Logic Assignment-1 Report

Krishna Sai Shishir Vuppala(24b0973), Aaditya Kumar(24b0975)

August 2025

Question 1 - Sudoku Solver

We must encode the rules of Sudoku into propositional rules, and then the SAT Solver will be able to find us a valid assignment, which can then be converted into the answer by decoding it.

Variable Encoding

We introduce propositional variables of the form

$$p_{i,j,n} = \text{cell in } i^{th} \text{ row and } j^{th} \text{ column contains the number } n$$

where

- $i \in \{0, 1 \dots 8\}$
- $j \in \{0, 1 \dots 8\}$
- $n \in \{1, 2 \dots 9\}$

The variable assignment is implemented within the code using:

$$\texttt{var\_generator}(i, j, n) \texttt{ = } 90 \cdot i + 10 \cdot j + n$$

This variable assignment gives a unique ID for each possible tuple of (i, j, n).

Approach Overview

The rules required for the game of sudoku are:-

1. Each cell should have exactly 1 number.
2. Each row should have one of each number from 1 to 9.
3. Each column should have one of each number from 1 to 9.
4. Each subbox should have one of each number from 1 to 9.

Apart from these, we also have to encode in the original conditions of whichever Sudoku we have.
So we generate the following clauses for each of the rules:-

Cell Constraints: This is encoded in two parts

Atleast one number per cell: In the cell at i^{th} row and j^{th} column

$$p_{i,j,1} \vee p_{i,j,2} \cdots p_{i,j,9}$$

At most one number per cell: In the cell at i^{th} row and j^{th} column, for all $n_1 \neq n_2$

$$\neg p_{i,j,n_1} \vee \neg p_{i,j,n_2}$$

Row Constraint: Each row needs to have exactly one of each number. So for each number n and for a row i,

$$p_{i,0,n} \vee p_{i,1,n} \cdots p_{i,8,n}$$

Column Constraint: Each column needs to have exactly one of each number. So for each number n and for a column j,

$$p_{0,j,n} \vee p_{1,j,n} \cdots p_{8,j,n}$$

Subbox constraint: Each 3x3 subbox needs to have exactly one of each number. So for a number n and for the subbox with index (p, q)

$$(\bigvee_{r,s \in \{1,2,3\}} p_{3p+r,3q+s,n})$$

Initial Conditions: In the sudoku we need to solve, if the cell in i^{th} row and j^{th} column has the number n. Then,

$$(p_{i,j,n})$$

should be added as a clause

Note: We don't have to add "at most 1 number" conditions for the rows, columns, and subbox because there are exactly 9 numbers and 9 places, and we already have the condition that makes sure there is exactly one number per cell.

Decoding

There will be exactly 81 propositional variables that are true in the final satisfying assignment, each corresponding to a number in a particular cell. So we get the encoded number for each of them, and we perform.

```
i = (element//90)
j = (element//10)%9
n = element%10
```

then within the grid at the i^{th} row and j^{th} column we place the number n.

Question 2 - Sokoban Solver

We must encode the rules of Sokoban into propositional clauses, and then the SAT Solver will be able to find us a valid assignment, which can then be converted into the answer by decoding it. Unlike Sudoku, which has obvious conditions, translating the rules of the game into the required propositional clauses takes longer

Variable Encoding

We introduce propositional variables of the form

$$p_{x,y,t} = \text{player is at cell (x,y) at time } = t$$

$$b_{b,x,y,t} = \text{box } b \text{ is at cell (i,j) at time } t$$

where

- $x \in \{1, 2 \dots M\}$
- $y \in \{1, 2 \dots N\}$
- $t \in \{0, 1 \dots T\}$
- $b \in \{0, 1 \dots B-1\}$

Here, B is the total number of boxes. The variable assignment is implemented within the code using:

$$\texttt{var\_player}(x, y, t) \texttt{ = } 1000 \cdot t + 21 \cdot y + x$$
$$\texttt{var\_box}(b, x, y, t) \texttt{ = } 100000 \cdot (b+1) + 1000 \cdot t + 21 \cdot y + x$$

This variable assignment gives a unique ID for each possible tuple of (x, y, t) and (b, x, y, t).

Approach Overview

The rules/conditions we encoded for the game of Sokoban are:-

1. Player should be at exactly one position at any particular time t.
2. Any box should be at only one position at any particular time t.
3. Player and walls cannot be at the same position at any particular time t.
4. Boxes and walls cannot be at the same position at any particular time t.
5. Player and boxes cannot be at the same position at any particular time t.
6. Boxes and other boxes cannot occupy the same position at any particular time t.

7. If the player is at (x, y) at time t, then at the next time $t + 1$, he can be at (x+1,y) or (x-1,y) or (x,y+1) or (x,y-1).

8. If any box b is at (x, y) at time t, and the player moves to (x, y) at time $t + 1$, then the box should move in the same direction that the player moved as long as it's possible.

9. If any box b is at (x, y) at time t, and the player DOES NOT move to (x, y) at time $t + 1$, then the box should remain in the same place at time $= t + 1$.

10. At the final time T, all the boxes should be in goal states.

Apart from these, we also have to encode the original conditions of the game. Thus, we need to encode the original position of the boxes and player as well as read the grid and find all the locations of the walls and goals.
So we generate the following clauses,

Initial conditions We have to encode in the original conditions for the player and the boxes.

Player: Find the initial position of the player while iterating through the grid, let it be (x, y) then,

$$p_{x,y,0}$$

Boxes: Find all the positions of boxes while interating through the grid, and for each position choose a new index b and append the clause,

$$b_{b,x,y,0}$$

Uniqueness: Player and the boxes must be at exactly one position at any time t.

