DS2020 Introduction to Artificial Intelligence

Lab 4 - Sudoku SAT Solver

Roll No.	*Name*
112301013	K V S Bharath
142301013	K Srirama Srikar

---

0. Executing the python file

Run the `main.py` python file, if need be to use a file (other than `p.txt`) as input, go to the `if __name__=="__main__"` conditional statement at the end of the program and make the fucntion call something similar to `solve_sudoku(filename)`. Upon executing the python file we get an output stating `Solutions written to output.txt`.
Refer to `output.txt` to check the solutions.

---

1. Encoding Sudoku as a Boolean SAT Problem

Each Sudoku grid is a **9×9** matrix, where each cell contains a number from **1 to 9**. To formulate this as a SAT problem, we define **Boolean variables** to represent possible values in each cell.

Boolean Variable Representation
Define a Boolean variable $X_{r,c,v}$, where:
- r = Row index (0 to 8)
- c = Column index (0 to 8)
- v = Digit (1 to 9)

and, $X_{r,c,v}$ is a Boolean variable that is **True** if digit v is placed in cell (r, c).

Variable Encoding: Each variable $X_{r,c,v}$ is encoded as a single integer using the formula:
$X_{r,c,v} = 81(r) + 9(c) + v$
This ensures each variable is uniquely represented by a number between 1 and 729.

Variable Decoding: Each variable is decoded as a tuple of three integers by using the formula:
$(r, c, v) = ((var - 1)//81, ((var - 1)\%81)//9, (var - 1)\%9 + 1)$
Where var is the variable encoding for an any value in any row and any column.

---

2. Generating CNF Clauses

We create six sets of CNF clauses:

1. **Each cell contains at least one value**
   $(X_{r,c,1} \lor X_{r,c,2} \lor ... \lor X_{r,c,9})$
   For every (r, c), we generate a clause ensuring at least one number is assigned.

2. **Each cell contains at most one value**
   $(\neg X_{r,c,v} \lor \neg X_{r,c,w})$
   For every (r, c) and for all pairs $v \neq w$, we generate clauses preventing multiple numbers in a single cell.

3. **Each row contains all values**
   $(X_{r,0,v} \lor X_{r,1,v} \lor ... \lor X_{r,8,v})$
   For every r and v, we generate a clause ensuring that each number appears in every row.

4. **Each column contains all values**
   $(X_{0,c,v} \lor X_{1,c,v} \lor ... \lor X_{8,c,v})$
   For every c and v, we generate a clause ensuring that each number appears in every column.

5. **Each 3×3 subgrid contains all values**
   For each block $(block_r, block_c)$, and for each value $v \in \{1, 2, ..., 9\}$, we enforce the constraint:
   $\forall (r_1, c_1), (r_2, c_2) \in \text{Block}, (r_1, c_1) \neq (r_2, c_2) : \neg X_{r_1,c_1,v} \lor \neg X_{r_2,c_2,v}$
   and we end up with the below condition for a particular block
   $\bigwedge_{(r_1,c_1)\neq(r_2,c_2)} (\neg X_{r_1,c_1,v} \lor \neg X_{r_2,c_2,v})$
   This ensures that the same number v does not appear twice in any 3×3 block. Thus, as a 3×3 block has 9 values and we do not have any repetitions, we end up having all the values in a block.

6. **Fixed values from the puzzle input**
   If a cell at (r, c) already contains a number v, we directly add: $(X_{r,c,v})$

---

3. Solving the CNF with pycosat

We use `pycosat` to solve the generated CNF in the `solve_sudoku()` function
The logic is something similar to the below code

```
import pycosat

solution = pycosat.solve(cnf_clauses)

if solution == "UNSAT":
    print("No solution exists")
```

```python
else:
    print(solution)
```

Note that we have defined an additional `is_valid` function to ckeck the validity of sudoku solution which requires the use of `numpy`. The way to use that function is mentioned in the `solve_sudoku` function at around line 152.