Import the Z3Interop package.

```
In[∘]:= Get@FileNameJoin[{DirectoryName@NotebookFileName[], "Z3Interop.wl"}]
```

Set up the problem: Sudoku

A sudoku problem has 81 integer variables.

```
In[∘]:= vars = Flatten@Table[a[i, j], {i, 1, 9}, {j, 1, 9}];
```

Some of those variables have known values.

```
In[∘]:= input = Partition[FromDigits /@ Characters[
          "008007060000060950000000000319000005008702000405600800002140000000006401000
            0050900100"], 9];
```

```
In[∘]:= known :=
         Cases[Flatten[Thread /@ Thread[input == Table[a[i, j], {i, 1, 9}, {j, 1, 9}]]],
          Equal[i_Integer ? (# > 0 &), a[_, _]]]
```

All of those variables are between 1 and 9.

```
In[∘]:= bounded = Flatten[{0 < #, # < 10} & /@ vars];
```

Each row must contain the numbers 1 through 9 at least once, and likewise for each column.

```
In[∘]:= rows = Flatten[Function[{x},
           Or @@@ Table[a[i, #] == x & /@ Range[1, 9], {i, 1, 9}]] /@ Range[1, 9]];
        cols = Flatten[Function[{x}, Or @@@ Table[a[#, i] == x & /@ Range[1, 9], {i, 1, 9}]] /@
            Range[1, 9]];
```

Each 3x3 box must contain the numbers 1 through 9 at least once.

```
In[∘]:= boxesF[row_, col_] := Function[{x},
           Or @@ Flatten@Table[a[i, j] == x, {i, row, row + 2}, {j, col, col + 2}]] /@ Range[9]
```

```
In[∘]:= boxes = Flatten@Table[boxesF[i, j], {i, 1, 9, 3}, {j, 1, 9, 3}];
```

The above constraints are the complete set of assertions we make about the variables.

```
In[∘]:= constraints := Assertion /@ Flatten[{bounded, known, rows, cols, boxes}];
```

Declare the variables to be integers.

```
In[∘]:= declared := Declare[#, Integer] & /@ vars;
```

Construct the program by concatenating the variable declaration section, constraint section, command to check satisfiability, and command to obtain a satisfaction.

```
In[∘]:= symbols = {a → "a"};
```

```
In[∘]:= program := Riffle[toString[symbols, #] & /@
           Flatten@{declared, constraints, CheckSat, GetModel}, "\n"] // StringJoin;
```

Run Z3

In[•]:=
```
s = OpenWrite[FormatType → OutputForm, PageWidth → Infinity];
Write[s, program];
outputLocation = Close[s]
```

Out[•]=
```
/private/var/folders/hz/9prp92151cqgf8370qt8ngfw0000gn/T/m00000383091
```

In[•]:=
```
output = RunProcess[{"z3", outputLocation},
    ProcessEnvironment → <|"PATH" → "/usr/local/bin/"|>]["StandardOutput"];
```

Is that instance satisfied?

In[•]:=
```
StringCases[output, RegularExpression["(un)?sat"]]
```

Out[•]=
```
{sat}
```

Parse the output

Obtain a model in which the constraints are satisfied, by parsing Z3’s output.

In[•]:=
```
answer = getDefinitions[symbols, output];
```

In[•]:=
```
Table[a[i, j], {i, 1, 9}, {j, 1, 9}] /. answer // Grid
```

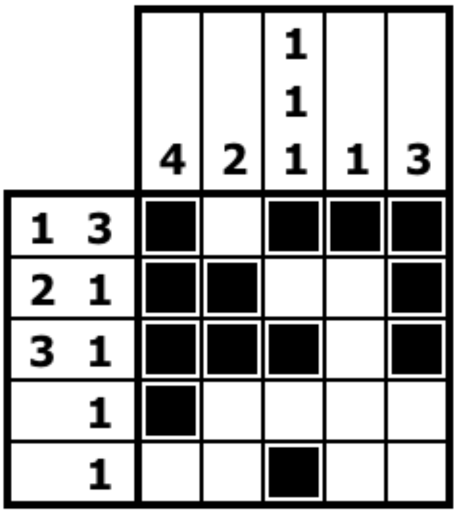
Out[•]=

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

Set up the problem: Nonogram

In[•]:=
```
$ContextPath = DeleteDuplicates@Append[$ContextPath, "Z3Interop`Nonogram`"];
```

Suppose we have as input a collection of row and column data. We will allow for multiple colours, although in this example the only colour is black.




```
In[•]:= rowsIn = {{{1, Black}, {3, Black}}, {{2, Black}, {1, Black}},
           {{3, Black}, {1, Black}}, {{1, Black}}, {{1, Black}}};
        colsIn = {{{4, Black}}, {{2, Black}}, {{1, Black}, {1, Black}, {1, Black}},
           {{1, Black}}, {{3, Black}}};
```

