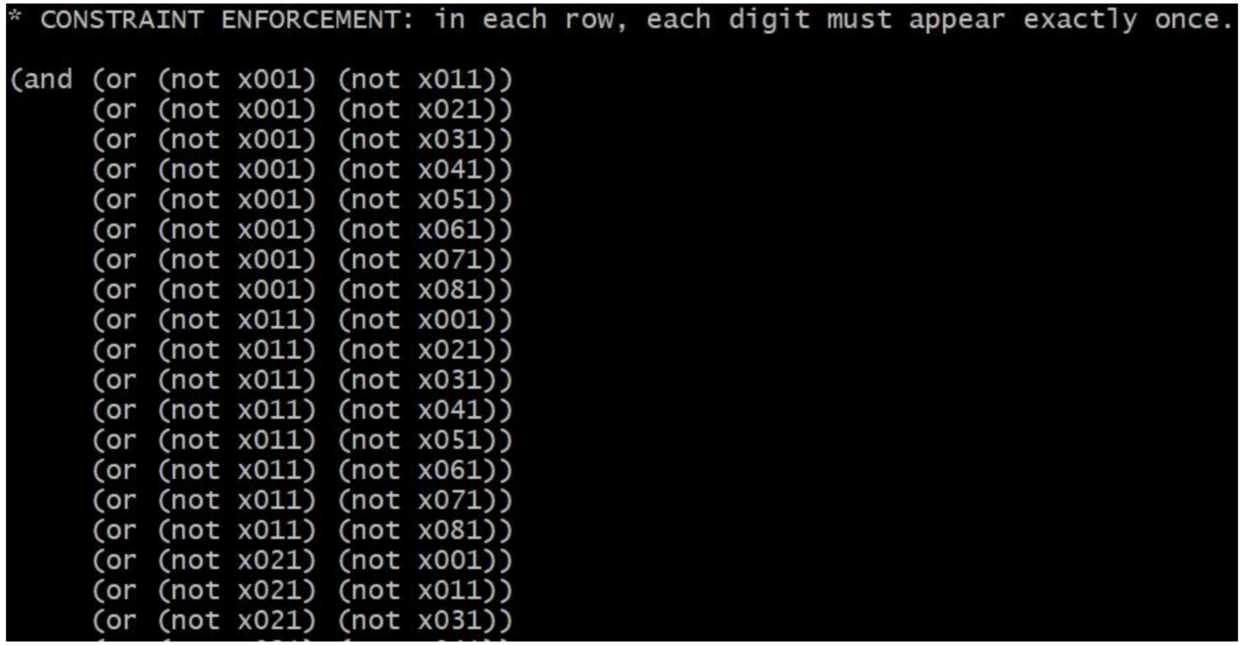
```
 //for each cell(i,j)
 for (k = 0; k < NUM_LINE; k++){
   for (i = 0; i < NUM_LINE; i++){
       //printf("\ni=%d - k=%d:\n", i,k);
       for (j = 0; j < NUM_LINE; j++){
         or_variables[j] =  variables[i][j][k];//Xijk: possible assignment into a cell(i,j)
         temp_or[0] = Z3_mk_not (*ctx, variables[i][j][k]); //not Xijk
         for (l = 0; l < NUM_LINE; l++){
              if(l!=j){
                temp_or[1] = Z3_mk_not (*ctx, variables[i][l][k]);//not Xilk with l!=j
                total_constraint[counter_total_constraint] = Z3_mk_or(*ctx, 2, temp_or);//(-Xijk)V(-Xilk)
                counter_total_constraint++;
              }
           }
       }//end of for j
      or_constraint =  Z3_mk_or (*ctx, cell_possible_entries, or_variables); //The content can be j = 1,2, ..., or 9: Xi1k V ...V Xi9k
      total_constraint[counter_total_constraint] = or_constraint;
      counter_total_constraint++;
   }//end of for i
 } //end of for k
 constraint = Z3_mk_and (*ctx, counter_total_constraint,total_constraint );
return constraint;
```

Generate Constraints

- Must Appear Once On Each Row



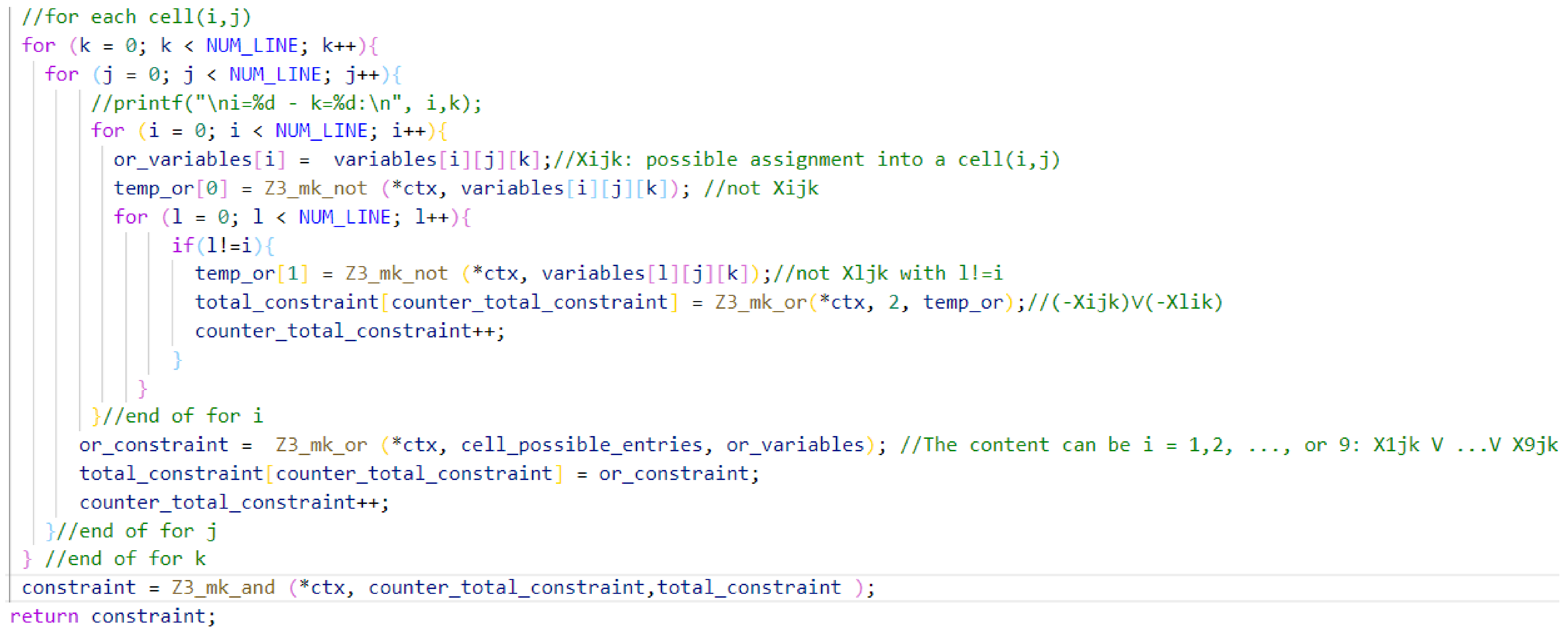
```
(or x001 x011 x021 x031 x041 x051 x061 x071 x081)
(or x101 x111 x121 x131 x141 x151 x161 x171 x181)
(or x201 x211 x221 x231 x241 x251 x261 x271 x281)
(or x301 x311 x321 x331 x341 x351 x361 x371 x381)
(or x401 x411 x421 x431 x441 x451 x461 x471 x481)
(or x501 x511 x521 x531 x541 x551 x561 x571 x581)
(or x601 x611 x621 x631 x641 x651 x661 x671 x681)
(or x701 x711 x721 x731 x741 x751 x761 x771 x781)
(or x801 x811 x821 x831 x841 x851 x861 x871 x881)
(or x002 x012 x022 x032 x042 x052 x062 x072 x082)
(or x102 x112 x122 x132 x142 x152 x162 x172 x182)
(or x202 x212 x222 x232 x242 x252 x262 x272 x282)
(or x302 x312 x322 x332 x342 x352 x362 x372 x382)
(or x402 x412 x422 x432 x442 x452 x462 x472 x482)
(or x502 x512 x522 x532 x542 x552 x562 x572 x582)
(or x602 x612 x622 x632 x642 x652 x662 x672 x682)
(or x702 x712 x722 x732 x742 x752 x762 x772 x782)
(or x802 x812 x822 x832 x842 x852 x862 x872 x882)
(or x003 x013 x023 x033 x043 x053 x063 x073 x083)
```



```
* CONSTRAINT ENFORCEMENT: in each row, each digit must appear exactly once.

(and (or (not x001) (not x011))
     (or (not x001) (not x021))
     (or (not x001) (not x031))
     (or (not x001) (not x041))
     (or (not x001) (not x051))
     (or (not x001) (not x061))
     (or (not x001) (not x071))
     (or (not x001) (not x081))
     (or (not x011) (not x001))
     (or (not x011) (not x021))
     (or (not x011) (not x031))
     (or (not x011) (not x041))
     (or (not x011) (not x051))
     (or (not x011) (not x061))
     (or (not x011) (not x071))
     (or (not x011) (not x081))
     (or (not x021) (not x001))
     (or (not x021) (not x011))
     (or (not x021) (not x031))
```

