Logic$_A ssignment 1$

Shreshta Bathini and Yashwanth Murasani Reddy

August 2025

Introduction

This report presents two puzzle solvers based on SAT (Boolean Satisfiability) techniques: a Sudoku solver and a Sokoban solver.

Sudoku is a classic number-placement puzzle where each row, column, and 3x3 box must contain all digits from 1 to 9 without repetition. We encode these constraints as Boolean formulas and solve them using a SAT solver, transforming the puzzle into a satisfiability problem.

Sokoban is a spatial puzzle involving pushing boxes to target locations on a grid while navigating walls and limited movement. We model player and box positions over time as propositional variables, encode movement and pushing rules into CNF, and use a SAT solver to find valid sequences of moves solving the puzzle.

Both solvers showcase the power of SAT encoding to translate complex combinatorial problems into formal logic solved efficiently by modern SAT solvers.

1 Sudoko Solver

1.1 About sat solver

Using a sat solver given some intial encoding ewe can decode and provide a satisfiable assignment to our sudoko problem in real time

1.2 Sudoko rules as CNF formulas

We have certain rules for solving sudoko that should append as sudoko solver

Sudoku rules in CNF form are:

1. Each cell contains at least one digit:

$$\bigvee_{k=1}^{9} x_{i,j,k} \quad \forall i, j = 1, \ldots, 9$$

2. Each cell contains at most one digit:

$$(\neg x_{i,j,k} \vee \neg x_{i,j,l}) \quad \forall i, j = 1, \ldots, 9, \quad \forall k \neq l \in \{1, \ldots, 9\}$$

3. Each digit appears at least once in each row:

$$\bigvee_{j=1}^{9} x_{i,j,k} \quad \forall i, k = 1, \ldots, 9$$

4. Each digit appears at most once in each row:

$$(\neg x_{i,j,k} \vee \neg x_{i,m,k}) \quad \forall i, k = 1, \ldots, 9, \quad \forall j \neq m \in \{1, \ldots, 9\}$$

5. Each digit appears at least once in each column:

$$\bigvee_{i=1}^{9} x_{i,j,k} \quad \forall j, k = 1, \ldots, 9$$

6. Each digit appears at most once in each column:

$$(\neg x_{i,j,k} \vee \neg x_{m,j,k}) \quad \forall j, k = 1, \ldots, 9, \quad \forall i \neq m \in \{1, \ldots, 9\}$$

7. Each digit appears at most once in each 3x3 block:

$$(\neg x_{r,s,k} \vee \neg x_{u,v,k}) \quad \forall k = 1, \ldots, 9, \text{ for distinct } (r, s), (u, v) \in B_{I,J}$$

where

$$B_{I,J} = \{(i, j) \mid i \in \{I, I+1, I+2\}, j \in \{J, J+1, J+2\}\} \quad I, J \in \{1, 4, 7\}$$

1.3 Approach Overview

To convert the Sudoku puzzle into a SAT problem, we introduce propositional variables $x_{i,j,k}$ which are true if the cell at row i, column j contains the digit k. The Sudoku rules are then expressed as logical clauses on these variables forming a CNF formula.

Specifically, constraints ensure that each cell contains exactly one digit, each digit appears exactly once in each row, column, and 3x3 subgrid. These constraints are translated into Conjunctive Normal Form clauses, which a SAT solver can process to find a satisfying assignment representing a valid Sudoku solution.

This transformation enables solving Sudoku puzzles by leveraging efficient SAT solver algorithms.

1.4 Variable Descriptions

The key variables used in the Sudoku SAT solver implementation are described below:

- `literal(i, j, num)`: A function that encodes the propositional variable representing the assignment of digit `num` to the cell at row `i` and column `j`. It returns a unique integer variable identifier used in CNF clauses.
- `position(lit)`: Given a SAT solver variable `lit`, this function decodes the variable back to the Sudoku cell coordinates (i, j) and digit `num` it represents. This is used to interpret the SAT solver's solution.
- `grid`: The input 9x9 Sudoku grid represented as a list of lists, where each element is an integer from 0 to 9. A zero denotes an empty cell to be solved.
- `clauses`: A CNF formula instance from PySAT that stores all the CNF clauses representing Sudoku constraints and the initial puzzle conditions.
- `s`: The PySAT SAT solver instance used to add clauses and solve the CNF formula.
- `sol`: The list of integers representing the satisfying assignment of SAT variables found by the solver that corresponds to a valid Sudoku solution.
- `new_grid`: The output 9x9 solved Sudoku grid reconstructed by decoding positive SAT variables from `sol` back into digit assignments in each cell.

These variables together enable the translation of the Sudoku puzzle constraints into CNF SAT clauses, their solution by the SAT solver, and conversion of the solver's output back into a human-readable Sudoku solution grid.

