ToC Spring 2022

HOMEWORK 6 REPORT

Bettelini Carlo, Likollari Kelvin, Milanesi Claudio,
Alexandra Willi

Introduction

As the name suggests, SAT solvers can be used to find a solution to a problem. Since the solver has its own language, the problem and the set of constraints needs to be encoded into the solver's language before using the solver. The idea is to encode each constraint as a set of clauses (consisting of ORs of variables) that will then be concatenated, creating a single CNF for which the solver will try to find a solution. Upon feeding the CNF to the solver, if the CNF is satisfiable, the solver will return the values of the variables for which the CNF evaluates to true. For the Sudoku satisfiability problem, the following conditions must hold:

1. Each cell must have a number between 1 and 9
2. Each number must appear only once in each row
3. Each number must appear only once in each column
4. Each number must appear only once in each 3 by 3 box
5. The initial input numbers must be present

Implementation

To encode the problem, we chose to denote each variable as $inCell(i, j, n)$, where i denotes the row, j denotes the column and n denotes the number in the cell. Since each cell can have a value from 1 to 9, and all cells need to be filled, we have up to $9 \cdot 9 \cdot 9 = 729$ different variables. Furthermore, since the SAT solver we chose, Z3, works with integers as variables in the clauses, we created a map from each variable to its unique ID.
Upon creating the mapping, we encoded the constraints into clauses as follows: the first condition can be encoded straight forward. What this means is that for each cell (i, j), a number between 1 and 9 must be present. We can write the clause as follows:

$$(inCell(1, 1, 1) \vee inCell(1, 1, 2) \vee inCell(1, 1, 3) \vee ...)$$

For the second condition, saying that each number must appear only once in each row is equivalent to saying that a same number cannot stay in the same row. This means that for each pair of cells of the row, the n-th number cannot be in both cells. Given the number 1 and the first two cells of the first row, this clause can be written as:

$$\neg(inCell(1,1,1) \wedge inCell(1,2,1))$$

which can be written as:

$$(\neg inCell(1,1,1) \vee \neg inCell(1,2,1))$$

If we consider more cells as well, we can write:

$$(\neg inCell(1,1,1) \vee \neg inCell(1,2,1)), (\neg inCell(1,1,1) \vee \neg inCell(1,3,1))$$
$$(\neg inCell(1,1,1) \vee \neg inCell(1,4,1)), \ \ldots \ , (\neg inCell(1,1,1) \vee \neg inCell(1,9,1))$$

We then repeat the process for each row.

This reasoning can be used for the third condition as well, but instead of iterating on pairs of cells of the rows, we iterate on pair of cells in the column. For part of the first column and the number 1 we can thus write as follows:

$$(\neg inCell(1,1,1) \vee \neg inCell(2,1,1)), (\neg inCell(1,1,1) \vee \neg inCell(3,1,1))...$$

Like for the rows, we repeat this process for all columns.

For the fourth constraint, we need to check that each number in each box is unique. Similarly as above, this is like saying that each number cannot be in the same box more than once. This means we have to check each pair in the box for each number. For part of the first box and the number one for example, we can write the clause as follows:

$$(\neg inCell(1,1,1) \vee \neg inCell(1,2,1)), (\neg inCell(1,1,1) \vee \neg inCell(1,3,1))$$
$$(\neg inCell(1,1,1) \vee \neg inCell(2,1,1)), (\neg inCell(1,1,1) \vee \neg inCell(2,2,1))$$
$$(\neg inCell(1,1,1) \vee \neg inCell(2,3,1)), (\neg inCell(1,1,1) \vee \neg inCell(3,1,1))$$
$$(\neg inCell(1,1,1) \vee \neg inCell(3,2,1)), (\neg inCell(1,1,1) \vee \neg inCell(3,3,1))$$
$$(\neg inCell(1,2,1) \vee \neg inCell(1,3,1)), (\neg inCell(1,2,1) \vee \neg inCell(2,1,1))$$
$$(\neg inCell(1,2,1) \vee \neg inCell(2,2,1)), (\neg inCell(1,2,1) \vee \neg inCell(2,3,1))$$

Once again we repeat this process for each number and remaining box.

For the last condition, we just need to write one clause for each variable we have in the input. For example, given number 3 in cell $(1,2)$ and number 6 in cell $(5,5)$, we write:

$$inCell(1,2,3), inCell(5,5,6)$$

where each variable denotes a clause: they must evaluate to True.

Upon generating all clauses, we feed them to the Z3 solver which will AND them and return a solution if the obtained CNF is solvable. A thing to note is that since the solver works with indexes and not with strings, instead of using the string representation of a variable in the clause, we use it's associated ID using the mapping we implemented previously. Also, for it to work we need to pass the clauses in a specific manner, like shown in the slides seen in class.