Generate Constraints

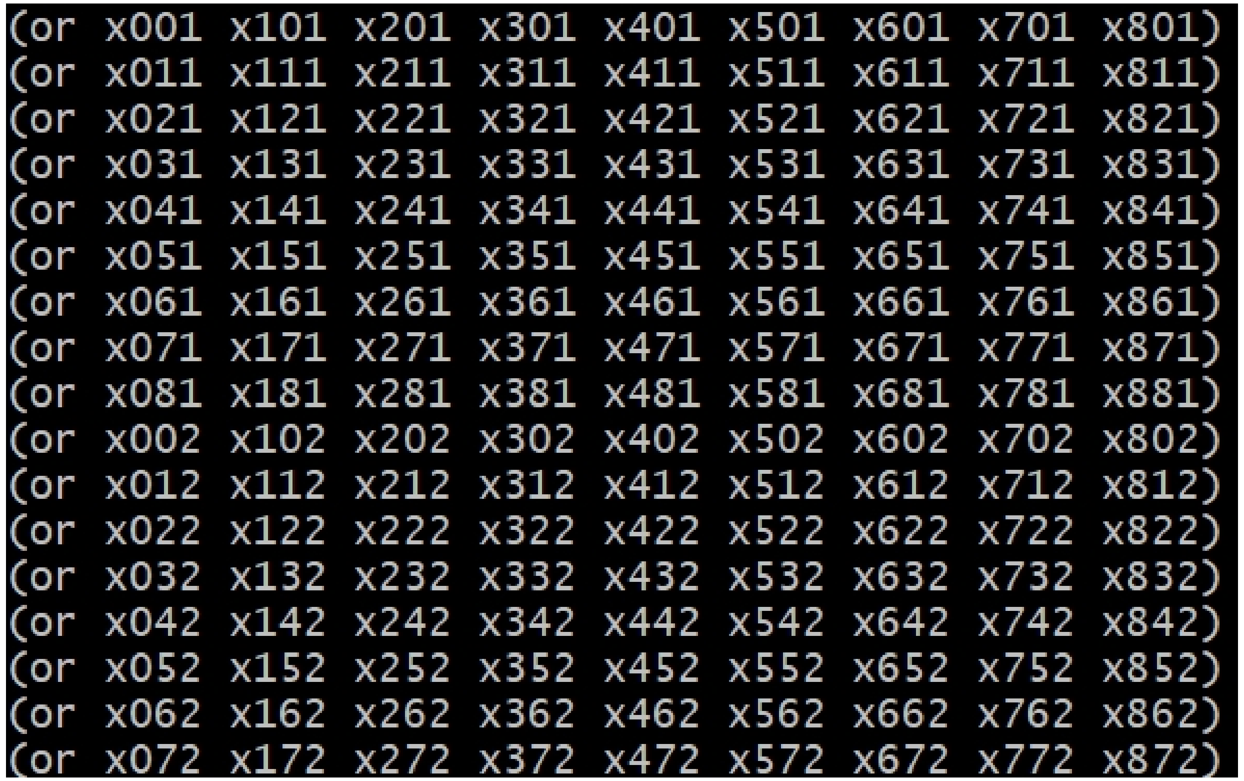
- Must Appear Once On Each Column



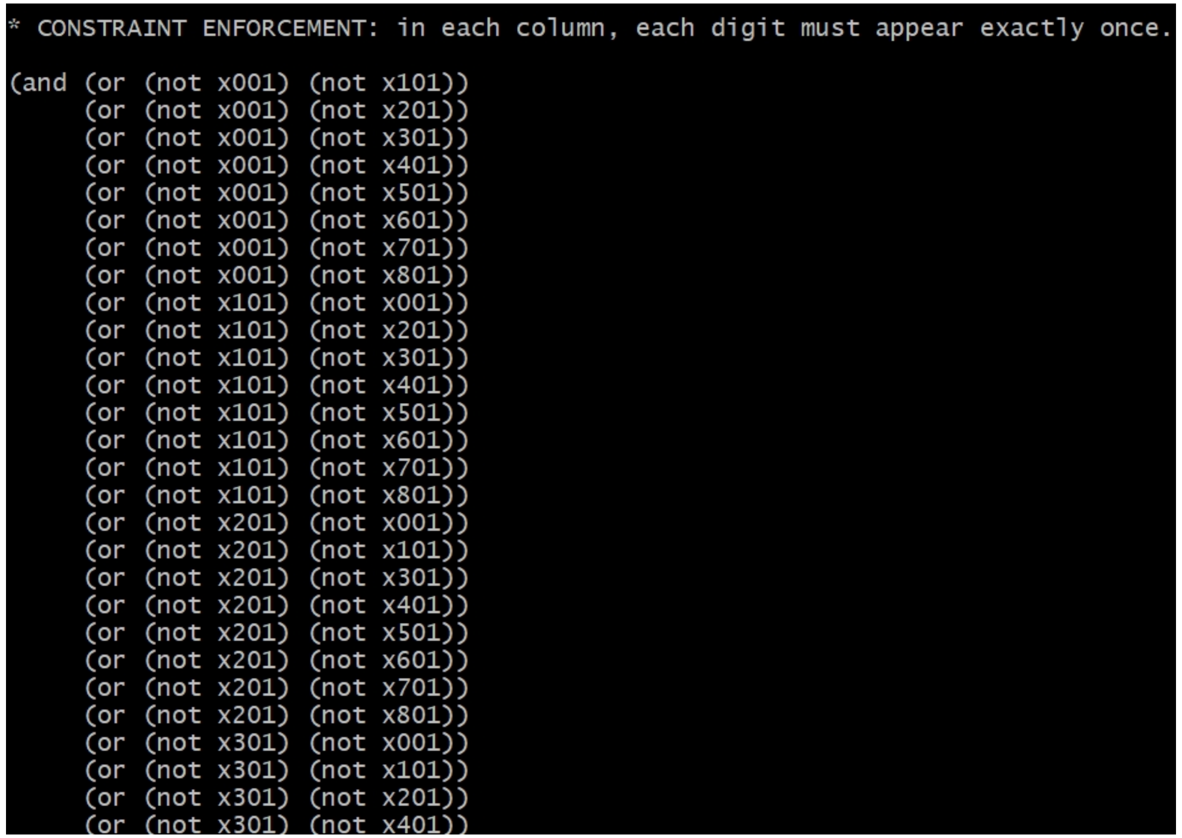
```
//for each cell(i,j)
for (k = 0; k < NUM_LINE; k++){
 for (j = 0; j < NUM_LINE; j++){
   //printf("\ni=%d - k=%d:\n", i,k);
   for (i = 0; i < NUM_LINE; i++){
    or_variables[i] =  variables[i][j][k];//Xijk: possible assignment into a cell(i,j)
    temp_or[0] = Z3_mk_not (*ctx, variables[i][j][k]); //not Xijk
    for (l = 0; l < NUM_LINE; l++){
      if(l!=i){
       temp_or[1] = Z3_mk_not (*ctx, variables[l][j][k]);//not Xljk with l!=i
       total_constraint[counter_total_constraint] = Z3_mk_or(*ctx, 2, temp_or);//(-Xijk)V(-Xlik)
       counter_total_constraint++;
      }
     }
   }//end of for i
  or_constraint =  Z3_mk_or (*ctx, cell_possible_entries, or_variables); //The content can be i = 1,2, ..., or 9: X1jk V ...V X9jk
  total_constraint[counter_total_constraint] = or_constraint;
  counter_total_constraint++;
 }//end of for j
} //end of for k
constraint = Z3_mk_and (*ctx, counter_total_constraint,total_constraint );
return constraint;
```

Generate Constraints

- Must Appear Once On Each Column



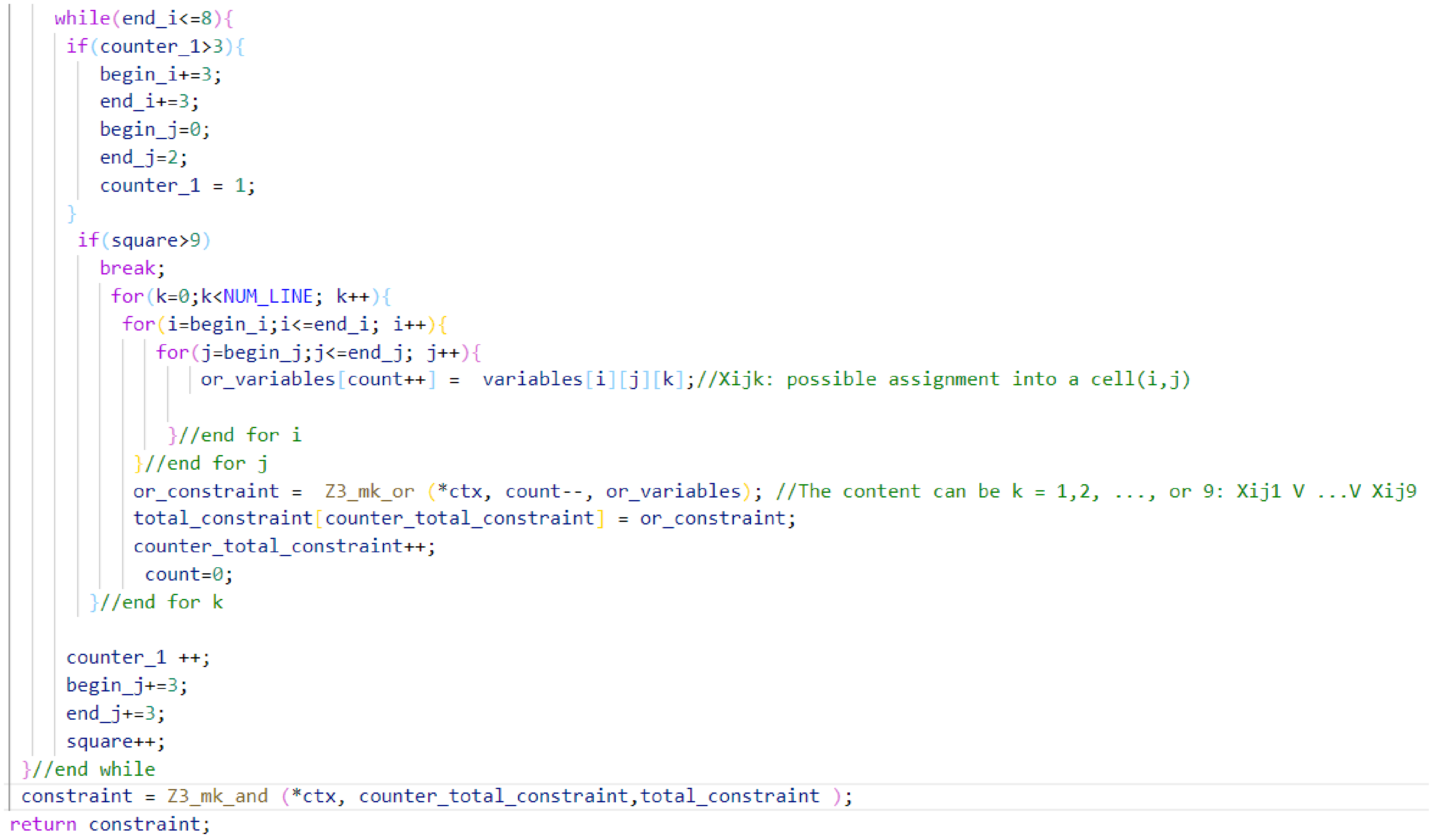
```
(or x001 x101 x201 x301 x401 x501 x601 x701 x801)
(or x011 x111 x211 x311 x411 x511 x611 x711 x811)
(or x021 x121 x221 x321 x421 x521 x621 x721 x821)
(or x031 x131 x231 x331 x431 x531 x631 x731 x831)
(or x041 x141 x241 x341 x441 x541 x641 x741 x841)
(or x051 x151 x251 x351 x451 x551 x651 x751 x851)
(or x061 x161 x261 x361 x461 x561 x661 x761 x861)
(or x071 x171 x271 x371 x471 x571 x671 x771 x871)
(or x081 x181 x281 x381 x481 x581 x681 x781 x881)
(or x002 x102 x202 x302 x402 x502 x602 x702 x802)
(or x012 x112 x212 x312 x412 x512 x612 x712 x812)
(or x022 x122 x222 x322 x422 x522 x622 x722 x822)
(or x032 x132 x232 x332 x432 x532 x632 x732 x832)
(or x042 x142 x242 x342 x442 x542 x642 x742 x842)
(or x052 x152 x252 x352 x452 x552 x652 x752 x852)
(or x062 x162 x262 x362 x462 x562 x662 x762 x862)
(or x072 x172 x272 x372 x472 x572 x672 x772 x872)
```



```
* CONSTRAINT ENFORCEMENT: in each column, each digit must appear exactly once.

(and (or (not x001) (not x101))
     (or (not x001) (not x201))
     (or (not x001) (not x301))
     (or (not x001) (not x401))
     (or (not x001) (not x501))
     (or (not x001) (not x601))
     (or (not x001) (not x701))
     (or (not x001) (not x801))
     (or (not x101) (not x001))
     (or (not x101) (not x201))
     (or (not x101) (not x301))
     (or (not x101) (not x401))
     (or (not x101) (not x501))
     (or (not x101) (not x601))
     (or (not x101) (not x701))
     (or (not x101) (not x801))
     (or (not x201) (not x001))
     (or (not x201) (not x101))
     (or (not x201) (not x301))
     (or (not x201) (not x401))
     (or (not x201) (not x501))
     (or (not x201) (not x601))
     (or (not x201) (not x701))
     (or (not x201) (not x801))
     (or (not x301) (not x001))
     (or (not x301) (not x101))
     (or (not x301) (not x201))
     (or (not x301) (not x401))
```

