CS228
Assignment 1

Rishika Dhiman (24B1090)
Uma Kumari (24B1026)

August 2025

Contents

1 Question 1: SAT-based Sudoku Solver

1.1 Approach Overview

The SAT-based Sudoku Solver employs the rules of sudoku to solve the partially filled sudoku that is provided as the input. The rules that have been used are-

- All the numbers that have been given in the partially filled sudoku as the input must be present in the output as well.
- All the numbers(numbers from 1 to 9) must be present in each column.
- All the numbers(numbers from 1 to 9) must be present in each row.
- No cell must have more than one number assigned to it.
- Since the Sudoku square has a side length of 9, we divide the Sudoku square is divided into 9 squares of equal side length. Each of these squares must have all the numbers(numbers from 1 to 9).

1.2 Constraints and Variable Encoding

We consider the $P(i, j, n)$ to be true when the number n is present in the i^{th} row and the j^{th} column, where $1 \leq i, j, n \leq 9$.
In the function, the propositional variable $P(i, j, n)$ is represented by a three-digit number given by

$$100.i + 10.j + n$$

where the first digit is the row number, the second digit is the column number, and the third digit is the number in that cell.

1.2.1 All the Numbers that have been given in the Partially Filled Sudoku as the Input must be Present in the Output as well

- All the $P(i, j, n)$ where it is given that the number in cell (i, j) is n, must be true so each of them should be appended to the CNF formula.

1.2.2 All the Numbers must be Present in each Row

- Numbers from 1 to 9 must all be present in each row thus for any row i, the conjunction across all values of n of the disjunction of $P(i, j, n)$ across all columns for each value of n must be true.
- Each row must contain all 9 numbers
  - Thus the following equation must evaluate to true after taking the conjunction across all values of i.

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

1.2.3 All the Numbers must be Present in each Column

- For each column j, $P(i,j,n)$ should hold for all n for some i.
- Therefore, the disjunction of $\bigvee_{i=1}^{9} P(i,j,n)$ for all n and j should be true. Hence, the following formula should evaluate to true.

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

1.2.4 No Cell must have Two Numbers Assigned to it

- No cell can have more than one number assigned to it.
- Thus, the following must evaluate to true

$$\bigwedge_{m=1}^{9}\bigwedge_{n=m}^{9}\neg\big(P(i,j,m)\wedge P(i,j,n)\big)$$

1.2.5 All 3x3 blocks must have all the Numbers

- Every number must be present in each of the 3x3 boxes
- For the block region, let $i,j \in \{1,2,3\}$, the following formula must hold:

$$\bigwedge_{n=1}^{9}\left(\bigvee_{i=1}^{3}\bigvee_{j=1}^{3} P(i,j,n)\right)$$

Similarly, the same can be done for other 3×3 blocks.
The conjunction of the formulas across all blocks can be written as:

$$\bigwedge_{a=0}^{2}\bigwedge_{b=0}^{2}\bigwedge_{n=1}^{9}\left(\bigvee_{i=1}^{3}\bigvee_{j=1}^{3} P(3a+i,\ 3b+j,\ n)\right)$$

where $P(i,j,n)$ means that the number n is placed in cell (i,j).

2 Question 2

2.1 Approach Overview

Appending the following conditions and constraints to the CNF.

- Initial conditions
- Player movement
- Box movement (push rules)
- Non-overlap constraints
- Goal conditions
- Other conditions

2.2 Variable Encoding

- Let $P(i, j, t)$ evaluate to true when player P is at position (i, j) at time t. Suppose it is represented by the following number-

$$10000 + 1000.i + 100.j + t$$

- Let $B(b, i, j, t)$ evaluate to true when Box b is at position (i, j) at time t. Suppose it is represented by the following number-

$$200000 + 10000.b + 1000.i + 100.j + t$$

- N is the number of rows in the grid, M is the number of columns in the grid and B is the number of boxes in the grid. T is the total time of movement.
- t varies from 0 to T, i varies from 0 to $N-1$, j varies from 0 to $M-1$ and b varies from 0 to $B-1$. $(t = 0)$ signifies the initial conditions.

2.3 Constraints

2.3.1 Initial Conditions

- $P(i, j, 0)$ must be true if the position of Player given initially is (i, j). Thus after encoding propositional variable $P(i, j, 0)$ we must append it to the CNF.
- $B(b, i, j, 0)$ must be true if the position of Box b given initially is (i, j). Thus after encoding propositional variable $B(b, i, j, 0)$ we must append it to the CNF.

