15-414 — Bug Catching — Fall 2006

Instructor: Edmund Clarke
Teaching assistant: Himanshu Jain

Assignment 2

Due date: Tuesday, October 10, 2006
Problem 2 can be done in groups of two.

1 Implication graphs

We will follow the notation from the GRASP paper in the following problem. For each of the parts you need to build an implication graph (Section 2 of the GRASP paper). Report if the implication graph contains a *conflict* node. In case of a conflict report the conflict induced clause.

(a) Clauses: $\omega_1 := (\neg a + b), \omega_2 := (\neg a + c), \omega_3 := (\neg b + \neg c + e), \omega_4 := (\neg e + d), \omega_5 := (\neg d + f)$
Current truth assignment: $\{\}$
Current decision assignment: $\{a = 1@1\}$

(b) Clauses: $\omega_1 := (\neg a + b), \omega_2 := (\neg c + \neg b + e), \omega_3 := (\neg b + \neg d + f), \omega_4 := (\neg e + \neg f), \omega_5 := (\neg g + \neg h + i), \omega_6 := (g + i + l)$
Current truth assignment: $\{c = 1@1, d = 1@2, g = 0@3, i = 1@4\}$
Current decision assignment: $\{a = 1@5\}$

2 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 (see HW1), 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.

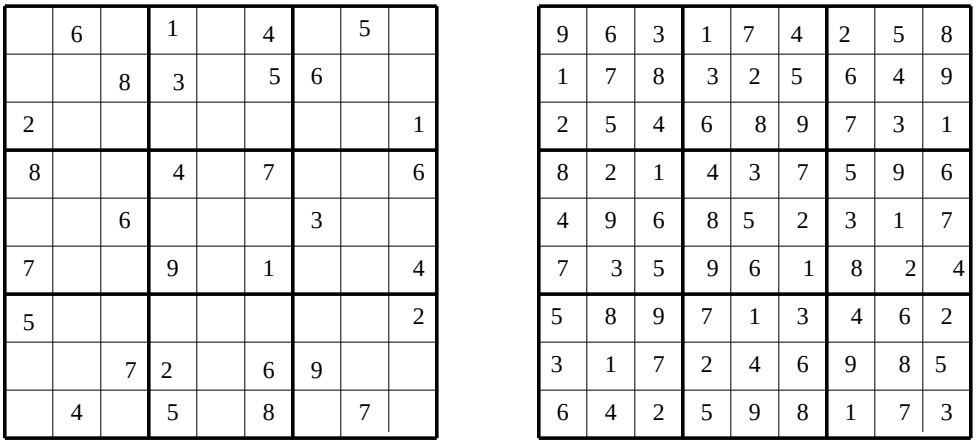
Suppose your program is called `solver.x` which produces an executable say `solver` after compilation. We should be able to run `solver` as follows:

```
./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
```



	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

```
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 TA. 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. 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.