Homework 2: Counting Sudoku solutions by using decision diagrams

> Practice/Real-Life Applications of Computational Algorithms, Spring 2021
>
> 0710006 Ke-Yu Lu

Command Line

```
python3 main.py [INPUT_FILE] [OUTPUT_FILE]
```

Implementations

1. Read the input file, and then build the SAT in CNF.
2. Encode the puzzle into CNF with n^3 variables with $O(n^4)$ clauses.
    - `X[r][c][v]` means whether the cell in (r, c) is v or not.
    - For each number, it can only appear in each row, column, and block.
    - Use `OneHot()` to get the clauses for variables in which only one variable can be assigned true.
3. Use `satisfy_count()` in `pyeda` to calculate the number of solution.

Below is the main code of the solver.

```python
def solve(Puzzle):
    n = len(Puzzle)
    sq = int(math.sqrt(n))
    X = exprvars('x', (1, n + 1), (1, n + 1), (1, n + 1))
    # each cell can has only one value
    V = And(*[
        And(*[
            OneHot(*[X[r, c, v] for v in range(1, n + 1)])
            for c in range(1, n + 1)
        ]) for r in range(1, n + 1)
    ])
    # each value can only appear once in each row
    R = And(*[
            And(*[
                OneHot(*[X[r, c, v] for c in range(1, n + 1)])
                for v in range(1, n + 1)
            ]) for r in range(1, n + 1)
        ])
        # each value can only appear once in each column
        C = And(*[
            And(*[
                OneHot(*[X[r, c, v] for r in range(1, n + 1)])
                for v in range(1, n + 1)
            ]) for c in range(1, n + 1)
        ])
        # each value can only appear once in each sq x sq block
        B = And(*[
            And(*[
                OneHot(*[
                    X[sq * br + r, sq * bc + c, v] for r in range(1, sq + 1)
                    for c in range(1, sq + 1)
                ]) for v in range(1, n + 1)
            ]) for br in range(sq) for bc in range(sq)
        ])
        # the assigned cell
        P = And(*[
            X[r + 1, c + 1, Puzzle[r][c]]
            for r, c in itertools.product(range(n), repeat=2) if Puzzle[r][c] > 0
        ])
        # And all clauses
        Fun = And(V, R, C, B, P)
        # return the number of solutions
        return Fun.satisfy_count()
```

Results

Input File	Time (s)
sudoku_4x4_9.txt	0.07
sudoku_9x9_1.txt	0.3
sudoku_9x9_125.txt	0.3
sudoku_16x16_1.txt	2.1