Instructor: Edmund M. Clarke Teaching Assistant: Michael Carl Tschantz

Assignment 5

1 Solving Sudoku using SAT

In Sudoku puzzles, one fills numbers 1 to 9 in the empty squares of a 9×9 grid such that every row and every column contains the digits 1 through 9. Furthermore, the 9×9 grid is divided into 9 smaller 3×3 grids. Each of these smaller grids must also contain the digits 1 through 9. Figure 1 shows a Sudoku puzzle and its solution.

	6		1		4		5	
		8	3		5	6		
2								1
8			4		7			6
		6				3		
7			9		1			4
5								2
		7	2		6	9		
	4		5		8		7	

Input Sudoku puzzle

9	6	3	1	7	4	2	5	8
1	7	8	3	2	5	6	4	9
2	5	4	6	8	9	7	3	1
8	2	1	4	3	7	5	9	6
4	9	6	8	5	2	3	1	7
7	3	5	9	6	1	8	2	4
5	8	9	7	1	3	4	6	2
3	1	7	2	4	6	9	8	5
6	4	2	5	9	8	1	7	3

Solution

Figure 1: Sudoku puzzle and its solution

In this problem you will write a program which does the following:

1. read a Sudoku puzzle from an input file,
2. encode the Sudoku puzzle as a CNF formula in DIMACS format (see HW4),
3. use the `MiniSat` SAT-solver to find a satisfying assignment to the CNF formula, and
4. read the satisfying assignment produced by `MiniSat` to generate the solution to the input Sudoku problem.

Suppose your program is called `solver.x` which produces an executable say `solver` after compilation. We should be able to run `solver` as follows:

```
./solver input.sudoku output.sudoku
```

where `input.sudoku` is the name of the file which contains the input Sudoku puzzle and `output.sudoku` is the name of the file which should contain the solution to `input.sudoku` after `solver` finishes.

The format of `input.sudoku` will be as follows (0 indicates that a square is empty):

```
0 6 0 1 0 4 0 5 0
0 0 8 3 0 5 6 0 0
2 0 0 0 0 0 0 0 1
8 0 0 4 0 7 0 0 6
0 0 6 0 0 0 3 0 0
7 0 0 9 0 1 0 0 4
5 0 0 0 0 0 0 0 2
0 0 7 2 0 6 9 0 0
0 4 0 5 0 8 0 7 0
```

Your solver should write a valid solution in `output.sudoku`. For the above example `output.sudoku` should contain the following:

```
9 6 3 1 7 4 2 5 8
1 7 8 3 2 5 6 4 9
2 5 4 6 8 9 7 3 1
8 2 1 4 3 7 5 9 6
4 9 6 8 5 2 3 1 7
7 3 5 9 6 1 8 2 4
5 8 9 7 1 3 4 6 2
3 1 7 2 4 6 9 8 5
6 4 2 5 9 8 1 7 3
```

This problem can be done in groups of two. One of the group members should email the code to the TA. Include in that email the names of both group members.

Please make sure that all your code for solving Sudoku problem is contained in a single file say `solver.x`. You should tell us how to compile your code on Andrew machine. We should not have to install third-party software to get your program to compile or run. The executable produced from your code should accept two command line inputs. The first input is a name of the file which contains a description of a Sudoku puzzle. The second input is a name of the file where you will write the solution to the Sudoku puzzle.

We will make several test cases available for testing and a possible template for organizing your code. You do not need to use this template and are free to use any programming language you wish provided that your program may be used as described above.

Extra credit will be rewarded for solvers that can handle boards larger than 9×9. The amount of extra credit will depend on how large and difficult of boards the solver can handle.