DS2020-Artificial intelligence

Sudoku Solver using Pycosat

Authour : Sumit Dnyaneshwar Garad (1123010320)
Course : Introduction to Artificial intelligence (DS2030)

Overview

This program solves Sudoku puzzles using a **SAT (Boolean Satisfiability) Solver** called **PycoSAT**. The main idea is to represent Sudoku rules as a **Conjunctive Normal Form (CNF)** and use PycoSAT to find a valid solution.

- The program reads Sudoku puzzles from a file (`p.txt`)
- Converts the puzzle into a SAT problem
- Solves it using PycoSAT, and
- Writes the solution to an output file (`Solved_sudoku.txt`).

Approach

Convert Sudoku rules into **CNF constraints**.Use a unique variable encoding (`var()`).Solve the CNF using PycoSAT**.**If a solution is found, decode it into a Sudoku grid.If no solution exists, return `None`.Store the solution in a file **.**Each solved Sudoku is written to `Solved_sudoku.txt`

1. `__init__(self, sudoku_lists, sudoku_no)`

- Initializes the `Solver` object with a given Sudoku puzzle.
- `sudoku_lists`: A list of Sudoku puzzles (strings read from a file).
- `sudoku_no`: The index of the Sudoku puzzle to solve.

- Selects the Sudoku puzzle from `sudoku_lists` using `sudoku_no`.
- Initializes an empty list `self.cnf` to store **CNF constraints**

---

2. `var(self, row, col, num)`

- Generates a **unique variable ID** for each cell in the Sudoku grid.
- `row`: Row index (0 to 8).
- `col`: Column index (0 to 8).
- `num`: Number (1 to 9)
- A unique integer for each `(row, col, num)` combination using: ID=(row×81)+(col×9)+num.This unique ID helps in encoding Sudoku rules as CNF constraints.

---

3. `get_row(self, pos)`

- Returns all numbers present in the row of the given position.
- `pos`: A tuple `(row, col)` representing a cell's position.
- A list of numbers present in the row (excluding `.`).
- Iterates through all 9 columns in the given row.
- If a cell is not empty (`!= '.'`), adds it to the list.

---

4. `get_coloumn(self, pos)`

- Returns all numbers present in the column of the given position.
- `pos`: A tuple `(row, col)`.
- A list of numbers present in the column (excluding `.`).
- Iterates through all 9 rows for the given column index.
- If a cell is not empty (`!= '.'`), adds it to the list.

---

5. `get_square(self, pos)`

- Returns all numbers present in the **3×3 sub-grid** that contains the given position.
- `pos`: A tuple `(row, col)`.
- A list of numbers in the 3×3 sub-grid (excluding `.`).
- Finds the **top-left corner** of the 3×3 sub-grid.
- Iterates over the 3×3 block, collecting numbers that are not `.`.

---

6. `generate_cnf(self)`

Generates the **CNF constraints** for the Sudoku puzzle.Following rules are used to generate **CNF** for each variable generated through the function var(row, column, num)

1. **Each cell contains at least one number:**
    - Adds CNF clauses ensuring that every cell contains at least one number between **1 and 9**.
2. **Each cell contains at most one number:**
    - Adds CNF clauses ensuring that no cell contains more than one number.
3. **Each row must have unique numbers (1-9):**
    - Uses `get_row()` to check which numbers are already present.
    - If a number is missing, adds CNF constraints ensuring its presence.
4. **Each column must have unique numbers (1-9):**
    - Uses `get_coloumn()` similarly to enforce column constraints.
5. **Each 3×3 sub-grid must have unique numbers (1-9):**
    - Uses `get_square()` to ensure numbers appear only once per sub-grid.
6. **Pre-filled numbers are fixed:**
    - For cells that already contain numbers, CNF constraints force them to retain their values.

---

7. `solve(self)`

- Uses **PycoSAT** to solve the encoded CNF and returns the solved Sudoku.
- Calls `generate_cnf()` to build CNF constraints.
- Passes the CNF to `pycosat.solve()`
- If the puzzle is **unsolvable**, returns `None`.

- Otherwise, decodes the solution using `decode_solution()`.

---

8. `decode_solution(self, solution)`

- Converts the SAT solver's output into a readable Sudoku grid.
- `solution`: The output from PycoSAT (list of integers representing true variables).
- A **string** representing the solved Sudoku.
- Initializes an empty list of 81 numbers(initially all are set as '.' which represent empty number)
- Iterates over **positive** numbers in `solution`:
- Extracts row, column, and number from the variable ID.
- Places the number in the correct grid position.
- Converts the grid into a single string string using join function and return it.

Main Execution Flow (`if __name__ == '__main__':`)

- **Reads the Sudoku puzzles** from `p.txt`.
- **Clears the output file (`Solved_sudoku.txt`)**.
- Iterates over each Sudoku puzzle and Creates a `Solver` object.
- Calls `solve()` to get the solution.
- Writes the solution to `Solved_sudoku.txt`.
- If no solution exists, writes `"No solution found"`.
  Convert Sudoku rules into **CNF constraints**.Use a unique variable encoding (`var()`).Solve the CNF using PycoSAT**.**If a solution is found, decode it into a Sudoku grid.If no solution exists, return `None`.Store the solution in a file **.**Each solved Sudoku is written to `Solved_sudoku.txt`