Generate Constraints

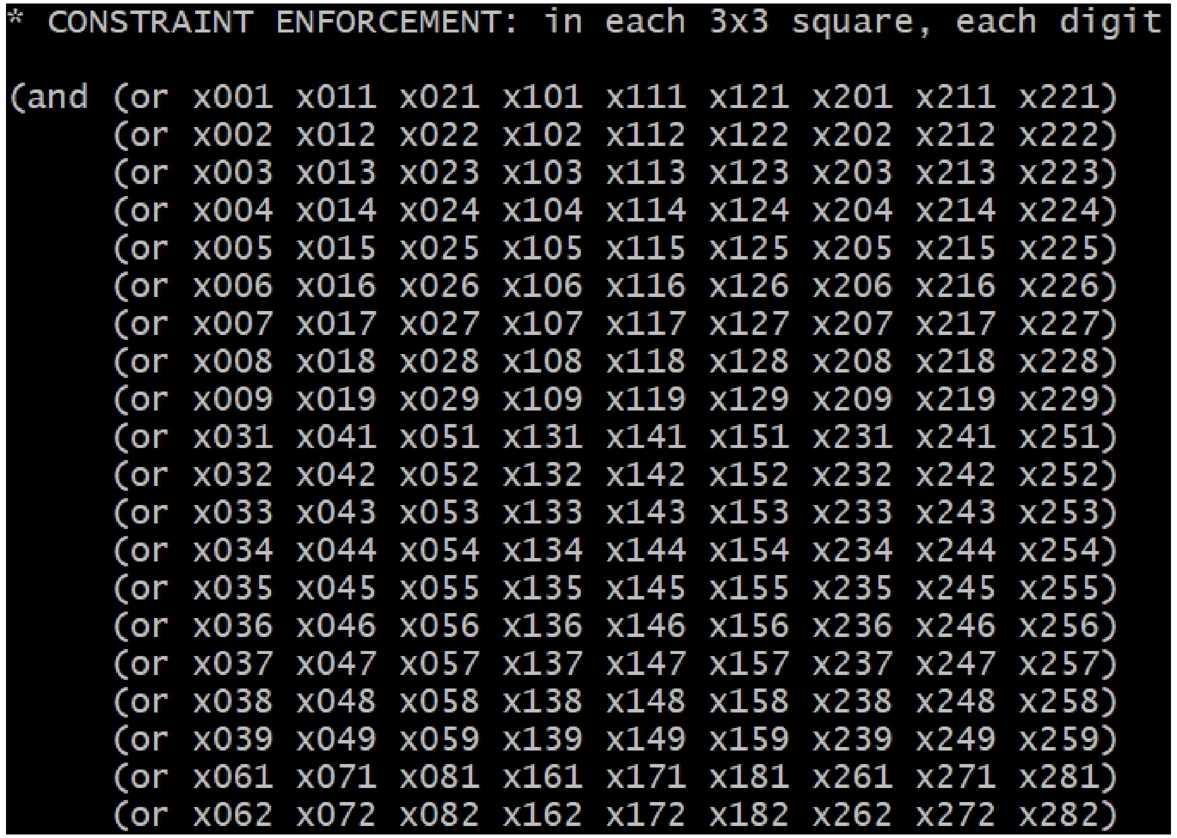
- Only one label per square



```
  while(end_i<=8){
   if(counter_1>3){
      begin_i+=3;
      end_i+=3;
      begin_j=0;
      end_j=2;
      counter_1 = 1;
   }
    if(square>9)
      break;
       for(k=0;k<NUM_LINE; k++){
        for(i=begin_i;i<=end_i; i++){
           for(j=begin_j;j<=end_j; j++){
               or_variables[count++] =  variables[i][j][k];//Xijk: possible assignment into a cell(i,j)

           }//end for i
         }//end for j
         or_constraint =  Z3_mk_or (*ctx, count--, or_variables); //The content can be k = 1,2, ..., or 9: Xij1 V ...V Xij9
         total_constraint[counter_total_constraint] = or_constraint;
         counter_total_constraint++;
          count=0;
     }//end for k

   counter_1 ++;
   begin_j+=3;
   end_j+=3;
   square++;
}//end while
constraint = Z3_mk_and (*ctx, counter_total_constraint,total_constraint );
return constraint;
```

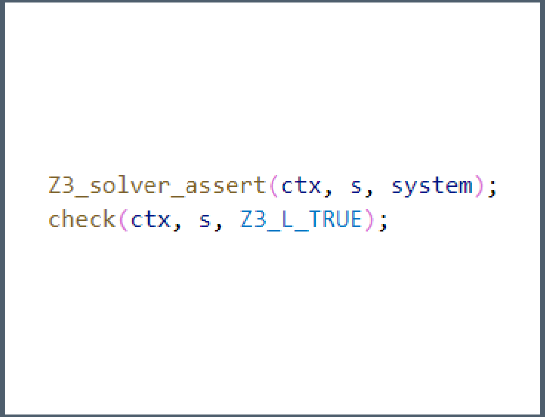
Generate Constraints

- Only one label per square

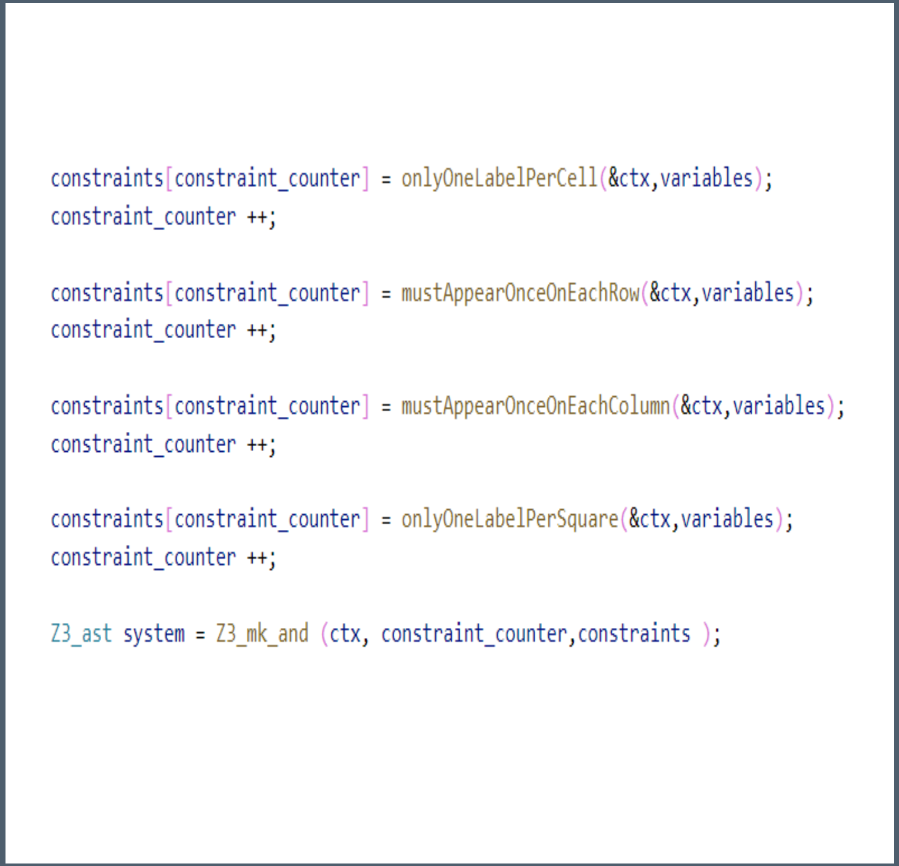


```
* CONSTRAINT ENFORCEMENT: in each 3x3 square, each digit

(and (or x001 x011 x021 x101 x111 x121 x201 x211 x221)
     (or x002 x012 x022 x102 x112 x122 x202 x212 x222)
     (or x003 x013 x023 x103 x113 x123 x203 x213 x223)
     (or x004 x014 x024 x104 x114 x124 x204 x214 x224)
     (or x005 x015 x025 x105 x115 x125 x205 x215 x225)
     (or x006 x016 x026 x106 x116 x126 x206 x216 x226)
     (or x007 x017 x027 x107 x117 x127 x207 x217 x227)
     (or x008 x018 x028 x108 x118 x128 x208 x218 x228)
     (or x009 x019 x029 x109 x119 x129 x209 x219 x229)
     (or x031 x041 x051 x131 x141 x151 x231 x241 x251)
     (or x032 x042 x052 x132 x142 x152 x232 x242 x252)
     (or x033 x043 x053 x133 x143 x153 x233 x243 x253)
     (or x034 x044 x054 x134 x144 x154 x234 x244 x254)
     (or x035 x045 x055 x135 x145 x155 x235 x245 x255)
     (or x036 x046 x056 x136 x146 x156 x236 x246 x256)
     (or x037 x047 x057 x137 x147 x157 x237 x247 x257)
     (or x038 x048 x058 x138 x148 x158 x238 x248 x258)
     (or x039 x049 x059 x139 x149 x159 x239 x249 x259)
     (or x061 x071 x081 x161 x171 x181 x261 x271 x281)
     (or x062 x072 x082 x162 x172 x182 x262 x272 x282)
```