Player: We will do this in two parts:

Player is present at atleast one position: For all positions in the grid, and for each time t, we add

$$\bigvee_{1 \leq x \leq M, 1 \leq y \leq N} p_{x,y,t}$$

Player is present at not more than one position: For any two positions(x_1, y_1) and (x_2, y_2) we add at time t.

$$\neg p_{x_1,y_1,t} \vee \neg p_{x_2,y_2,t}$$

Box: We will do this in two parts:

Any box b is present at atleast one position: For all positions in the grid, and for each time t, we add

$$\bigvee_{1 \leq x \leq M, 1 \leq y \leq N} b_{b,x,y,t}$$

Any box b is at not more than one position: For any two positions(x_1, y_1) and (x_2, y_2) we add,

$$\neg b_{b,x_1,y_1,t} \vee \neg b_{b,x_2,y_2,t}$$

No overlap: We need to make sure player and boxes cant be present in same cell as each other or in the same cell as a wall. Let all the positions of the wall be stored in a list wallpositions.

Player and walls: for each (x_i, y_i) in wallpositions and at every time t, we get,

$$\neg p_{x_i,y_i,t}$$

Boxes and walls: for each (x_i, y_i) in wallpositions and at every time t for every box index b, we get,

$$\neg b_{b,x_i,y_i,t}$$

Player and boxes For any cell in the grid, at any time t and for any box index b.

$$\neg p_{x,y,t} \vee \neg b_{b,x,y,t}$$

Boxes and boxes: For any cell in the grid, at any time t and for two unique box indices $b_!, b_2$. We have,

$$\neg b_{b_1,x,y,t} \vee \neg b_{b_2,x,y,t}$$

Move conditions: We need to encode the move conditions of the game, like player moving and box pushing, as clauses.

Player movement: if the player is at (x, y) at time t, he should be at $(x + 1, y)$ or $(x - 1, y)$ or $(x, y + 1)$ or$(x, y - 1)$ at time $t + 1$. So we add,

$$\neg p_{x,y,t} \vee p_{x+1,y,t+1} \vee p_{x-1,y,t+1} \vee p_{x,y+1,t+1} \vee p_{x,y-1,t+1}$$

Note: to cover edge cases we make sure to check if $(x+1, y)$, $(x-1, y)$, $(x, y + 1)$ and$(x, y - 1)$ are valid positions before appending them in the clause.

Box movement: This is done in three steps

Player allowed to push: if a box b is at (x, y) at time $= t$ then at time $= t + 1$ player is at (x, y), then the box must get displaced from its original position depending on where the player was in the previous turn.

$$\neg p_{x-1,y,t} \vee \neg p_{x,y,t+1} \vee \neg b_{b,x,y,t} \vee b_{b,x+1,y,t+1}$$

$$\neg p_{x+1,y,t} \vee \neg p_{x,y,t+1} \vee \neg b_{b,x,y,t} \vee b_{b,x-1,y,t+1}$$

$$\neg p_{x,y+1,t} \vee \neg p_{x,y,t+1} \vee \neg b_{b,x,y,t} \vee b_{b,x,y-1,t+1}$$

$$\neg p_{x,y-1,t} \vee \neg p_{x,y,t+1} \vee \neg b_{b,x,y,t} \vee b_{b,x,y+1,t+1}$$

Note: to cover edge cases, we make sure to check if $(x + 1, y)$, $(x - 1, y)$, $(x, y + 1)$, and$(x, y - 1)$ are valid positions before appending them in the clause.

Preventing illegal moves: Player shouldn't be allowed to push the box if the box is on the edge and the player is pushing it out of the box. so for all y, we get,

$$\neg p_{M-1,y,t} \vee \neg b_{b,M,y,t} \vee \neg p_{M,y,t+1}$$

$$\neg p_{2,y,t} \vee \neg b_{b,1,y,t} \vee \neg p_{1,y,t+1}$$

for all x, we get,

$$\neg p_{x,N-1,t} \vee \neg b_{b,x,N,t} \vee \neg p_{x,N,t+1}$$

$$\neg p_{x,2,t} \vee \neg b_{b,x,1,t} \vee \neg p_{x,1,t+1}$$

No box movement: If a box b is at (x, y) and no player is at (x, y) at time $= t$. Then we know that no pushing should have happened, and so at time $= t+1$, also box is on the same position.

$$\neg b_{b,x,y,t} \vee p_{x,y,t} \vee b_{b,x,y,t+1}$$

Final condition: at time T each box b should be at a goal state. Let all the positions of the goals be stored in goalpositions. Thus for each box b, and for all (x_i, y_i) in goalpositions.

$$b_{b,x_1,y_1,T} \vee b_{b,x_2,y_2,T} \ldots \vee b_{b,x_n,y_n,T}$$

Decoding

Since we only care about the moves that were done, we only need to worry about the player's position. For every time t, there will be one position where the player is located.

So for every time from $t = 1$ to T, we store the previous position of the player in a variable(for t = 1 we store the initial position of the player in the variable) and we check for which (x, y) is $p_{x,y,t}$ true(i.e. check if var_player(x, y, t) is true in the satisfying assignment). Then by comparing (x, y) i.e., the current state with the previous position of the player to figure out which direction he went in, and then append that move in a list of all the moves.

Distribution of Workload

Question 1

Both Aaditya and I thought of the logic and coded it up independently. Our codes were of similar speed, and we worked equally on this problem.

Question 2

Aaditya and I discussed the problem together to come up with the logic of how exactly to model this problem. Then I was the one to code it up on my laptop. Finally, I went through the code and wrote up some custom test cases to debug the code properly.

Report

I was the one who wrote the report, and Aaditya performed any revisions to remove any errors in the LaTeX file.