CSE 235H SAT Homework 05 Spring 2017

Due: Monday, March 27, 2017 Written by Daniel Geschwender

Programmatically Generating the PL sentence of a Sudoku in CNF (Part 2 of 3):
The goal of this homework is to write a simple program to generate the text file, in the DIMACS CNF 'format,' of a Sudoku instance. The homework is broken into three parts as follows:

- *Part 1*: Manually write a simple CNF file in the DIMACS format; solve with MiniSAT; and write code to generate the first set of clauses that model a Sudoku puzzle.
- *Part 2*: Write code to generate a CNF file modeling all the rules of an empty Sudoku board and solve with MiniSAT.
- *Part 3*: Write code to parse a string representing a partially filled instance of Sudoku. Add the corresponding clauses to the CNF file and solve with MiniSAT.

In this homework, you have to do only Part 2.

Grading Rubric for Part 2:

To Do	Points
Problem A: Code and README are clear and commented	3
Problem A: Program generates the correct clauses	6
Problem A: Output is properly formatted DIMACS CNF file	2
Problem B: MiniSAT finds a solution	2
Problem B: Solution is verified with `sudokuVerifier.pl`	2
Total:	15

General Instructions:

- The program may be written in any language that can be compiled and run on `cse.unl.edu`.
- You are free to use file input/output or standard input (stdin) and standard output (stdout). If using stdout (printing to the console) you can write to a file by redirecting the output. (`./yourProgram > outputFilename.cnf`)
- A README file must be included. It should clearly describe all necessary steps to compile and run the program.
- The model should follow the Sudoku CNF formulation from the textbook (see page 33) and reproduced below.
- The generated output should conform to the DIMACS CNF file specifications described below.
- Submit your code, the README file, and all accompanying files via handin. No hard copy is required. All submitted files should include your last name using the following filename format: `<lastname>-<filename>.<extension>`
- This homework must be completed individually. LaTeX bonus and partner policy do *not* apply.

The Sudoku CNF Formulation:
We will adopt the following formulation of the Sudoku problem to generate the CNF file:

- The proposition $p(i, j, n)$ indicates that the cell in row i and column j is given value n. In the CNF file, represent $p(i, j, n)$ by ijn. (e.g., $\neg p(3, 8, 7)$ corresponds to -387 in the CNF file)
- Every row contains every number:

$$\bigwedge_{i=1}^{9} \bigwedge_{n=1}^{9} \bigvee_{j=1}^{9} p(i, j, n) \tag{1}$$

- Every column contains every number:

$$\bigwedge_{j=1}^{9} \bigwedge_{n=1}^{9} \bigvee_{i=1}^{9} p(i, j, n) \tag{2}$$

- Every 3x3 block contains every number:

$$\bigwedge_{r=0}^{2} \bigwedge_{s=0}^{2} \bigwedge_{n=1}^{9} \bigvee_{i=1}^{3} \bigvee_{j=1}^{3} p(3r + i, 3s + j, n) \tag{3}$$

- No cell contains more than one number:

$$\bigwedge_{i=1}^{9} \bigwedge_{j=1}^{9} \bigwedge_{n=1}^{8} \bigwedge_{m=n+1}^{9} (\neg p(i, j, n) \vee \neg p(i, j, m)) \tag{4}$$

DIMACS CNF Format Specification:

- Comment lines begin with the character 'c'.
- A problem line must be included before any clauses. The problem line uses the following format: `p cnf <# variables> <# clauses>`.
- Each clause is given by a line of non-null numbers, separated by spaces, and ending with a '0'. The numbers correspond to variables. A negative number represents a negated variable in the clause.

Note on MiniSAT Variables:
Because of the way the variables are specified in our Sudoku model, there are gaps in the numbering of the variables. MiniSAT will see that the highest variable is 999 and will assume that there are 999 variables. This is a 'feature' of MiniSAT. The solution generated by MiniSAT will include all the variables in 1...999. The additional variables (1...110, 120, 130, etc.) should simply be ignored.

Tasks for Part 2

Problem A:
Modify your program from Part I to generate a complete CNF file to model an empty Sudoku grid. You will need to write loops to generate the remaining sets of Sudoku clauses (i.e., Expressions (2), (3), and (4)). Use loops similar to those used to generate the first set of clauses (i.e., Expression (1)). Pay attention to differences in the indices, negations, and clause termination location. The code should produce output resembling Figure 1, Figure 2, and Figure 3.

Your code should produce output that is properly formatted in the DIMACS CNF format. It should include a comment header briefly describing the problem, a problem line, and comment lines indicating each of the four sets of clauses. Figure 4 shows required structure for the CNF file.

```
111 211 311 411 511 611 711 811 911 0
112 212 312 412 512 612 712 812 912 0
113 213 313 413 513 613 713 813 913 0
114 214 314 414 514 614 714 814 914 0
115 215 315 415 515 615 715 815 915 0
116 216 316 416 516 616 716 816 916 0
117 217 317 417 517 617 717 817 917 0
118 218 318 418 518 618 718 818 918 0
119 219 319 419 519 619 719 819 919 0
121 221 321 421 521 621 721 821 921 0
122 222 322 422 522 622 722 822 922 0
123 223 323 423 523 623 723 823 923 0
                ⋮
```

Figure 1: Expected output for Expression (2).

```
111 121 131 211 221 231 311 321 331 0
112 122 132 212 222 232 312 322 332 0
113 123 133 213 223 233 313 323 333 0
114 124 134 214 224 234 314 324 334 0
115 125 135 215 225 235 315 325 335 0
116 126 136 216 226 236 316 326 336 0
117 127 137 217 227 237 317 327 337 0
118 128 138 218 228 238 318 328 338 0
119 129 139 219 229 239 319 329 339 0
141 151 161 241 251 261 341 351 361 0
142 152 162 242 252 262 342 352 362 0
143 153 163 243 253 263 343 353 363 0
                ⋮
```

Figure 2: Expected output for Expression (3).

```
-111 -112 0
-111 -113 0
-111 -114 0
-111 -115 0
-111 -116 0
-111 -117 0
-111 -118 0
-111 -119 0
-112 -113 0
-112 -114 0
-112 -115 0
-112 -116 0
      ⋮
```

Figure 3: Expected output for Expression (4).

```
c <Description of problem>
p cnf <#variables> <#clauses>
c <Description of clauses>
<Clauses from Expression (1)>
c <Description of clauses>
<Clauses from Expression (2)>
c <Description of clauses>
<Clauses from Expression (3)>
c <Description of clauses>
<Clauses from Expression (4)>
```

Figure 4: Structure of Sudoku CNF file.

Problem B:
Execute the following steps:

1. Take your generated Sudoku CNF file and solve it using MiniSAT. (An 'empty' Sudoku board can be solved by MiniSAT.)
2. Obtain the results file from MiniSAT.
3. Use the provided Sukodu verifier perl script to print your solution in an easily readable grid layout and verify its correctness:

   `cse.unl.edu/~cse235h/files/verifySudoku.pl`

   The script reads from standard input. So, you need to use input redirection to read in the results file and output redirection to write the formatted results to a new file:

   `perl verifySudoku.pl < minisatResults.txt > formattedResults.txt`

Files to Submit to Handin:
Use the following filename format: `<lastname>-<filename>.<extension>`

- Your code
- A README file
- Your generated Sudoku CNF file
- The MiniSAT results file
- The formatted solution output generated by the Sudoku verifier script