Report for the sudoku project

Alexandre Bourguignon

November 22th 2023

1 Constraints generation

1.1 *sudoku_constraints_number*()

In this function I calculated the number of constraint for each constraints.

- Constraints 1 and 2 : We have to generate $N * N$ constraints because there is one constraint per number per cell .
- Constraints 3 and 4 : We have to generate on each row (resp. columns) $N^2 * (N - 1)//2$ constraints and since we have N rows (resp. columns) we need to multiply the result by N so the result is $N * N^2 * (N - 1)//2$.
- Constraint 5 : In each square we have to generate $N * N * (N - 1)//2$ constraints and since we have N squares we need to multiply the result by N. The result is : $(N * N * (N - 1)//2) * N$.

1.2 *sudoku_generic_constraints*

1.2.1 *new_lit*

To be able to generate constraints for 16x16 and 25x25 sudoku I had to think of another way of encoding the propositions into SAT. The idea is to transform every proposition into 6 numbers to be able to go from 1 to 16 (resp 1 to 25). That's why i used the $zfill$ function to add an 0 if needed to form a correct proposition such as 081409.

1.2.2 Constraints

- Constraint 1: Ensures at least one number per cell. We iterate through each cell and list every possibility.
- Constraint 2: Generates a clause for every pair of different numbers, specifying that those two numbers cannot occupy the same cell. This is done for every cell.
- Constraints 3 and 4: Analyzes each row (respectively columns) to ensure that every pair of cells in that row (respectively columns) cannot have the same number.
- Constraint 5: In each square, compares every pair of cells to prevent two identical numbers in a square. Additionally, for two cells A = (x,y) and B = (a,b), ensures A != B and that A comes before B in the logical order of cells. This guarantees the avoidance of duplicate clauses.

2 *sudoku_other_solution_constraint*

2.1 Constraint generation

One way to find if there is another solution is to take the negation of every cell in the solution and restart the sudoku solver. If the solver tells that there's no solution, that means that the previous solution was in fact the only one.

3 *sudoku_solve*

3.1 Formula for $N = 16$ and $N = 25$

Since we have 6 numbers in a proposition, we have to change the formula. Suppose we have the proposition 151309. Our goal is to obtain $sudoku[14][12] = 09$ because the index starts at 0. In the first brackets we have $151309//10000 - 1 = 15 - 1 = 14$ and in the second we have $(151309//100)\%10 - 1 = 1513\%10 - 1 = 13 - 1 = 12$. Finally we put in $sudoku[14][12]$ the number $151309\%10 = 09$

4 Sudoku Generation

For this section I added a few functions such as *sudoku_generate_complete*, *update_cnf_file* and *random_move*. Lets review those :

- *random_move* : this function simply adds a random value in a random cell.
- *update_cnf_file* : as it says, this function simply updates the CNF file to be able to solve another sudoku
- *sudoku_generate_complete* : this function allows to create a full random sudoku. First it adds a random number in a random cell. This makes sure we always create a random sudoku. If we use the "-cm" option, it simply goes through every cell and remove the number $< size >$ from the sudoku.

In the *sudoku_generate* function, I start by creating a completed random sudoku. Then I'm entering a loop where I keep removing numbers and just after I check if the solution is unique. If not I'm putting the number back in his cell and if this happend more than twice, the generation stops there and I obtain a unique sudoku.

5 Collaboration

For this project I didn't exchange any piece of code with anyone except the part where I added a timer to be able to see in how much time the code ran (I shared this with Julien Peffer and another student, I don't remember his name). However, with Julien Peffer, we tried to understand together how the 16x16 ad 25x25 sudoku could have been resolveded and at the very beginning we also discovered how the CNF file works with the meaning of propositions etc.