Introduction

All student names in group: Jerry Xiao, Dylan Lai, Safwan Kader

I understand that my submission needs to be my own group's work: [all group member's initials] JX DL SK

I understand that ChatGPT / Copilot / other AI tools are not allowed: [all group member's initials] JX DL SK

Points: 10

Complete this notebook and submit it (save/print as pdf). Make sure all output is correct in the pdf before submitting (it sometimes gets cut off). The notebook needs to be a complete project report with your implementation, documentation including a short discussion of how your implementation works and your design choices, and experimental results (e.g., tables and charts with simulation results) with a short discussion of what they mean. Use the provided notebook cells and insert additional code and markdown cells as needed.

Z3 and Z3py resources

For this problem you will be solving constraint satisfaction and logic problems using the Z3 SMT solver via the z3py python library.

The demo code using z3py we went over in class is here: link

Online help for z3py is here: link

Installation

We first install to get the z3solver library using pip and test that works.

```
!pip install z3-solver
```

```
Collecting z3-solver
  Obtaining dependency information for z3-solver from https://files.pythonhosted.org/packages/25/c0/dd978c813288f6860bcfb9e4d2d1d3b31
  Downloading z3_solver-4.13.0.0-py2.py3-none-win_amd64.whl.metadata (757 bytes)
Downloading z3_solver-4.13.0.0-py2.py3-none-win_amd64.whl (58.4 MB)
   ---------------------------------------- 0.0/58.4 MB ? eta -:--:--
   ---------------------------------------- 0.0/58.4 MB ? eta -:--:--
   ---------------------------------------- 0.1/58.4 MB 812.7 kB/s eta 0:01:12
   ---------------------------------------- 0.7/58.4 MB 6.3 MB/s eta 0:00:10
   - -------------------------------------- 1.6/58.4 MB 11.6 MB/s eta 0:00:05
   - -------------------------------------- 2.5/58.4 MB 12.1 MB/s eta 0:00:05
   -- ------------------------------------- 3.7/58.4 MB 14.7 MB/s eta 0:00:04
   --- ------------------------------------ 5.3/58.4 MB 17.8 MB/s eta 0:00:03
   ---- ----------------------------------- 7.0/58.4 MB 20.3 MB/s eta 0:00:03
   ----- ---------------------------------- 8.5/58.4 MB 21.6 MB/s eta 0:00:03
   ------ --------------------------------- 10.2/58.4 MB 23.2 MB/s eta 0:00:03
   -------- ------------------------------- 11.8/58.4 MB 29.7 MB/s eta 0:00:02
   -------- ------------------------------- 13.4/58.4 MB 34.6 MB/s eta 0:00:02
   ---------- ----------------------------- 15.1/58.4 MB 34.4 MB/s eta 0:00:02
   ----------- ---------------------------- 16.8/58.4 MB 34.4 MB/s eta 0:00:02
   ------------ --------------------------- 18.2/58.4 MB 34.4 MB/s eta 0:00:02
   ------------- -------------------------- 19.9/58.4 MB 34.4 MB/s eta 0:00:02
   -------------- ------------------------- 21.4/58.4 MB 34.4 MB/s eta 0:00:02
   --------------- ------------------------ 23.3/58.4 MB 34.4 MB/s eta 0:00:02
   ---------------- ----------------------- 23.9/58.4 MB 36.4 MB/s eta 0:00:01
   ---------------- ----------------------- 23.9/58.4 MB 36.4 MB/s eta 0:00:01
   ----------------- ---------------------- 25.0/58.4 MB 27.3 MB/s eta 0:00:02
   ----------------- ---------------------- 26.1/58.4 MB 25.2 MB/s eta 0:00:02
   ------------------ --------------------- 27.0/58.4 MB 24.2 MB/s eta 0:00:02
   ------------------- -------------------- 27.9/58.4 MB 23.4 MB/s eta 0:00:02
   ------------------- -------------------- 28.7/58.4 MB 23.4 MB/s eta 0:00:02
   -------------------- ------------------- 29.4/58.4 MB 22.6 MB/s eta 0:00:02
   -------------------- ------------------- 29.4/58.4 MB 22.6 MB/s eta 0:00:02
   -------------------- ------------------- 29.4/58.4 MB 22.6 MB/s eta 0:00:02
   -------------------- ------------------- 29.4/58.4 MB 22.6 MB/s eta 0:00:02
   -------------------- ------------------- 29.4/58.4 MB 22.6 MB/s eta 0:00:02
   -------------------- ------------------- 29.4/58.4 MB 22.6 MB/s eta 0:00:02
   -------------------- ------------------- 29.4/58.4 MB 22.6 MB/s eta 0:00:02
   -------------------- ------------------- 29.4/58.4 MB 22.6 MB/s eta 0:00:02
   -------------------- ------------------- 29.4/58.4 MB 22.6 MB/s eta 0:00:02
   -------------------- ------------------- 29.5/58.4 MB 11.3 MB/s eta 0:00:03
   -------------------- ------------------- 30.5/58.4 MB 11.1 MB/s eta 0:00:03
   --------------------- ------------------ 31.7/58.4 MB 10.9 MB/s eta 0:00:03
   ---------------------- ----------------- 32.9/58.4 MB 10.7 MB/s eta 0:00:03
   ----------------------- ---------------- 33.9/58.4 MB 10.6 MB/s eta 0:00:03
```

```
---------------------- --------------- 35.1/58.4 MB 11.5 MB/s eta 0:00:03
---------------------- --------------- 36.3/58.4 MB 11.5 MB/s eta 0:00:02
------------------------ -------------- 37.7/58.4 MB 11.7 MB/s eta 0:00:02
------------------------- ------------- 39.3/58.4 MB 12.3 MB/s eta 0:00:02
-------------------------- ------------ 40.8/58.4 MB 28.5 MB/s eta 0:00:01
--------------------------- ----------- 42.3/58.4 MB 28.5 MB/s eta 0:00:01
---------------------------- ---------- 43.9/58.4 MB 31.2 MB/s eta 0:00:01
----------------------------- --------- 45.6/58.4 MB 32.8 MB/s eta 0:00:01
------------------------------ -------- 47.3/58.4 MB 34.4 MB/s eta 0:00:01
------------------------------- ------- 48.6/58.4 MB 34.4 MB/s eta 0:00:01
-------------------------------- ------ 50.3/58.4 MB 34.4 MB/s eta 0:00:01
--------------------------------- ----- 51.8/58.4 MB 36.4 MB/s eta 0:00:01
```

```
# Run the first example from the z3py guide as a test: https://ericpony.github.io/z3py-tutorial/guide-examples.htm
from z3 import *

x = Int('x')
y = Int('y')
solve(x > 2, y < 10, x + 2*y == 7)
```

```
[y = 0, x = 7]
```