Generate Constraints



```
Z3_solver_assert(ctx, s, system);
check(ctx, s, Z3_L_TRUE);
```

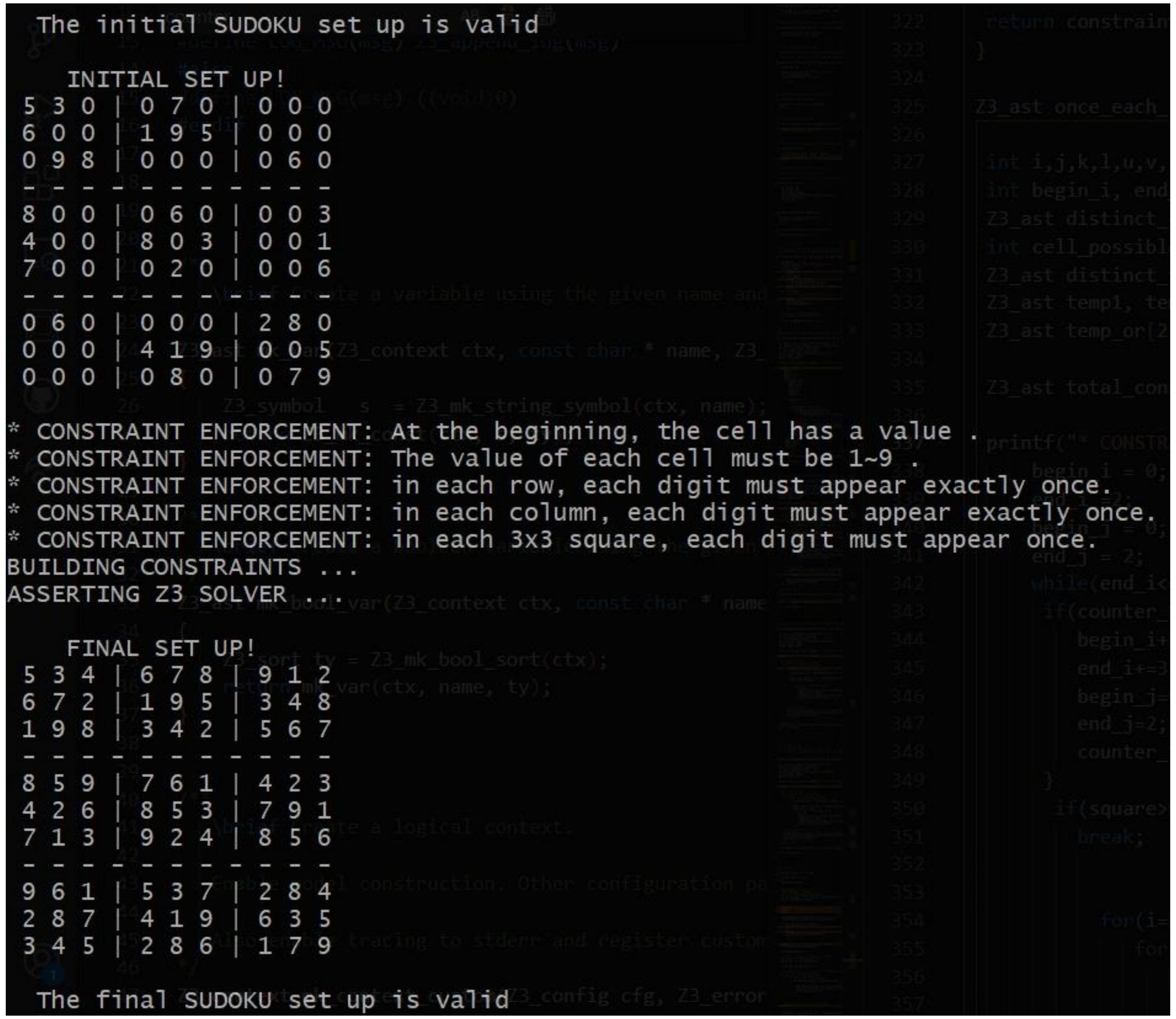


```
constraints[constraint_counter] = onlyOneLabelPerCell(&ctx,variables);
constraint_counter ++;

constraints[constraint_counter] = mustAppearOnceOnEachRow(&ctx,variables);
constraint_counter ++;

constraints[constraint_counter] = mustAppearOnceOnEachColumn(&ctx,variables);
constraint_counter ++;

constraints[constraint_counter] = onlyOneLabelPerSquare(&ctx,variables);
constraint_counter ++;

Z3_ast system = Z3_mk_and (ctx, constraint_counter,constraints );
```

Check if Sudoku is Solvable

```c
void check(Z3_context ctx, Z3_solver s, Z3_lbool expected_result)
{
    Z3_model m      = 0;
    Z3_lbool result = Z3_solver_check(ctx, s);
    switch (result) {
    case Z3_L_FALSE:
        printf("\nThe puzzle in unsolvable\n");
        break;
    case Z3_L_UNDEF:
        printf("unknown\n");
        m = Z3_solver_get_model(ctx, s);
        if (m) Z3_model_inc_ref(ctx, m);
        printf("potential model:\n%s\n", Z3_model_to_string(ctx, m));
        break;
    case Z3_L_TRUE:
        /*m = Z3_solver_get_model(ctx, s);
        if (m) Z3_model_inc_ref(ctx, m);
        printf("sat\n%s\n", Z3_model_to_string(ctx, m));*/
        dumpModel(&ctx,s);
        publishZ3SudokuResult();

        break;
    }
    if (result != expected_result) {
        exitf("unexpected result");
    }
    if (m) Z3_model_dec_ref(ctx, m);
}
```