2 Grading Assignment gone wrong

2.1 Variable Representation in Sokoban SAT Solver

The SAT encoding of the Sokoban puzzle uses the following propositional variables to represent the dynamic state of the game over time:

- **Player position variable:**

  $$\texttt{var\_player}(x, y, t)$$

  This variable is true if and only if the player occupies the position (x, y) on the grid at time step t.

- **Box position variable:**

  $$\texttt{var\_box}(b, x, y, t)$$

  This variable is true if and only if the box indexed by b is located at position (x, y) at time step t.

- **Walls:** Wall positions are fixed and represented as a set of grid coordinates $\{(x, y)\}$ where neither the player nor any box can occupy at any time. These positions are encoded as constraints forbidding variables from being true at these positions.

- **Goals:** Goal positions are grid coordinates where boxes must be moved. In the final time step T, constraints enforce that all boxes occupy these goal cells.

The variables are indexed by spatial coordinates (x, y) referring to the grid cells, the box index b to differentiate boxes, and the time step t to model the progression of moves over time.

These variables form the encoding foundation for expressing Sokoban puzzle rules and movement restrictions in propositional logic suitable for SAT solving.

2.2 Translation of Sokoban Rules into CNF Constraints

The Sokoban puzzle's rules are encoded into Conjunctive Normal Form (CNF) constraints to be processed by a SAT solver. The key aspects of this encoding include:

- **Initial State Constraints:** Clauses assert the player and boxes' initial positions at time step zero. This fixes the starting configuration of the puzzle.

- **Existence Constraints:** For every time step, the player must occupy exactly one position on the grid, and each box must occupy exactly one position. These constraints ensure that the player and boxes do not "disappear" or occupy multiple spots simultaneously.

- **Grid Boundary and Wall Constraints:** Positions corresponding to walls are forbidden for the player and boxes at all times, encoded by clauses negating any variable that places an entity on a wall cell.

- **Non-overlapping Constraints:** No two boxes can occupy the same cell simultaneously, and the player cannot share a cell with any box, ensuring unique occupancy of cells.

- **Player Movement Constraints:** Player movement is constrained to valid moves between adjacent cells (up, down, left, right), considering grid boundaries and blocking entities. Clauses enforce that at time t, the player's position must be reachable from a neighboring position at time $t - 1$.

- **Box Movement and Pushing Logic:** Boxes can move only when pushed by the player from an adjacent cell. Clauses model this by requiring that for a box to be at position (x, y) at time t, the player must be at the appropriate adjacent location at time $t - 1$ to push it. The destination cell of the box must be free, respecting walls and other boxes.

- **Consistency Constraints:** Ensure that if a box moves, the player moves correspondingly to the pushing position. Similarly, if a box stays still, movement constraints restrict player positions accordingly.
- **Goal Conditions:** At the final time step T, clauses enforce that all boxes occupy predefined goal cells, representing the puzzle's solution criteria.

This systematic encoding captures the dynamics and restrictions of Sokoban gameplay into a SAT problem. The resulting CNF formula can then be fed into a SAT solver which produces a variable assignment corresponding to a valid solution sequence of moves if one exists.

2.3 Function Explanations in Sokoban SAT Solver

2.3.1 SokobanEncoder Class

`__init__`**(self, grid, T)**
Initializes the encoder with the Sokoban grid, maximum allowed steps T, grid dimensions, and parses initial positions of the player, boxes, goals, and walls.

`_parse_grid`**(self)**
Parses the input grid character matrix to locate the initial coordinates of the player (`P`), boxes (`B`), goals (`G`), and walls (`#`).

`var_player`**(self, x, y, t)**
Given grid coordinates (x, y) and timestep t, returns the unique SAT variable identifier representing the player's presence at that cell and time. It returns a default variable if position is out of bounds.

`var_box`**(self, b, x, y, t)**
Given box index b, position (x, y), and timestep t, returns the unique SAT variable ID representing box b at that position and time. Returns an error/out-of-range variable if indices exceed grid limits.

`encode`**(self)**
Constructs the propositional CNF clauses corresponding to:

Initial placement of player and boxes:
Assigned true to the events of player and boxes occupying the given positions at time t=0.

Player and boxes occupying exactly one cell per timestep:
Encoded the logic that at least one position is occupied and not more that one position is occupied by player and boxes.

Wall, box, and player collision avoidance:
Encoded the logic to avoid the collision of walls with boxes and player,collision

among the boxes and collision of boxes with player.

Possible player moves and box pushes frame-by-frame:
Added constraints to define the player movements using the fast that if a player exits at a position at the time t(t!=0) he must have not moved or come from the neighbors.
Same logic of player is applied to boxes but an extra logic is included , if a box is moved in a particular direction by a unit the player must have moved in the same direction by one unit to get to the previous position of the box enforcing box pushes are paired with adjacent player positions..

