CSE 235H SAT Homework 04 Spring 2018

Due: Monday, February 19, 2018 Written by Daniel Geschwender

Programmatically Generating the PL sentence of a Sudoku in CNF (Part 1 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 expressing 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 1.

Grading Rubric for Part 1:

Dessert problem CNF file is properly formatted	3
MiniSAT gives correct solution when run on the dessert CNF file	3
Code is clear and commented	3
Program generates the correct output	6
Total:	15

General Instructions:

- The program must be written in Java and compile and run on Webgrader (`cse.unl.edu/~cse235h/grade/`).
- Your program must use standard input (stdin) and standard output (stdout).
- 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 and all accompanying files via handin. No hard copy is required. All submitted files must match the filenames specified in the assignment. Webgrader will use the files submitted through handin and requires exact filenames.
- 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 I

Problem A:
Consider the following dessert problem:

> There are four choices of desserts: ice cream, fruit bowl, cake, pie. Exactly one dessert must be selected (i.e., one and only one).

$$
\begin{aligned}
& (iceCream \vee fruitBowl \vee cake \vee pie) \\
\wedge\ & (\neg iceCream \vee \neg fruitBowl) \\
\wedge\ & (\neg iceCream \vee \neg cake) \\
\wedge\ & (\neg iceCream \vee \neg pie) \\
\wedge\ & (\neg fruitBowl \vee \neg cake) \\
\wedge\ & (\neg fruitBowl \vee \neg pie) \\
\wedge\ & (\neg cake \vee \neg pie)
\end{aligned}
$$

Manually write a DIMACS file (**dessert.cnf**) of the CNF sentence that model the desserts problem described above.

- Add a comment line to the file to describe your variables.
- Make sure you include the problem line.
- Run MiniSAT on the file that you wrote, store the MiniSAT results with the solution in an output file (**results.txt**).

Problem B:
Write a program (**GenerateSudoku.java**) to generate the first set of clauses describing the Sudoku rules(Expression (1)). Use the pseudocode shown in Algorithm 1 to help structure your program. Notice the correspondence between the loops and the iterated conjunction/disjunction operators. The code should produce output resembling Figure 1. Comment/problem lines are *not* required at this point. The output does *not* yet need to be run on MiniSAT.

```
1 foreach i ∈ [1...9] do
2     foreach n ∈ [1...9] do
3         foreach j ∈ [1...9] do
4             print " "+i+j+n
5         print " 0\n"
```

Algorithm 1: Loop structure for printing clauses in DIMACS CNF format. Note: the '+' symbol represents a string concatenation operator.

```
111 121 131 141 151 161 171 181 191 0
112 122 132 142 152 162 172 182 192 0
113 123 133 143 153 163 173 183 193 0
114 124 134 144 154 164 174 184 194 0
115 125 135 145 155 165 175 185 195 0
116 126 136 146 156 166 176 186 196 0
117 127 137 147 157 167 177 187 197 0
118 128 138 148 158 168 178 188 198 0
119 129 139 149 159 169 179 189 199 0
211 221 231 241 251 261 271 281 291 0
212 222 232 242 252 262 272 282 292 0
213 223 233 243 253 263 273 283 293 0

⋮
```

Figure 1: A sample of the expected output.

Files to Submit to Handin:

- The dessert problem CNF file (**dessert.cnf**)
- The MiniSAT results file for the dessert problem (**results.txt**)
- Your code (**GenerateSudoku.java**)

Running on Webgrader:
After submitting your files on Handin, you can run the Webgrader to verify your submission. You can access the Webgrader at `cse.unl.edu/~cse235h/grade/`. The Webgrader script will print the contents of all required files, compile your code (using 'javac -J-Xmx256m GenerateSudoku.java'), run your code (using 'java -Xmx256m GenerateSudoku'), and print the program output.