Master M2 ORCO – 2019/2020

SAT/SMT solver

Report of Sudoku problem

Yuzhen Wang

Content table

1. Introduction

The sudoku problem is a well-known problem. Our goal is to fill in numbers 1, 2, …, 9 to a 9*9 matrix such that each line, each row and each 3*3 little square, each number just appears once.

In the second part of this report, we introduced how to solve the sudoku problem by a z3 solver. And then, we propose our own solver by the back-tracking algorithm. At last, we give our results in the experiments and a conclusion about what we discovered and what we could do to go further.

2. Solver by z3

The package z3 is an efficient package to solve the SAT problem. In the first part of this section, we give an idea to convert the Sudoku problem to the SAT problem. While in the second part, we use the integer variable to solve Sudoku problem by using z3.

2.1. Boolean variables

The first idea to convert a Sudoku problem to a SAT problem, for each cell $a_{i,j}$, we define a Boolean variable $b_{i,j,k}$ which mean for the cell $a_{i,j}$, if the number k is chosen. Easily to see, we have $i, j, k \epsilon \{1, \dots, 9\}$. Then, we could give the following constraints.

To ensure that each cell just take one number:

$$b_{i,j,k} \Rightarrow \bigwedge_{t=1,t\neq k}^{9} \neg b_{i,j,t} \; \forall i, j, k \in \{1, \dots, 9\} \tag{1}$$

This could be done by the "Implies" in z3 solver.

To ensure that each number appears once in each line:

$$\bigvee_{j=1}^{9} b_{i,j,k} = 1 \forall i, k \in \{1, \dots, 9\} \tag{2}$$

Since the equation (1) makes sure that each cell picks one number, then, if all the numbers from 1 to 9 appear in one line, that means all the cell in this line pick the different numbers.

Similar, to ensure that each row have the number from 1 to 9:

$$\bigvee_{i=1}^{9} b_{i,j,k} = 1 \forall j, k \in \{1, \dots, 9\} \tag{3}$$

Similarly, we could also do this for all the 3*3 small squares:

$$b_{1,1,k} \vee b_{1,2,k} \vee b_{1,3,k} \vee b_{2,1,k} \vee b_{2,2,k} \vee b_{2,3,k} \vee b_{3,1,k} \vee b_{3,2,k} \vee b_{3,3,k} = 1 \forall k \in \{1, \dots, 9\} \tag{4}$$

……

If we have already known some cell, for example, we known that $a_{i,j} = k$, we give the constraint:

$$b_{i,j,k} = 1 \tag{5}$$

So far by now, we have converted the Sudoku problem to an SAT problem. By calling the z3 package, we could solve a Sudoku problem.

2.2. Integer variables

In z3 package, we could also define integer variables. That means we could also solve the Sudoku problem by using integer variables. And we could introduce the Distinct function to make sure that all the numbers are different for each line, each row and each little square.

3. Solver by back-tracking

In this section, we introduce the back-tracking algorithm to solve the Sudoku problem. At first, we present the original brutal algorithm. And in the second part, we make 2 possible small and simple optimizations.

3.1. Original algorithm

The back-tracking algorithm uses the similar idea of depth first search algorithm. Here, we propose the algorithm as follow.

1) Check if the matrix is filling
2) Find the first unknown cell
3) Test the possible filling number
4) Update the matrix and jump to step 2)
5) Reset the matrix

In this algorithm, to find an unknown cell, we fill the matrix by an order. Each time we when find the filling number is not feasible, we go back and retry other number. It uses the similar idea as DFS.

3.2. A small optimization

3.2.1. The first optimization

In the previous algorithm, to find an unknown cell, we search from the first cell of the matrix until the last one. This sometimes gives the redundant search. At each time, we note the position line i and row j, then, to search the unknown cell, we start from the position i,j.

3.2.2. The second optimization

In this part, we give another idea of optimization. At first, we calculate the possible number for each blank cell. And we start the search from the cell who has the least possibilities.

4. Implementation and conclusion

For the solvers using z3, the functions are implemented in the file “sudoku_z3”. The function “z3_sat” convert the sudoku problem to an SAT problem and the function “z3_int” uses the integer variables.

The functions in “solver.py” are the solvers by back-tracking algorithm. Here, we could call this function like “solver_back_tracking(mat, strategy)”, where “mat” is the matrix to be solved and “strategy” means the strategy users want to use for the back-tracking. The value of strategy defined as below.

0 the brutal algorithm
1 the first optimization by marking the position
2 the second optimization by search from the least possibilities

Here, we give the time in the following table. For the easy matrix, we used the instance of the teacher. For the hard matrix, we copy one hard Sudoku problem from the Internet like in Figure.

Table 1 Table of results

Functions	Hard matrix	Easy matrix
Z3_sat	7s	200ms
Z3_int	4.5s	100ms
Back-tracking (strategy 0)	1.5s	150ms
Back-tracking (strategy 1)	1.5s	1.5s
Back-tracking (strategy 2)	1.5s	1.5s

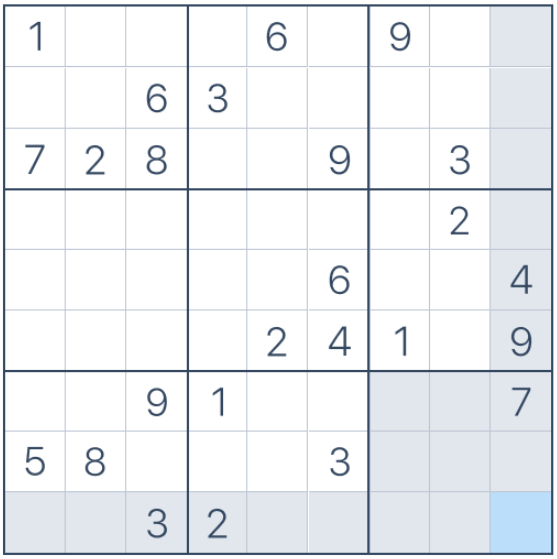

Figure 1 A hard Sudoku problem

We could find that the z3 solver is more stable. In further study, we could implement more algorithm for example with machine learning.