2.3.2 The Player Cannot be in the Same Cell as a Box

The player and a box cannot be at the same cell at the same time thus, the following must evaluate to true.

$$\bigwedge_{t=0}^{T} \bigwedge_{i=0}^{N-1} \bigwedge_{j=0}^{M-1} \bigwedge_{b=0}^{B-1} (\neg P(i,j,t) \ \vee \ \neg B(b,i,j,t))$$

Each variable can then be encoded, and the lists of the variables which are connected by disjunction can be appended to the CNF.

2.3.3 Player position constraints

- The player can only be at one position at a time

$$\bigwedge_{t=0}^{T} \bigwedge_{i=0}^{N-1} \bigwedge_{j=0}^{M-1} \bigwedge_{\substack{n=0\\ n\neq i}}^{N-1} \bigwedge_{\substack{m=0\\ m\neq j}}^{M-1} (\neg P(i,j,t) \ \vee \ \neg P(n,m,t))$$

  Thus the above must evaluate to true. Each variable can then be encoded and the lists of the variables which are connected by disjunction can be appended to the CNF.

- The player cannot be present inside a box having walls
  Hence $\neg P(i,j,t)$ should be appended to the CNF for all $0 \leq t \leq T$ if (i,j) position in the grid is a wall ($grid[i][j]$ is equal to '#').

2.3.4 Box position constraints

- The box can only be at one position at a time

$$\bigwedge_{t=0}^{T} \bigwedge_{b=0}^{B-1} \bigwedge_{i=0}^{N-1} \bigwedge_{j=0}^{M-1} \bigwedge_{\substack{n=0\\ n\neq i}}^{N-1} \bigwedge_{\substack{m=0\\ m\neq j}}^{M-1} (\neg B(b,i,j,t) \ \vee \ \neg B(b,n,m,t))$$

  Thus the above must evaluate to true. Each variable can then be encoded and be the lists of the variables which are connected by disjunction can be appended to the CNF.

- The box cannot be present inside a box having walls.
  Hence $\neg B(b,i,j,t)$ should be appended to the CNF for all $0 \leq t \leq T$ and b, if (i,j) position in the grid is a wall ($grid[i][j]$ is equal to '#').

- If at any time $t+1$ where, $0 \leq t \leq T-1$, a box is at (i,j), then at time t it must have been at any of the following positions (subject to the boundaries of the grid):
  - (i,j)
  - $(i-1,j)$ when $i > 0$

  - $(i+1, j)$ when $i < N-1$
  - $(i, j-1)$ when $j > 0$
  - $(i, j+1)$ when $j < M-1$

Hence, the formula can be written as:

$$B(b, i, j, t+1) \rightarrow S$$

where S is the disjunction of all possible box positions at time t, i.e.

$$S = \bigvee B(b, i', j', t)$$

for all valid positions (i', j').
Hence the following clause should hold for all b,

$$\neg B(b, i, j, t) \vee S$$

This is done to ensure that the box does not teleport (cover two or more cells in any direction).

- A box cannot be bounded from two sides if they are perpendicular to each other at all times except when they are in the goal otherwise the player will not be able to move it. By being bounded it is meant that either a wall is present on its sides or it is at the boundary of the grid.

  - The boxes cannot be at the corners. Hence the following should be appended to the CNF.

    $$\neg B(b,0,0,t) \wedge \neg B(b,0,M-1,t) \wedge \neg B(b,N-1,0,t) \wedge \neg B(b,N-1,M-1,t)$$

  - A box cannot have walls at two sides as then the player cannot move the box thus, the following should be appended to the CNF for valid i and j-

    $$\neg B(b, i, j, t) \quad \text{if } \mathit{grid}(i+1, j) = \# \text{ and } \mathit{grid}(i, j-1) = \#$$

    $$\neg B(b, i, j, t) \quad \text{if } \mathit{grid}(i+1, j) = \# \text{ and } \mathit{grid}(i, j+1) = \#$$

    $$\neg B(b, i, j, t) \quad \text{if } \mathit{grid}(i-1, j) = \# \text{ and } \mathit{grid}(i, j-1) = \#$$

    $$\neg B(b, i, j, t) \quad \text{if } \mathit{grid}(i-1, j) = \# \text{ and } \mathit{grid}(i, j+1) = \#$$

– a box cannot have an edge of the grid on one of its edges and a wall on an adjacent edge.the following must be appended for valid i and j-

$$\neg B(b,i,j,t) \quad \text{if } (i=0 \textit{ or } i=N{-}1) \wedge \big(grid(i,j{-}1)=\# \textit{ or } grid(i,j{+}1)=\#\big)$$

$$\neg B(b,i,j,t) \quad \text{if } (j=0 \textit{ or } j=M{-}1) \wedge \big(grid(i{-}1,j)=\# \textit{ or } grid(i{+}1,j)=\#\big)$$

