Practical Decision Procedures — Spring 2007

Optional exercise

1 Solving Sudoku using SAT

In Sudoku puzzle one fills numbers 1 to 9 in the empty squares of a 9×9 grid such that every row, every column, and every 3x3 box contains the digits 1 through 9. Figure 1 shows a 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, 3) Use `MiniSat` SAT-solver to find a satisfying assignment to the CNF formula, 4) read the satisfying assignment produced by `MiniSat` to generate the solution to the input Sukodu 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
```

DIMACS format for describing a CNF formula

Current SAT solvers require the input CNF formula in a particular format called the `DIMACS` format. Consider the following file `sample.cnf` in `DIMACS` format.

	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

```
p cnf 4 5
1 0
2 -3 0
-4 -1 0
-1 -2 3 4 0
-2 4 0
```

The first line (p cnf x y) says that the input is a CNF formula containing `x` variables and `y` clauses. Our example has 4 variables (1, 2, 3, 4) and five clauses. The negation of a variable is denoted by putting a minus sign in front of the variable number. Each clause is described in a line terminated by a zero. Note that 0 cannot be used as a variable number. So `sample.cnf` denotes the following CNF formula: $1 \wedge (2 \vee \neg 3) \wedge (\neg 4 \vee \neg 1) \wedge (\neg 1 \vee \neg 2 \vee 3 \vee 4) \wedge (\neg 2 \vee 4)$.

Running a SAT solver

You can run `MiniSat` SAT solver simply by the following command:

```
MiniSat_v1.14_linux sample.cnf sample.result
```

The file `sample.cnf` is a description of a CNF formula in DIMACS format. `MiniSat` reports whether the given formula is (un)satisfiable in the file `sample.result`. If the formula is satisfiable, then a satisfying assignment is also written to `sample.result`.

Several test cases available for testing and a possible template for organizing your code is available from `http://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15414-f06/www/index.html`. You can download the latest version of MiniSat SAT solver from `http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/`.