Sudoku as a constraint satisfaction problem (CSP)

Sudoku is a popular number-placement puzzle that originated in France in the end of the 19th century. Modern Sudoku was likely invented by Howard Garns from Connersville, Indiana and was first published in 1979 under the name *Number Place*. The objective of the puzzle is to place numbers 1-9 on a 9×9 grid, such that each number occurs only once in every row, every column, and every of the nine 3×3 sub-grids that compose the main grid. Sudoku puzzles are grids that have been partially occupied with numbers. The task is then to occupy the remaining fields in such a way that the constraints on rows, columns, and sub-grids are satisfied. For more information about Sudoku refer to its Wikipedia page at http://en.wikipedia.org/wiki/Sudoku.

This problem has two parts. In the first part, you will write the boolean constraints in mathematical notation for solving a Sudoku puzzle. In the second part, you will write code and invoke the Z3 solver to solve the Sudoku instance and answer various questions about the solution.

Part 1 (Constraints)

In text, define and write constraints over **boolean** variables corresponding to each number being in each cell. For example, you may use $X^k_{i,j}$ as teh variable that is true if and only if the number k is in row i and column j (the variable is true if k is in cell (i, j) and false otherwise). Now write the following boolean constaints over these variables:

- Write the boolean formula for the constraints that the number 5 can occur at most once in the first row. **(1 point)**

$\bigvee_{i=0}^{9}((X^5_{1,i;\ i>0}) \bigwedge_{j=1;\ i!=j}^{9} \neg X^5_{1,j})$

- Write the boolean formula for the constraints that the number 6 can occur at most once in the first column. **(1 point)**

$\bigvee_{i=0}^{9}((X^6_{i,1;\ i>0}) \bigwedge_{j=1;\ i!=j}^{9} \neg X^6_{j,1})$

- Write the boolean formula for the constraints that the number 9 can occur at most once in the top left 3×3 sub-grid. **(1 point)**