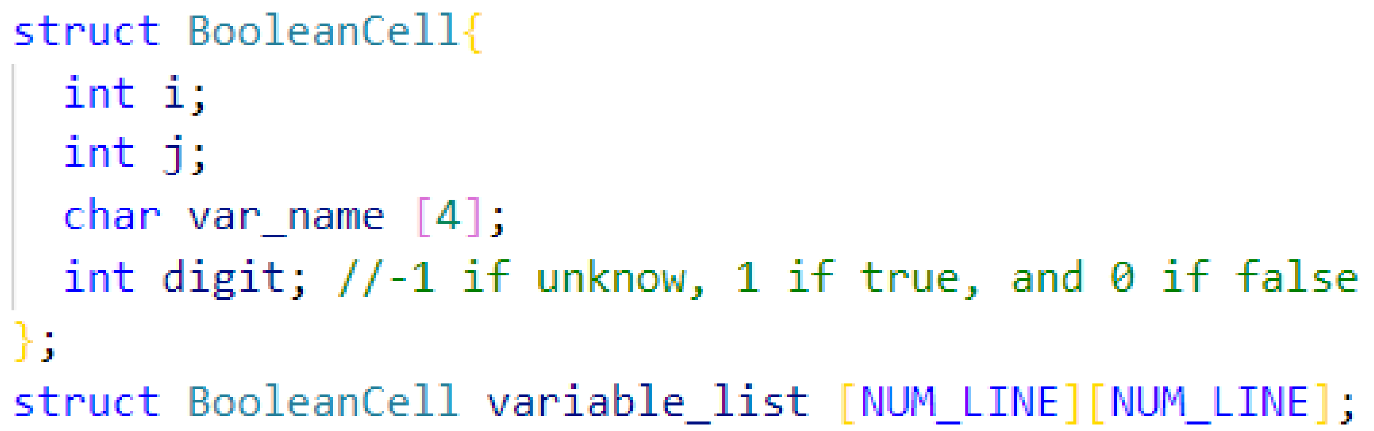
Solvable Result



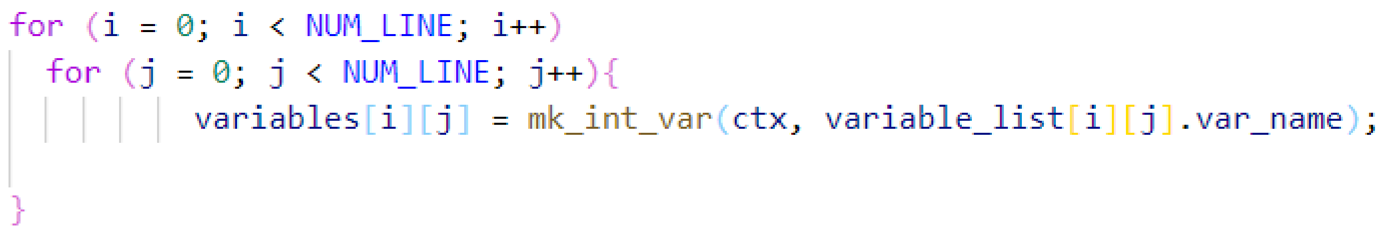
```
  The initial SUDOKU set up is valid

    INITIAL SET UP!
 5 3 0 | 0 7 0 | 0 0 0
 6 0 0 | 1 9 5 | 0 0 0
 0 9 8 | 0 0 0 | 0 6 0
 - - - - - - - - - - -
 8 0 0 | 0 6 0 | 0 0 3
 4 0 0 | 8 0 3 | 0 0 1
 7 0 0 | 0 2 0 | 0 0 6
 - - - - - - - - - - -
 0 6 0 | 0 0 0 | 2 8 0
 0 0 0 | 4 1 9 | 0 0 5
 0 0 0 | 0 8 0 | 0 7 9

* CONSTRAINT ENFORCEMENT: At the beginning, the cell has a value .
* CONSTRAINT ENFORCEMENT: The value of each cell must be 1~9 .
* CONSTRAINT ENFORCEMENT: in each row, each digit must appear exactly once.
* CONSTRAINT ENFORCEMENT: in each column, each digit must appear exactly once.
* CONSTRAINT ENFORCEMENT: in each 3x3 square, each digit must appear once.
BUILDING CONSTRAINTS ...
ASSERTING Z3 SOLVER ...

    FINAL SET UP!
 5 3 4 | 6 7 8 | 9 1 2
 6 7 2 | 1 9 5 | 3 4 8
 1 9 8 | 3 4 2 | 5 6 7
 - - - - - - - - - - -
 8 5 9 | 7 6 1 | 4 2 3
 4 2 6 | 8 5 3 | 7 9 1
 7 1 3 | 9 2 4 | 8 5 6
 - - - - - - - - - - -
 9 6 1 | 5 3 7 | 2 8 4
 2 8 7 | 4 1 9 | 6 3 5
 3 4 5 | 2 8 6 | 1 7 9

  The final SUDOKU set up is valid
```

Satisfiability Modulo Theories

Create Integer Variable



```
struct BooleanCell{
    int i;
    int j;
    char var_name [4];
    int digit; //-1 if unknow, 1 if true, and 0 if false
};
struct BooleanCell variable_list [NUM_LINE][NUM_LINE];
```



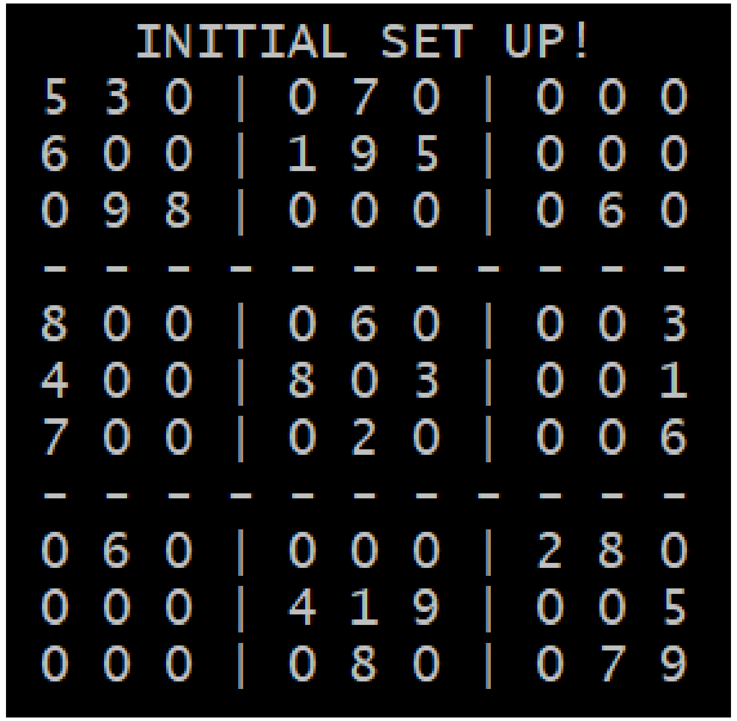
```
for (i = 0; i < NUM_LINE; i++)
  for (j = 0; j < NUM_LINE; j++){
          variables[i][j] = mk_int_var(ctx, variable_list[i][j].var_name);

}
```

- Create 9*9 = 81 integer variables

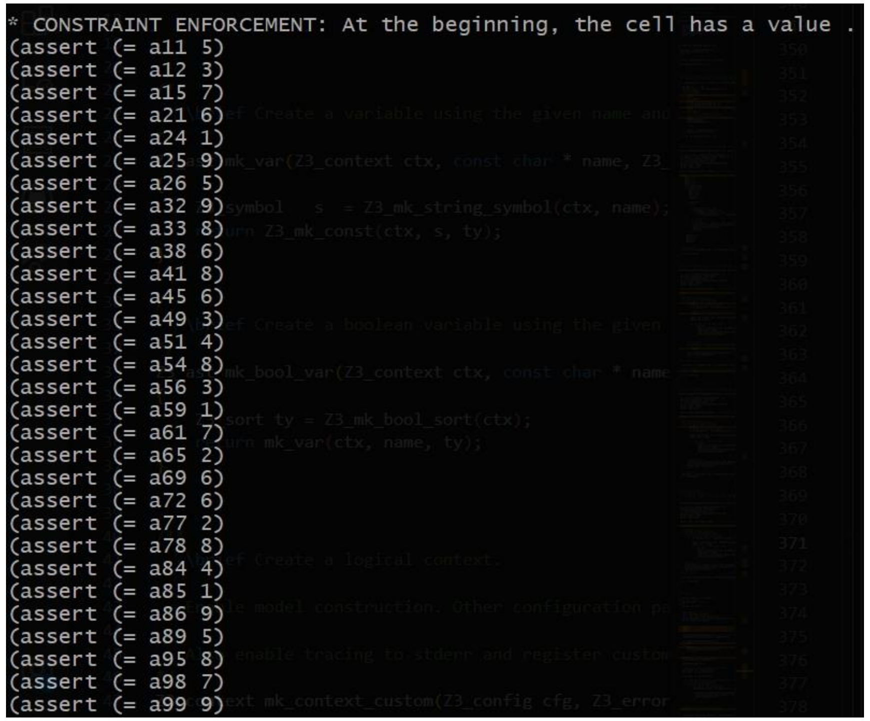
```
//for each cell(i,j)
 for (i = 0; i < NUM_LINE; i++)
   for (j = 0; j < NUM_LINE; j++){
           if(variable_list[i][j].digit != 0){ // the cell is already assigned a content
             total_constraint[counter_total_constraint] = Z3_mk_eq(*ctx, variables[i][j], Z3_mk_unsigned_int64(*ctx, variable_list[i][j].digit, Z3_mk_int_sort(*ctx)));
             counter_total_constraint ++;
           }
 }
constraint = Z3_mk_and (*ctx, counter_total_constraint,total_constraint );
```

Generate Constraints

First Equal



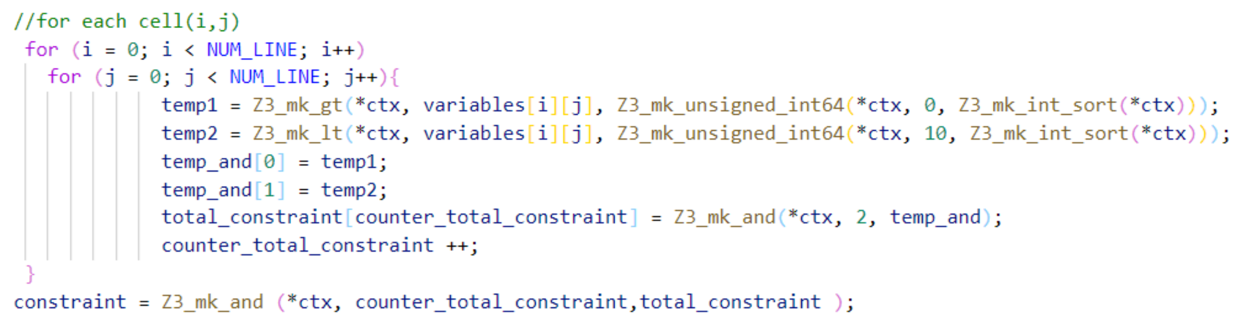
```
    INITIAL SET UP!
5 3 0 | 0 7 0 | 0 0 0
6 0 0 | 1 9 5 | 0 0 0
0 9 8 | 0 0 0 | 0 6 0
- - - - - - - - - - -
8 0 0 | 0 6 0 | 0 0 3
4 0 0 | 8 0 3 | 0 0 1
7 0 0 | 0 2 0 | 0 0 6
- - - - - - - - - - -
0 6 0 | 0 0 0 | 2 8 0
0 0 0 | 4 1 9 | 0 0 5
0 0 0 | 0 8 0 | 0 7 9
```