2.3.5 Player movement constraints

If at any time t where, $0 \le t \le T-1$, a player is at (i,j), then at time $t+1$ the player can be at either of the following positions (subject to the boundaries of the grid):

- (i,j)
- $(i-1,j)$ when $i > 0$
- $(i+1,j)$ when $i < N-1$
- $(i,j-1)$ when $j > 0$
- $(i,j+1)$ when $j < M-1$

Hence, the formula can be written as:

$$P(i,j,t) \ \rightarrow \ S$$

where S is the disjunction of all possible player positions at time $t+1$, i.e.

$$S = \bigvee P(i',j',t+1)$$

for all valid moves (i',j').
Hence the following should hold,

$$\neg P(i,j,t) \vee S$$

The case of the player moving into the place of a box is handled by the box movement constraints: if a box cannot be moved, then that player move will not be possible.

2.3.6 Box movement constraints

- If a box moved it was moved by the player.

  If a box was present at (i, j) at time t and it moved to $(i+1, j)$ at time $t+1$, then the player was present at $(i-1, j)$ at time t and moved to (i, j) at time $t+1$ (for values of (i, j) within the bounds).

  Hence, the formula is:

  $$B(i,j,t) \ \wedge \ B(i+1,j,t+1) \ \rightarrow \ P(i-1,j,t) \ \wedge \ P(i,j,t+1)$$

  which is equivalent to,

  $$\neg B(i,j,t) \ \vee \ \neg B(i+1,j,t+1) \ \vee \ (P(i-1,j,t) \ \wedge \ P(i,j,t+1))$$

  Applying distributivity of $\vee$ over $\wedge$, we can rewrite it in CNF as:

  $$\big(\neg B(i,j,t) \vee \neg B(i{+}1,j,t{+}1) \vee P(i{-}1,j,t)\big) \wedge \big(\neg B(i,j,t) \vee \neg B(i{+}1,j,t{+}1) \vee P(i,j,t{+}1)\big)$$

  These two clauses should be appended in the CNF for all b.
  Similarly we can derive clauses for movements of the box in other directions.

- If the box does not move from its position then the player has not moved it. So we have to employ the constraint that the player cannot move from adjacent position to position of box if the box has not moved and also that if the box is at the edge of the grid then it cannot be moved perpendicular to it. Thus the following must be appended-

  for $i > 0$:

  $$B(b,i,j,t) \wedge B(b,i,j,t+1) \quad \rightarrow \quad \neg(P(i-1,j,t) \ \wedge \ P(i,j,t+1))$$

  $$\neg B(b,i,j,t) \ \vee \ \neg B(b,i,j,t+1) \ \vee \ \neg P(i-1,j,t) \ \vee \ \neg P(i,j,t+1)$$

  if $i = 0$:

  $$B(b,i,j,t) \quad \rightarrow \quad \neg B(b,i+1,j,t+1)$$

  $$\neg B(b,i,j,t) \ \vee \ \neg B(b,i+1,j,t+1)$$

  This can be extended to other directions and boxes.

2.3.7 Goal conditions

At time T every box must be in some goal. Hence the following clause must hold for every box:

$$\bigvee_{(i,j)}^{goals} B(b,i,j,T)$$

where goals is the set of positions of cells marked with 'G'.

2.4 Decoding process

The assignment returned by the SAT solver will be a list of integers, positive integer denotes the variable being true and negative means false. As mentioned in the variable encoding, two propositional variables have been used, the integers having magnitude less than 100000 represent the variable used for player and greater than 100000 represent the variable used for box. We consider all the positive integers less than 100000 in the list and get the player position (i,j) at time t as follows: Let the integer be i, then the least significant two digits represent the time, so
`t = i%100`
The third least significant digit is the y coordinate so,
`y = (i // 100) % 10`
The fourth least significant digit is the x coordinate so,
`x = (i // 1000) %10` We then store them in a list with the list [x,y] stored at index t. Thus we obtain a list with indices from 0 to T. We initialize a list (`dir`) to store the directions of the moves made by the player. The move made by the player at time t can be calculated by subtracting the x and y coordinates at time t-1 from the x and y coordinates at time t respectively. We do this for $1 \leq t \leq T$ and each time store the value as a tuple and look for the same in the `DIRS` dictionary given. If the value is found we append the key corresponding to that value to `dir`. This will give us a solution for the Sokoban grid if it is solvable

3 Contributions

3.1 Question 1

Rishika Dhiman (24B1090) and Uma Kumari (24B1026)
Both contributed equally.

3.2 Question 2

Rishika Dhiman (24B0190)

3.3 Report

Rishika Dhiman (24B0190) and Uma Kumari (24B1026)
Both contributed equally.