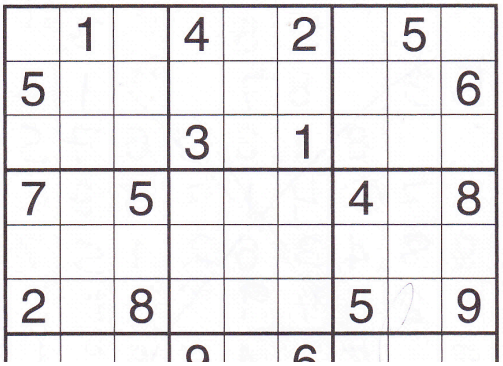
$\bigvee_{i=1}^{3} \bigvee_{j=1}^{3}((X^9_{i,j;\ i>0 \bigwedge\ j>0}) \bigwedge_{k=1}^{3} \bigwedge_{l=1}^{3} \neg X^9_{i,j}\ if\ i! = k\ or j! = l)$

Solution:

Part 2 (Coding)

Encode the above constraints and all the other ones as a CSP using z3py and solve the Sudoku instance given in the figure below. **Use only boolean variables** and do not use the **Distinct** function.

- Provide the code to solve that solves the given problem instance using z3py and only boolean variables. The code should output a reasonable visualization of the solution, for example printed in text. **(1 points)**



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

```
# z3py code to solve above Sudoku goes here. Do not use Int(*), only Bool(*)
from z3 import *

# In form X[row][column][number]
X = [[[Bool(f'X_({i},{j})^{k+1}') for k in range(9)]for j in range(9)]for i in range(9)]

solve = Solver()
#Encode starting conditions

starting_condition = [X[0][1][0],X[0][3][3],X[0][5][1],X[0][7][4],X[1][0][4],X[1][8][5],X[2][3][2],X[2][5][0],X[3][0][6]
                      ,X[3][2][4],X[3][6][3],X[3][8][7],X[5][0][1],X[5][2][7],X[5][6][4],X[5][8][8],X[6][3][8],X[6][5][5]
                     ,X[7][0][5],X[7][8][2],X[8][1][6],X[8][3][0],X[8][5][2],X[8][7][3]]
for start in starting_condition:
    solve.add(start)


# Encode that for every number, it must uniquely exist in a row
for x in range(1,10):
    # for every row:
    for row in range(0,9):
        ors = []
        for i in range(9):
            ands = []
            ands.append(X[row][i][x-1])
            for j in range(9):
                if j!=i:
                    ands.append(Not(X[row][j][x-1]))
            ors.append(And(*ands))
        solve.add(Or(*ors))

# Encode that for every column it must uniquely exist in a column:
for x in range(1,10):
    for col in range(0,9):
        ors = []
        for i in range(9):
            ands = []
            ands.append(X[i][col][x-1])
            for j in range(9):
                if j!=i:
                    ands.append(Not(X[j][col][x-1]))
            ors.append(And(*ands))
        solve.add(Or(*ors))

# Encode that every number must exist uniquely in a box
for x in range(1,10):
    for boxRow in range(0,9,3):
        for boxCol in range(0,9,3):
            ors = []
            for row in range(boxRow,boxRow+3):
                for col in range(boxCol,boxCol+3):
                    ands = []
                    ands.append(X[row][col][x-1])
                    for row1 in range(boxRow,boxRow+3):
                        for col1 in range(boxCol,boxCol+3):
                            if row1!=row or col1!=col:
                                ands.append(Not(X[row1][col1][x-1]))
                    ors.append(And(*ands))
            solve.add(Or(*ors))

# Encode that every entry has a unique number
for row in range(9):
    for col in range(9):
        ors = []
        for val in range(1,10):
            ands = []
            ands.append(X[row][col][val-1])
            for i in range(1,10):
                if i!=val:
                    ands.append(Not(X[row][col][i-1]))
            ors.append(And(*ands))
        solve.add(Or(*ors))


if solve.check()==sat:
    model = solve.model()
    solution = [[0 for _ in range(9)] for _ in range(9)]
```

```
    for row in range(9):
        for col in range(9):
            for val in range(1,10):
                if is_true(model[X[row][col][val-1]]):
                    solution[row][col]=val
                    break
    for row in solution:
        print(" ".join(str(val) for val in row))
else:
    print("failed to solve sudoku ")
```