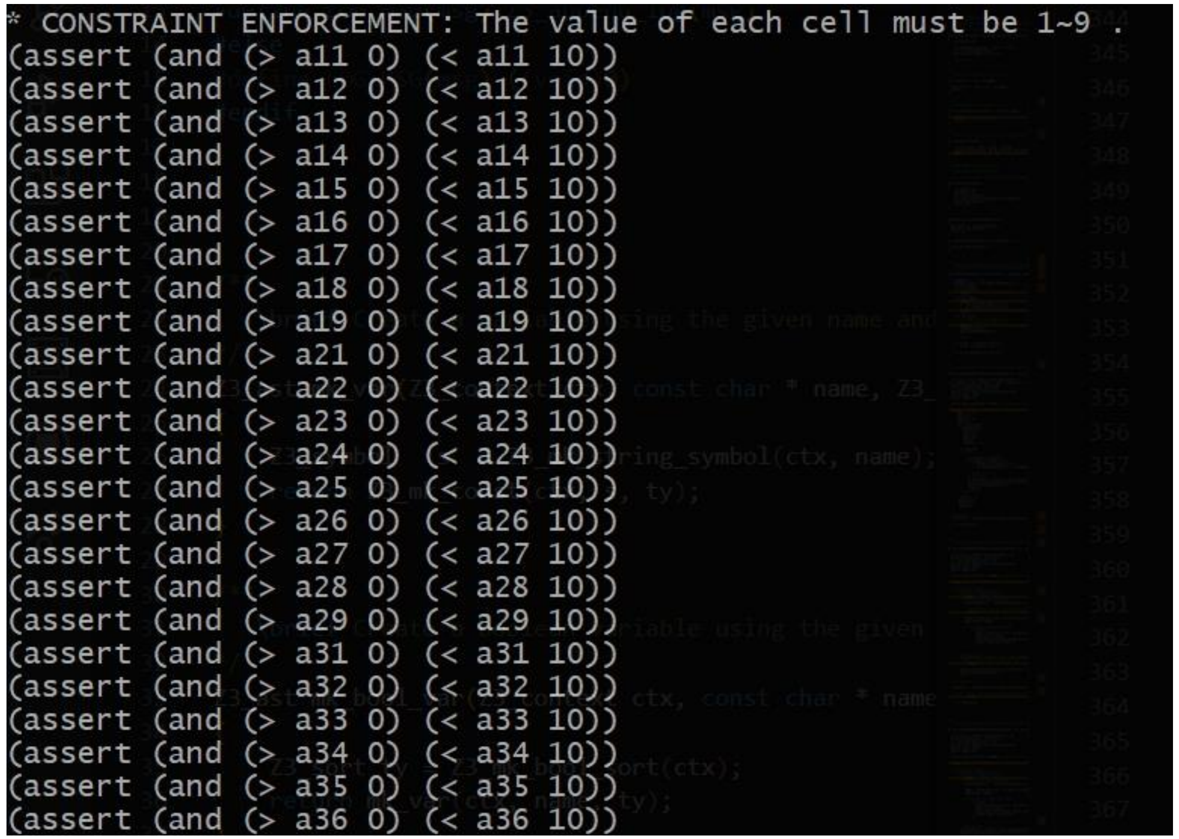
```
* CONSTRAINT ENFORCEMENT: At the beginning, the cell has a value .
(assert (= a11 5))
(assert (= a12 3))
(assert (= a15 7))
(assert (= a21 6))
(assert (= a24 1))
(assert (= a25 9))
(assert (= a26 5))
(assert (= a32 9))
(assert (= a33 8))
(assert (= a38 6))
(assert (= a41 8))
(assert (= a45 6))
(assert (= a49 3))
(assert (= a51 4))
(assert (= a54 8))
(assert (= a56 3))
(assert (= a59 1))
(assert (= a61 7))
(assert (= a65 2))
(assert (= a69 6))
(assert (= a72 6))
(assert (= a77 2))
(assert (= a78 8))
(assert (= a84 4))
(assert (= a85 1))
(assert (= a86 9))
(assert (= a89 5))
(assert (= a95 8))
(assert (= a98 7))
(assert (= a99 9))
```

Generate Constraints

- Cell Value



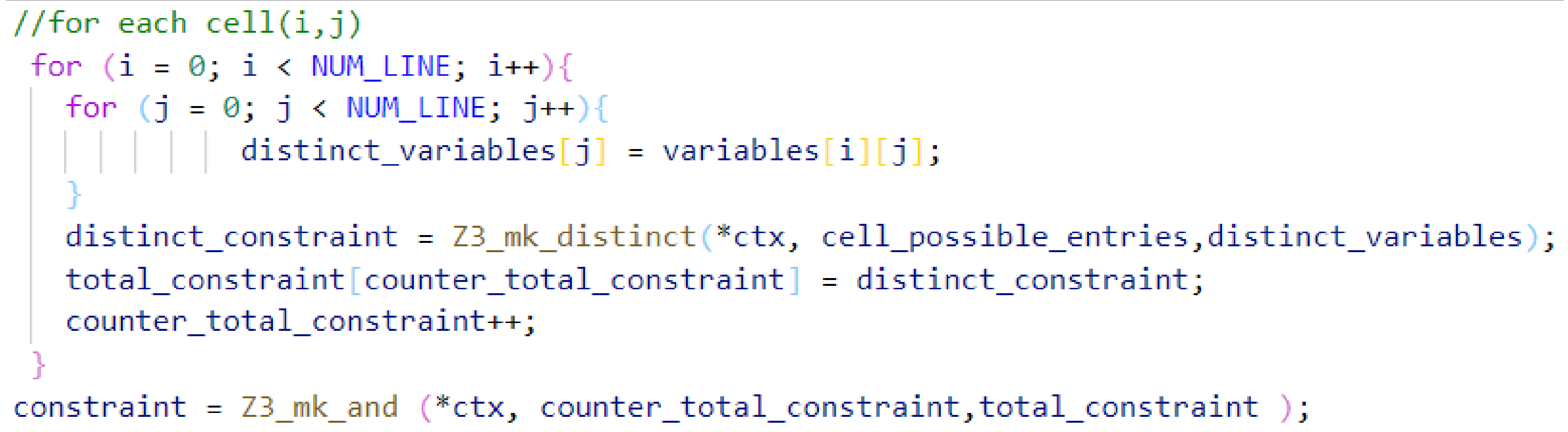
```
//for each cell(i,j)
 for (i = 0; i < NUM_LINE; i++)
   for (j = 0; j < NUM_LINE; j++){
            temp1 = Z3_mk_gt(*ctx, variables[i][j], Z3_mk_unsigned_int64(*ctx, 0, Z3_mk_int_sort(*ctx)));
            temp2 = Z3_mk_lt(*ctx, variables[i][j], Z3_mk_unsigned_int64(*ctx, 10, Z3_mk_int_sort(*ctx)));
            temp_and[0] = temp1;
            temp_and[1] = temp2;
            total_constraint[counter_total_constraint] = Z3_mk_and(*ctx, 2, temp_and);
            counter_total_constraint ++;
 }
constraint = Z3_mk_and (*ctx, counter_total_constraint,total_constraint );
```



```
* CONSTRAINT ENFORCEMENT: The value of each cell must be 1~9 .
(assert (and (> a11 0) (< a11 10)))
(assert (and (> a12 0) (< a12 10)))
(assert (and (> a13 0) (< a13 10)))
(assert (and (> a14 0) (< a14 10)))
(assert (and (> a15 0) (< a15 10)))
(assert (and (> a16 0) (< a16 10)))
(assert (and (> a17 0) (< a17 10)))
(assert (and (> a18 0) (< a18 10)))
(assert (and (> a19 0) (< a19 10)))
(assert (and (> a21 0) (< a21 10)))
(assert (and (> a22 0) (< a22 10)))
(assert (and (> a23 0) (< a23 10)))
(assert (and (> a24 0) (< a24 10)))
(assert (and (> a25 0) (< a25 10)))
(assert (and (> a26 0) (< a26 10)))
(assert (and (> a27 0) (< a27 10)))
(assert (and (> a28 0) (< a28 10)))
(assert (and (> a29 0) (< a29 10)))
(assert (and (> a31 0) (< a31 10)))
(assert (and (> a32 0) (< a32 10)))
(assert (and (> a33 0) (< a33 10)))
(assert (and (> a34 0) (< a34 10)))
(assert (and (> a35 0) (< a35 10)))
(assert (and (> a36 0) (< a36 10)))
```

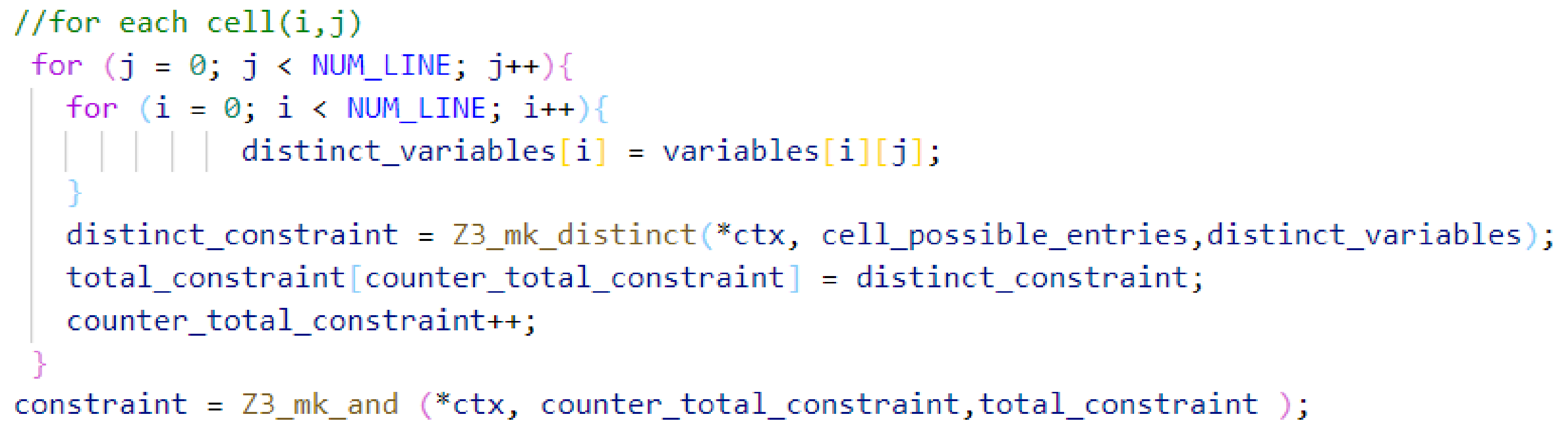
Generate Constraints

Once Each Row



```
//for each cell(i,j)
 for (i = 0; i < NUM_LINE; i++){
   for (j = 0; j < NUM_LINE; j++){
            distinct_variables[j] = variables[i][j];
   }
   distinct_constraint = Z3_mk_distinct(*ctx, cell_possible_entries,distinct_variables);
   total_constraint[counter_total_constraint] = distinct_constraint;
   counter_total_constraint++;
 }
constraint = Z3_mk_and (*ctx, counter_total_constraint,total_constraint );
```

