Homework 3 – Logic and SAT

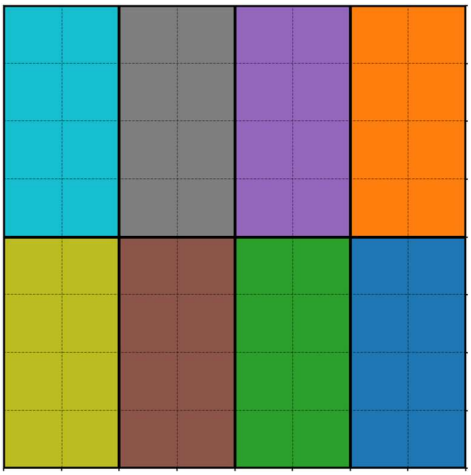
Introduction

In this exercise, you will create a solver for an advanced variation of the popular game sudoku, using the DPLL algorithm learned in class.

Basic Rules

Sudoku is a game where you need to fill a board with numbers 1-N, following certain rules. The board is usually size 9x9, but we will solve boards of (possibly) different sizes, so the boards will be size NxN for some number N.

Inside the board there are rectangles of area N, with sides of lengths KxL. The board is naturally divided into LxK rectangles. N = L*K. Here is an illustration of such a partition:



This is an 8x8 board, with rectangles of 4x2 (first number represents the number of rows).

The following rules apply to any correct assignment of numbers:

- A number never appears twice in the same row.
- A number never appears twice in the same column.
- A number never appears twice in the same rectangle.
- Every square has exactly one number in it.

Every board will also include some squares that are already assigned to some numbers.

Additional Rule

In addition to the board, you will also receive pairs of adjacent squares on the board, and their

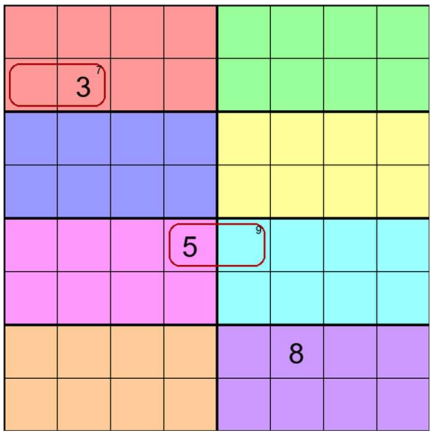
expected sum. You need to ensure that the sum of the numbers in these squares equals the expected sum. For example, if you get an input of (0,0,1,0,9) this indicates that the sum of the numbers in squares 0,0 and 1,0 must be exactly 9.

Input

The input is given as a list with the following elements

- Tuple of the length and width of the rectangles (K, L)
- List of square assignments, given as tuples of (row, column, assignment)
- List of pairs that require some sum, given as tuples of (row1, column1, row2, column2, sum)

We will always make sure K*L = N, and all squares given in the lists will be within the board. **The input might not be solvable**.



Example of a board with 2x4 rectangles, and some pre-assigned numbers and pairs of squares with required sum. The corresponding input is [(2,4), [(1,1,3), (4,3,5), (6,5,8)], [(1,0,1,1,7), (4,3,4,4,9)]].

Task

You are expected to write four functions:

1. to_CNF(input) – Takes the input and translates it into a SAT formula in CNF form such that the formula is satisfiable if and only if the input is solvable. Also creates a list of vairables for this SAT problem. Returns (variables, CNF_formula).
2. solve_SAT(variables, CNF_formula, assignment) – The implementation of a SAT solver. Gets a list of variables and a formula, as well as a (possibly partial) assignment. The function should check if the formula is satisfiable, and if so, return a valid full assignment for the variables. If it is not satisfiable, return None for the assignment. Returns (is_satisfiable, assignment).
3. clause_status(clause, assignment) – Takes a clause and a (possibly partial) assignment as input. A clause can have one of four possible statuses: “satisfied”, “unit”, “conflict”,

"unresolved" (as explained in clarifications below). The function should check the status of the clause given the assignment, and if it is a unit clause, return the needed assignment for it. Otherwise return None for the assignment. Returns (status, assignment).

4. numbers_assignment(variables, assignment, input) – Takes the variables, the assignment of the SAT solver, and the initial input. The function should create a board corresponding to the assignment. Returns a matrix (list of lists) of numbers, in the same shape as the board, such that in index [i][j] of the matrix is the number of the corresponding solution.

We will give an input to the first function, then its output to the second function (with an empty dictionary for the assignment), and if the problem is solvable, we will give the assignment of the second function to the fourth (along with the variables from part 1 and the input). In total, for every input, the entire code should run in under 60 seconds.

In addition to checking your code for solving entire inputs, we will check every part of the code separately. One exception is the last function, which only must work with your implementation of the translation to logic and SAT solver, and will only be checked in this environment.

Clarifications

- The decision variables could be of any structure you want, but all variables must be distinct under the == method (i.e. for every two variables A,B if must hold that A==B is false).
- The structure of the CNF should be a list of clauses, every clause a list of tuples, every tuple a pair of (variable, Boolean) where the Boolean value of False represents a negated literal and Boolean value of true represents the literal is not negated.
- The function solve_SAT should work on any list of variables and formula, not just your encodings.
- The four status values for a clause are regarding a (possibly partial) assignment:
    - "satisfied" – at least one literal in the clause has its corresponding assignment.
    - "unit" – all but one literal in the clause have wrong assignments, the last literal is not assigned yet.
    - "conflict" – all the literals have wrong assignments.
    - "unresolved" – otherwise.
- The function clause_status should work on any clause and assignment, including on variables you have not defined and clauses that are not a result of running the to_CNF function you wrote.
- You don't have to use the clause_status function to solve problems, but you might find it useful.

- In the numbers_assignmet function you can assume that the variables are the ones you defined for the given input.
- The numbers_assignmet function must return a full solution to the board, including the parts that have been preassigned in the input. A solution that does not return the correct preassigned values will be considered incorrect.

Code Handout

You will receive 4 files for this homework:

1. ex3.py – this is the only file you should modify for this assignment.
2. check.py – this will run your code and makes sure it runs in the time constraints. It will also check the validity of your solution for the entire input (but not each part separately).
3. inputs.py – the inputs for the none-competitive part of the assignment.
4. utils.py – the file that contains some utility functions. You may use the contents of this file as you see fit.

Note: You can import any package that appears in the python standard library. Any other packages are not allowed.

Submission and Grading

You are to submit only the file named ex3.py as a python file (no zip, rar, etc.). We will run check.py with your ex3.py, and we will also check the outputs of each function separately with our own code. The check is fully automated, so it is important to be careful with the names of the functions. The grades will be assigned as follows:

- 30 points – creating a sound CNF representation for every problem given in the inputs (that is logically equivalent to the problem's satisfiability).
- 10 points – calculating the statuses of clauses correctly.
- 15 points – SAT solver is implemented correctly.
- 30 points – solving all given inputs in under 60 seconds (60 seconds per input).
- 25 points – competitive part. We will give your code more inputs, which they should also solve in under 60 seconds. The extra inputs can be bigger and more difficult than the inputs in the non-competitive part.
- Notice, you can get more than 100 points in this HW. Scores above 100 will not be rounded down to 100.
- The submission is due on 28.1.
- Submission in pairs only or alone. Submission in larger groups is not approved.

- You should submit in different pairs for every homework in this course (excluding homework 0).
- Write your ID numbers in the appropriate field (‘ids’ in ex3.py) as strings. If you submit alone, leave only one string.
- When grading, we check the validity of your solutions.