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

Uniqueness

Is your solution unique? Prove it with a Z3py solver or provide a second solution. **(1 point)**

```
# Code goes here
model = solve.model()
ors = []
solveunique = solve
for row in range(9):
    for cols in range(9):
        for val in range(1,10):
            if is_true(model[X[row][col][val-1]]):
                ors.append(Not(X[row][col][val-1]))
                break
solveunique.add(Or(*ors))

if solveunique.check()==sat:
    model = solveunique.model()
    solution = [[0 for _ in range(9)] for _ in range(9)]

    for row in range(9):
        for col in range(9):
            for val in range(1,10):
                if is_true(model[X[row][col][val-1]]):
                    solution[row][col]=val
                    break
    for row in solution:
        print(" ".join(str(val) for val in row))
else:
    print("failed to solve sudoku")
    print("implies solution is unique")
```

```
 failed to solve sudoku
 implies solution is unique
```

More Uniqueness

If you delete the 1 in the top left box of the Sudoku problem above, how many unique solutions are there? Hint: should be less than 5000. **(1 point)**

```
#Redeclare a new solver

solveuniques = Solver()
#Encode starting conditions

#Now starting condition doesn't have that 1 in the corner
starting_condition = [X[0][3][3],X[0][5][1],X[0][7][4],X[1][0][4],X[1][8][5],X[2][3][2],X[2][5][0],X[3][0][6]
                      ,X[3][2][4],X[3][6][3],X[3][8][7],X[5][0][1],X[5][2][7],X[5][6][4],X[5][8][8],X[6][3][8],X[6][5][5]
                     ,X[7][0][5],X[7][8][2],X[8][1][6],X[8][3][0],X[8][5][2],X[8][7][3]]
for start in starting_condition:
    solveuniques.add(start)


# Encode that for every number, it must uniquely exist in a row
for x in range(1,10):
    # for every row:
    for row in range(0,9):
        ors = []
        for i in range(9):
            ands = []
            ands.append(X[row][i][x-1])
            for j in range(9):
                if j!=i:
                    ands.append(Not(X[row][j][x-1]))
            ors.append(And(*ands))
        solveuniques.add(Or(*ors))

# Encode that for every column it must uniquely exist in a column:
for x in range(1,10):
    for col in range(0,9):
        ors = []
        for i in range(9):
            ands = []
            ands.append(X[i][col][x-1])
            for j in range(9):
                if j!=i:
                    ands.append(Not(X[j][col][x-1]))
            ors.append(And(*ands))
        solveuniques.add(Or(*ors))

# Encode that every number must exist uniquely in a box
for x in range(1,10):
    for boxRow in range(0,9,3):
        for boxCol in range(0,9,3):
            ors = []
            for row in range(boxRow,boxRow+3):
                for col in range(boxCol,boxCol+3):
                    ands = []
                    ands.append(X[row][col][x-1])
                    for row1 in range(boxRow,boxRow+3):
                        for col1 in range(boxCol,boxCol+3):
                            if row1!=row or col1!=col:
                                ands.append(Not(X[row1][col1][x-1]))
                    ors.append(And(*ands))
            solveuniques.add(Or(*ors))

# Encode that every entry has a unique number
for row in range(9):
    for col in range(9):
        ors = []
        for val in range(1,10):
            ands = []
            ands.append(X[row][col][val-1])
            for i in range(1,10):
                if i!=val:
                    ands.append(Not(X[row][col][i-1]))
            ors.append(And(*ands))
        solveuniques.add(Or(*ors))

uniqueCount = 0
while solveuniques.check()==sat:
    uniqueCount+=1
    ors = []
    model1 = solveuniques.model()
    for row in range(9):
        for col in range(9):
            for val in range(9):
```

```
                if is_true(model1[X[row][col][val]]):
                    ors.append(Not(X[row][col][val]))
    solveuniques.add(Or(*ors))

print(f'Number of unique solutions is {uniqueCount}')
```

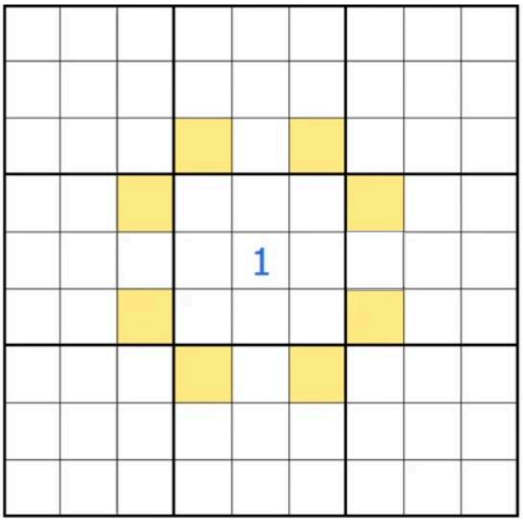
```
Number of unique solutions is 275
```