Define an arbitrary mapping of colours to numbers, so that we can represent the problem in integers.

```
In[•]:= mapping = With[{colours = Union@Cases[rowsIn, _?ColorQ, All]},
          Assert[FreeQ[colours, White]];
          MapIndexed[#1 → First@#2 &, Append[colours, White]]]

Out[•]= {■ → 1, □ → 2}
```

The constraints on a row are of the following form.
Suppose the input is the following:

```
In[•]:= col = {{4, ■}, {2, ■}, {2, ■}, {5, ■}, {1, ■}, {4, ■}, {4, ■}, {12, ■}};
```

Find the positions of colour changes x1,x2,x3,x4,… such that:

1<=x1

x1+4<=x2

x2+2<=x3

x3+2<x4

x4+5<=x5

x5+1<x6

x6+4<x7

x7+4<=x8

x8+12<=#rows

We represent this as the list of left-hand sides (x1+4, x2+2, …, x8+12), right-hand sides (x2, x3, …, x9), and operations (<=, <=, <, …), then prepend the first line and append the last.

```
In[•]:= gapsToConstraints[col, 1, 60, colGap]

Out[•]= {1 ≤ colGap[1, 1], 4 + colGap[1, 1] ≤ colGap[1, 2], 2 + colGap[1, 2] ≤ colGap[1, 3],
  2 + colGap[1, 3] < colGap[1, 4], 5 + colGap[1, 4] ≤ colGap[1, 5],
  1 + colGap[1, 5] < colGap[1, 6], 4 + colGap[1, 6] < colGap[1, 7],
  4 + colGap[1, 7] ≤ colGap[1, 8], 12 + colGap[1, 8] ≤ 61}
```

```
In[•]:= constrainedColumns :=
  MapIndexed[gapsToConstraints[#1, First@#2, Length@rowsIn, colGap] &, colsIn];
```

```
In[•]:= constrainedRows :=
  MapIndexed[gapsToConstraints[#1, First@#2, Length@colsIn, rowGap] &, rowsIn];
```

And tie together the row and column constraints.
The cell at index {row, column} is of colour colsIn⟦column, i, 2⟧ if colGap[column, i] <= row < colGap[-column, i]+colsIn⟦column,i,1⟧, and of colour white otherwise.
Similarly for the rows.

```
In[•]:= additionalConstraints := constrainedCells[rowGap, colGap, cell,
   rowsIn, colsIn, constrainedRows, constrainedColumns, mapping];
```

Form the program:

```
In[•]:= vars := DeleteDuplicates@
   Flatten@{Cases[{constrainedCells, constrainedColumns, constrainedRows},
      colGap[_, _], Infinity], Cases[{additionalConstraints, constrainedColumns,
       constrainedRows}, rowGap[_, _], Infinity], Cases[{additionalConstraints,
       constrainedColumns, constrainedRows}, cell[_, _], Infinity]};
```

```
In[•]:= constraints := Assertion /@
  Flatten[{additionalConstraints, constrainedColumns, constrainedRows}]
```

```
In[•]:= declared := Declare[#, Integer] & /@ vars
```

```
In[•]:= symbols = {colGap → "colGap", rowGap → "rowGap", cell → "cell"};
```

```
In[•]:= program := Riffle[toString[symbols, #] & /@
     Flatten@{declared, constraints, CheckSat, GetModel}, "\n"] // StringJoin;
```

This is an example where the built-in Z3Interop `toString` does not know how to perform addition. Teach it:

```
In[•]:= toString[symbols_, a_ + b_] :=
 StringJoin["(+ ", toString[symbols, a], " ", toString[symbols, b], ")"]
```

Write and run the program:

```
In[•]:= s = OpenWrite[FormatType → OutputForm, PageWidth → Infinity];
Write[s, program];
outputLocation = Close[s]

Out[•]= /private/var/folders/hz/9prp92151cqgf8370qt8ngfw0000gn/T/m00000684551
```

```
In[•]:= output = RunProcess[{"z3", outputLocation},
    ProcessEnvironment → <|"PATH" → "/usr/local/bin"|>]["StandardOutput"];
```

```
In[•]:= StringCases[output, RegularExpression["(un)?sat"]]
Out[•]= {sat}
```

Parse out the solution:

```
In[•]:= answer = getDefinitions[symbols, output];
In[•]:= Table[cell[i, j], {i, 1, Length@rowsIn}, {j, 1, Length@colsIn}] /. answer /.
          (Reverse /@ mapping) // Grid
```

Out[•]=