Once Each Column

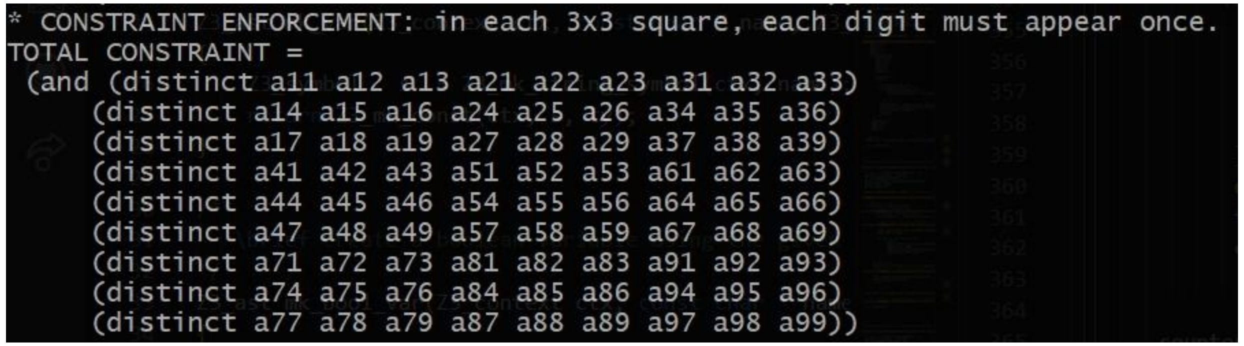


```
//for each cell(i,j)
 for (j = 0; j < NUM_LINE; j++){
   for (i = 0; i < NUM_LINE; i++){
            distinct_variables[i] = variables[i][j];
   }
   distinct_constraint = Z3_mk_distinct(*ctx, cell_possible_entries,distinct_variables);
   total_constraint[counter_total_constraint] = distinct_constraint;
   counter_total_constraint++;
 }
constraint = Z3_mk_and (*ctx, counter_total_constraint,total_constraint );
```

Generate Constraints

- Once Each Square

```
 while(end_i<=8){
  if(counter_1>3){
     begin_i+=3;
     end_i+=3;
     begin_j=0;
     end_j=2;
     counter_1 = 1;
  }
   if(square>9)
     break;
       for(i=begin_i;i<=end_i; i++){
          for(j=begin_j;j<=end_j; j++){
              distinct_variables[count++] =  variables[i][j];//Xijk: possible assignment into a cell(i,j)
           }//end for i
        }//end for j
        distinct_constraint =  Z3_mk_distinct (*ctx, count--, distinct_variables); //The content can be k = 1,2, ..., or 9: Xij1 V ...V Xij9
        total_constraint[counter_total_constraint] = distinct_constraint;
        counter_total_constraint++;
         count=0;
  counter_1 ++;
  begin_j+=3;
  end_j+=3;
  square++;
}//end while
constraint = Z3_mk_and (*ctx, counter_total_constraint,total_constraint );
```



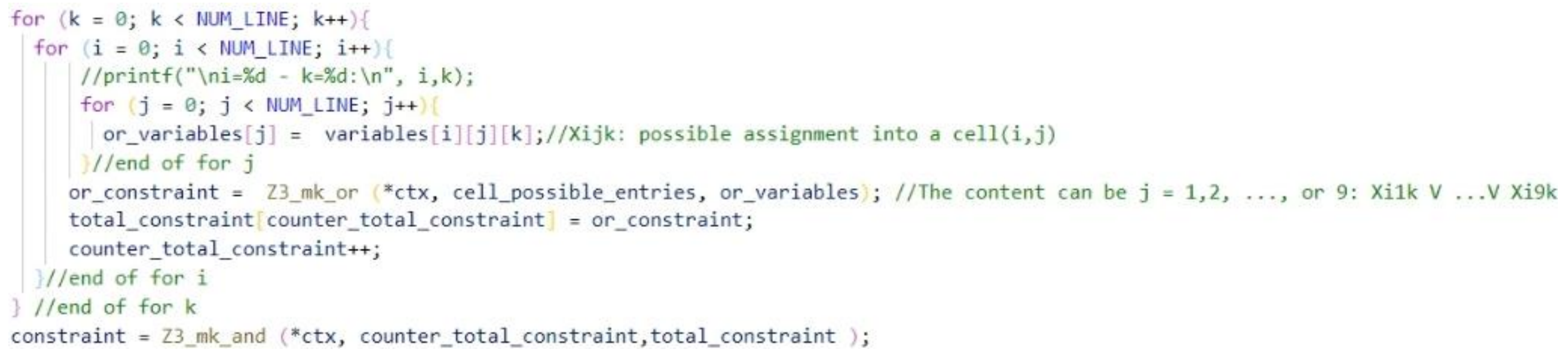
```
* CONSTRAINT ENFORCEMENT: in each 3x3 square, each digit must appear once.
TOTAL CONSTRAINT =
 (and (distinct a11 a12 a13 a21 a22 a23 a31 a32 a33)
     (distinct a14 a15 a16 a24 a25 a26 a34 a35 a36)
     (distinct a17 a18 a19 a27 a28 a29 a37 a38 a39)
     (distinct a41 a42 a43 a51 a52 a53 a61 a62 a63)
     (distinct a44 a45 a46 a54 a55 a56 a64 a65 a66)
     (distinct a47 a48 a49 a57 a58 a59 a67 a68 a69)
     (distinct a71 a72 a73 a81 a82 a83 a91 a92 a93)
     (distinct a74 a75 a76 a84 a85 a86 a94 a95 a96)
     (distinct a77 a78 a79 a87 a88 a89 a97 a98 a99))
```

Result and Discussion

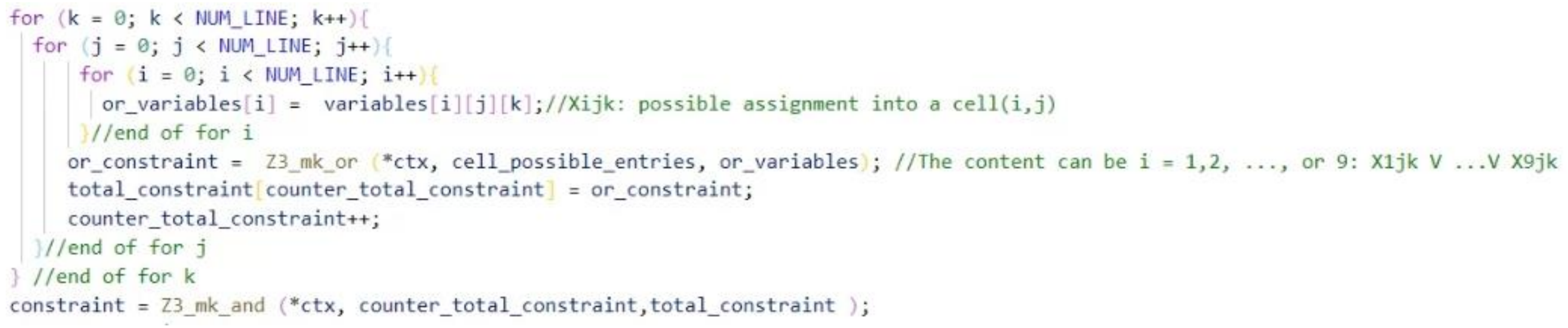
04

章節 PART

SAT Solver Optimization



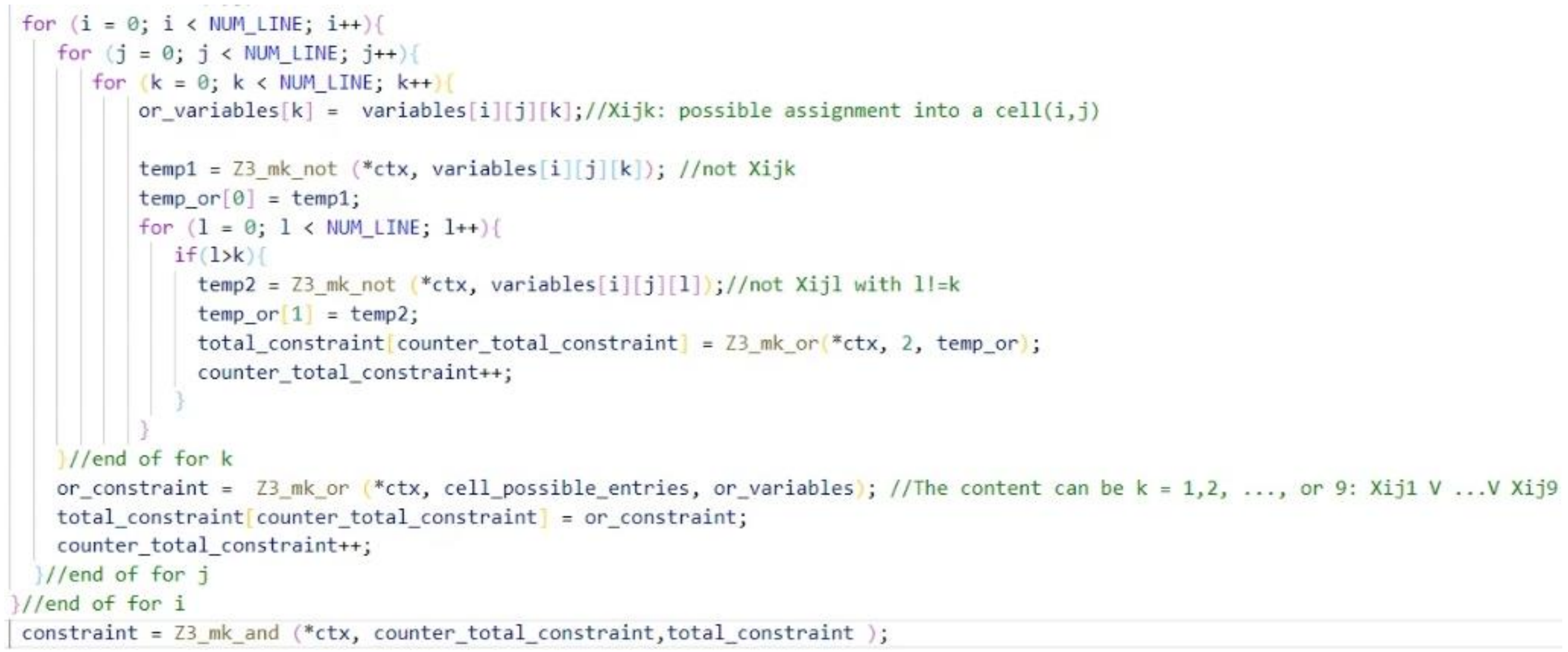
```
for (k = 0; k < NUM_LINE; k++){
  for (i = 0; i < NUM_LINE; i++){
      //printf("\ni=%d - k=%d:\n", i,k);
      for (j = 0; j < NUM_LINE; j++){
        or_variables[j] =  variables[i][j][k];//Xijk: possible assignment into a cell(i,j)
      }//end of for j
    or_constraint =  Z3_mk_or (*ctx, cell_possible_entries, or_variables); //The content can be j = 1,2, ..., or 9: Xi1k V ...V Xi9k
    total_constraint[counter_total_constraint] = or_constraint;
    counter_total_constraint++;
  }//end of for i
} //end of for k
constraint = Z3_mk_and (*ctx, counter_total_constraint,total_constraint );
```



```
for (k = 0; k < NUM_LINE; k++){
  for (j = 0; j < NUM_LINE; j++){
      for (i = 0; i < NUM_LINE; i++){
        or_variables[i] =  variables[i][j][k];//Xijk: possible assignment into a cell(i,j)
      }//end of for i
    or_constraint =  Z3_mk_or (*ctx, cell_possible_entries, or_variables); //The content can be i = 1,2, ..., or 9: X1jk V ...V X9jk
    total_constraint[counter_total_constraint] = or_constraint;
    counter_total_constraint++;
  }//end of for j
} //end of for k
constraint = Z3_mk_and (*ctx, counter_total_constraint,total_constraint );
```