Miracle Sudoku

Using z3py and only boolean variables, solve the Miracle Sudoku (spoiler). In this puzzle normal Sudoku rules apply in addition to the following rules:

- Any two cells separated by a knight's move from chess (moving forward two boxes and over one, in any direction) cannot contain the same digit (see image below).
- Any two cells separated by a king's move from chess (one box in any direction including diagonals) cannot contain the same digit.
- Any two orthogonally adjacent (up, down, left, or right) cells cannot contain consecutive digits.

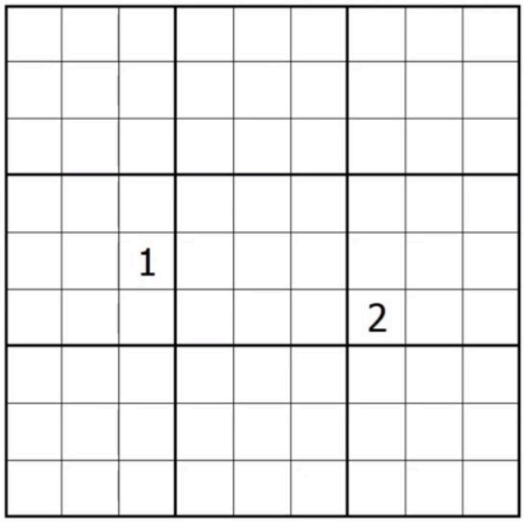
Knight's move constraint image



The knight's move constraint would mean the yellow boxes cannot contain a 1.

Miracle Sudoku Starting Information

Provide the code to solve the puzzle given below. The code should output a reasonable visualization of the solution, for example using text. **(2 points)**


1
2

```
# Solution code for Miracle Sudoku using z3py
from z3 import *

X = [[[Bool(f'X_({i},{j})^{k+1}') for k in range(9)]for j in range(9)]for i in range(9)]

miraclesolve = Solver()
#Encode starting conditions

starting_condition = [X[4][2][0],X[5][6][1]]
for start in starting_condition:
    miraclesolve.add(start)


# Encode that for every number, it must uniquely exist in a row
for x in range(1,10):
    # for every row:
    for row in range(0,9):
        ors = []
        for i in range(9):
            ands = []
            ands.append(X[row][i][x-1])
            for j in range(9):
                if j!=i:
                    ands.append(Not(X[row][j][x-1]))
            ors.append(And(*ands))
        miraclesolve.add(Or(*ors))

# Encode that for every column it must uniquely exist in a column:
for x in range(1,10):
    for col in range(0,9):
        ors = []
        for i in range(9):
            ands = []
            ands.append(X[i][col][x-1])
            for j in range(9):
                if j!=i:
                    ands.append(Not(X[j][col][x-1]))
            ors.append(And(*ands))
        miraclesolve.add(Or(*ors))

# Encode that every number must exist uniquely in a box
for x in range(1,10):
    for boxRow in range(0,9,3):
        for boxCol in range(0,9,3):
            ors = []
            for row in range(boxRow,boxRow+3):
                for col in range(boxCol,boxCol+3):
                    ands = []
                    ands.append(X[row][col][x-1])
                    for row1 in range(boxRow,boxRow+3):
                        for col1 in range(boxCol,boxCol+3):
                            if row1!=row or col1!=col:
                                ands.append(Not(X[row1][col1][x-1]))
                    ors.append(And(*ands))
            miraclesolve.add(Or(*ors))

# Encode that every entry has a unique number
for row in range(9):
    for col in range(9):
        ors = []
        for val in range(1,10):
            ands = []
            ands.append(X[row][col][val-1])
            for i in range(1,10):
                if i!=val:
                    ands.append(Not(X[row][col][i-1]))
            ors.append(And(*ands))
        miraclesolve.add(Or(*ors))

# Now encode that knights move away cannot have same numbers

def valid(x,y):
    if x<9 and x>=0 and y<9 and y>=0:
        return True
    return False

knight_moves = [(-2,-1),(-2,1),(-1,2),(1,2),(2,1),(2,-1),(1,-2),(-1,-2),(-2,-1)]
```

Propositional Logic