Goal conditions:
Checked if there exist a possible state achieved with the conditions satisified such that there exist a box at each position of the goal.

Returns the complete CNF formula for SAT solving.

2.3.2 Decoding Functions

`decode(model, encoder)`
Converts the SAT solver's satisfying assignment (list of variable assignments) back into a human-readable list of player moves. It extracts the player's coordinates for each timestep from positive variables, then deduces the moves (`U`, `D`, `L`, `R`) by comparing sequential positions.

`solve_sokoban(grid, T)`
Main function invoking the encoding, solving the CNF formula using PySAT's solver, and decoding the solution. Returns the sequence of moves solving the Sokoban puzzle or `-1` if unsatisfiable.

2.4 Decoding Process

After the SAT solver finds a satisfying assignment for the CNF formula, the solution must be interpreted to recover the sequence of player moves completing the Sokoban puzzle.

The decoding works as follows:

- The assignment is a list of integers where positive integers represent variables assigned `True`, indicating the player's or a box's presence at certain coordinates and time steps.
- The decoding function scans through these positive variables to record the player's position (x, y) at each time step t.
- To extract player positions, the variable numbering scheme is inverted: given a positive variable, its encoded indices (x, y, t) are computed by arithmetic inversions based on the variable encoding formula.

- By collecting the player's positions over all time steps, the decoder computes the player's move directions by comparing consecutive positions:

$$\begin{cases} (x_t = x_{t-1} + 1) & \rightarrow \text{Down ('D')} \\ (x_t = x_{t-1} - 1) & \rightarrow \text{Up ('U')} \\ (y_t = y_{t-1} + 1) & \rightarrow \text{Right ('R')} \\ (y_t = y_{t-1} - 1) & \rightarrow \text{Left ('L')} \end{cases}$$

- This produces an ordered list of moves $\{U, D, L, R\}$ that represent the solution path for the player from the initial configuration to the goal state.

This method provides a readable sequence of player moves interpretable as the solution to the original Sokoban puzzle derived from the SAT solver's variable assignment.

Division of Work and Individual Contributions

3 Question1

3.1 24B1039

I was responsible for defining the encoding scheme that mapped Sudoku cells and digits to propositional SAT variables. Their work included:

- Implementing the `literal` function, which generates a unique integer variable identifier for each $(row, column, digit)$ triplet.
- Writing the helper `position` function to decode SAT variables back to Sudoku coordinates and digits.
- Constructing CNF clauses that enforce:
  - Each cell contains at least one digit.
  - Each cell contains at most one digit.
  - Clauses fixing pre-filled (given) digits in the input Sudoku grid.

This groundwork ensured a consistent and sound base for representing Sudoku puzzles as CNF formulas.

3.2 24B1085

I focused on further encoding the classical Sudoku constraints and integrating the SAT solver to find solutions. Their tasks included:

- Adding CNF clauses that impose the uniqueness of digits in rows, columns, and 3x3 sub-boxes.

- Utilizing the PySAT solver to add clauses, run the solver, and check satisfiability.
- Implementing the decoding logic to interpret the satisfying assignment from the solver and reconstruct the solved Sudoku grid.
- Handling edge cases such as no solution scenarios.

Constructing CNF clauses that enforce:

- Each cell contains at least one digit.
- Each cell contains at most one digit.
- Clauses fixing pre-filled (given) digits in the input Sudoku grid.

Together, these contributions resulted in a complete and functional Sudoku solver using SAT technology, demonstrating teamwork in both problem formulation and solution extraction phases and I have also written the entire report.

4 Question2

The development of the Sokoban SAT solver was split into two key components, each managed by a different collaborator to ensure a balanced workload and clear responsibilities.

4.1 24B1085

I focused on formulating the SAT encoding of the Sokoban puzzle, implementing:

- The parsing of the Sokoban grid into player start position, box positions, wall locations, and goal cells.
- The variable encoding schema for player and box positions at each timestep to unique SAT variables.
- Construction of CNF constraints, including:
  - Initial conditions for player and box placement.
  - Existence and uniqueness constraints ensuring the player and boxes occupy exactly one cell each timestep.
  - Collision avoidance clauses preventing overlapping of player, boxes, and walls.

4.2 24B1039

I handled the dynamic movement and goal-related logic, including:

- Encoding player movement possibilities respecting adjacency and boundaries.
- Modeling box movements and pushing actions linked to player moves via CNF clauses.
- Implementing goal state constraints requiring boxes to occupy targets at the final timestep.
- Developing the decoding algorithm to map the satisfying CNF assignment back to a readable sequence of player moves (Up, Down, Left, Right).
- Integrating the SAT solver invocation, managing solvability checks, and formatting the final solution output.

This division fostered a clear project structure allowing parallel work streams on static encoding and dynamic solving portions of the Sokoban SAT solver.