SAT Solver Optimization

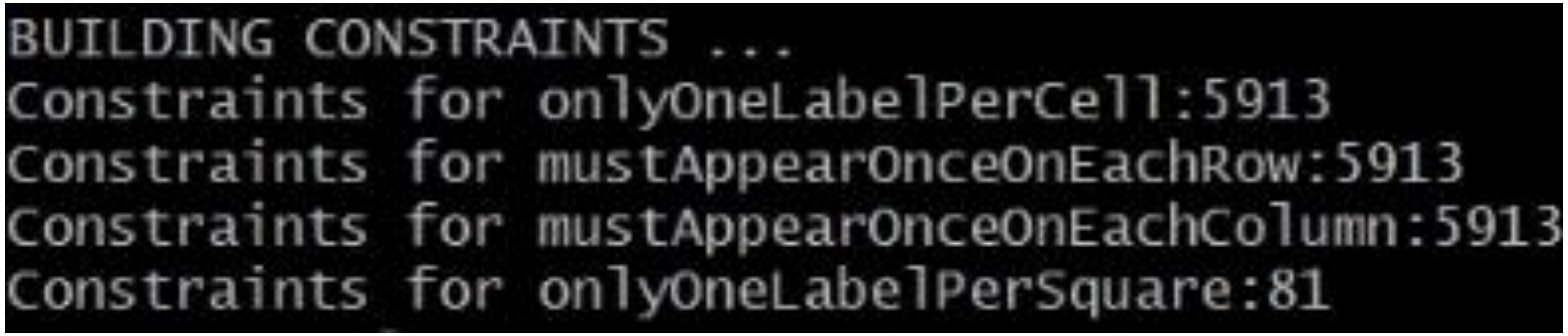
$(\neg X_{191} \lor \neg X_{911}) \; is \; same \; as \; (\neg s_{911} \lor \neg s_{191})$



```
for (i = 0; i < NUM_LINE; i++){
   for (j = 0; j < NUM_LINE; j++){
      for (k = 0; k < NUM_LINE; k++){
          or_variables[k] =  variables[i][j][k];//Xijk: possible assignment into a cell(i,j)

          temp1 = Z3_mk_not (*ctx, variables[i][j][k]); //not Xijk
          temp_or[0] = temp1;
          for (l = 0; l < NUM_LINE; l++){
             if(l>k){
               temp2 = Z3_mk_not (*ctx, variables[i][j][l]);//not Xijl with l!=k
               temp_or[1] = temp2;
               total_constraint[counter_total_constraint] = Z3_mk_or(*ctx, 2, temp_or);
               counter_total_constraint++;
             }
          }
   }//end of for k
   or_constraint =  Z3_mk_or (*ctx, cell_possible_entries, or_variables); //The content can be k = 1,2, ..., or 9: Xij1 V ...V Xij9
   total_constraint[counter_total_constraint] = or_constraint;
   counter_total_constraint++;
  }//end of for j
}//end of for i
constraint = Z3_mk_and (*ctx, counter_total_constraint,total_constraint );
```

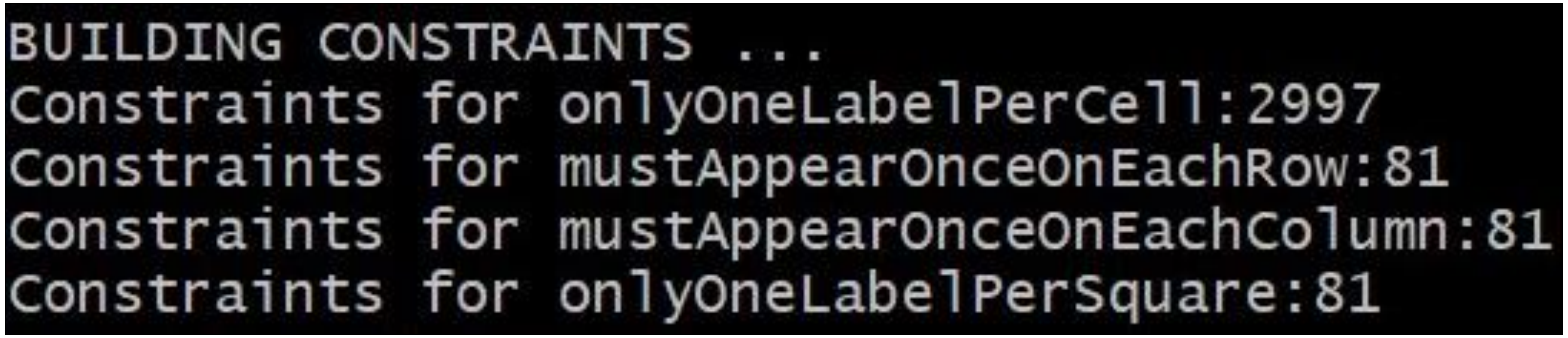
SAT Solver Optimization

- Before



```
BUILDING CONSTRAINTS ...
Constraints for onlyOneLabelPerCell:5913
Constraints for mustAppearOnceOnEachRow:5913
Constraints for mustAppearOnceOnEachColumn:5913
Constraints for onlyOneLabelPerSquare:81
```

- After



```
BUILDING CONSTRAINTS ...
Constraints for onlyOneLabelPerCell:2997
Constraints for mustAppearOnceOnEachRow:81
Constraints for mustAppearOnceOnEachColumn:81
Constraints for onlyOneLabelPerSquare:81
```

Comparison

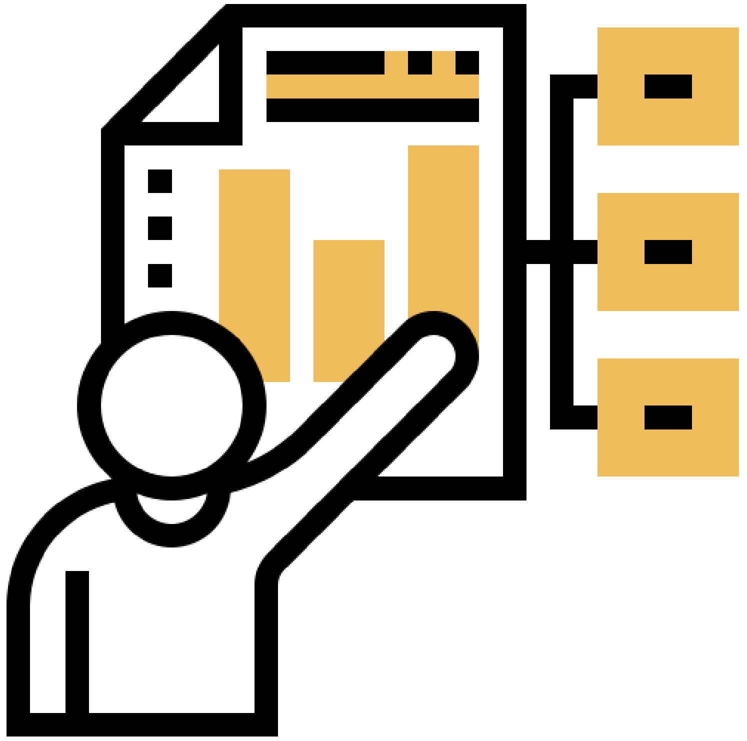
Simple Case	Backtracking	SAT Solver	Optimized SAT	SMT Solver
Time Consumption of generating Constraints		643.009 (ms)	208.019 (ms)	13.4789 (ms)
Time Consumption of Solving Sudoku	1.4581 (ms)	68.3863 (ms)	69.1039 (ms)	60.1699 (ms)
Total Time Consumption	1.9854 (ms)	719.193 (ms)	287.998 (ms)	82.5657 (ms)
Number of Constraints		17820	3240	108

Hard Case	Backtracking	SAT Solver	Optimized SAT	SMT Solver
Time Consumption of generating Constraints		636.420 (ms)	214.359 (ms)	13.4490 (ms)
Time Consumption of Solving Sudoku	95.9016 (ms)	63.1917 (ms)	910.141 (ms)	228.984 (ms)
Total Time Consumption	96.4829 (ms)	708.250 (ms)	1141.40 (ms)	250.178 (ms)

05

章節 PART

Conclusion



Graph Theory

THANK YOU

王梓帆 | 林冠名 | 顏子傑
Date: 2022 / 06 / 10 Fri.

Group: G