Hexadecimal Sudoku Solver

Jordan Turley

For this project, we were tasked with reducing hexadecimal Sudoku problems into SAT, and then using a SAT solver to find a solution. We were given input files with some spaces filled in and some spaces left blank, and we had to use these and a SAT solver to find a solution to these problems.

To frame this project, I had a SAT Boolean variable for each possible value of each cell on the board. $X_{1,1,0}$ would be the variable for a 0 appearing in the (1, 1) position on the board. $X_{5,6,c}$ would be the variable for a c appearing at (5,6). To reduce these to SAT, I had the variables 1 through 16 represent 0 through f for (1, 1) (the top left cell on the board), 17 through 32 represent 0 through f for (2, 1) (one to the right of the top left), and so on. At the end of the row, it just keeps going on to the next row.

To use my code, you create a SudokuBoard object, passing it the name of the input file, and the name of the file to output the SAT reduction to. Then, you call satReduce() on the object, which will reduce the problem to SAT and output it to the filename given before. Then, you can just use any SAT solver you want on the file, and it will give you a solution. The Runner class goes through the ten given Sudoku problems and finds a solution, shows it, then checks if there is another solution.

I used Java for my implementation of hexadecimal Sudoku. I did use a third party package. I used a package called SAT4J, which is a SAT solver written in Java. This made it a lot easier than using minisat, which probably could have been faster, but SAT4J was still extremely fast, and found solutions or determined they were unsatisfiable in very little time. For

my data structures, I used a 2D array of Integer objects to represent the Sudoku board when I read it in from the file. This way, instead of just using a 2D array if ints, I could store the number of a cell in the array if it was set, or leave it as null if it was not. For the SAT reduction, at first I thought that I should use some sort of 3D array, but then I realized that I would just be using the indexes of the array instead of the actual values of the array, so I just used a bunch of loops using the size of the board (which I stored in a constant SIZE in SudokuBoard). I had the general idea down, but I had to do some trial and error, and looking through the output file to see what the variables should be and what they actually were before I figured the problem out completely. I didn't receive any outside help on this project, and didn't have any unresolved bugs that I know of. I developed this in Eclipse, and that made it easier to include the SAT4J .jar file in my project, but to run it from the command line, as long as you have the .jar, which I included with my code, you can run it like this:

javac -cp ".;org.sat4j.core.jar Runner.java

java -cp ".;org.sat4j.core.jar" Runner

I included the output from my program, with solutions to all the problems, as Output.txt with my code. For every problem, the solution that I found was unique.

I learned a lot from this assignment. The main thing I learned is how powerful a SAT reduction and a SAT solver can be. From this project, we can see a real life use of a SAT solver. When we made our own SAT solvers, we just used them to solve files that had no real meaning, but after doing this, we can see what a SAT solver can really do, and how it can solve a real problem. If you can do a SAT reduction of a problem, you can send it to a SAT solver, and it will solve it in a jiffy, as opposed to implementing some sort of search algorithm on the problem, and

coming up with a specific heuristic that will only be used for that one problem, and then having to run the search algorithm for many minutes, hours, or even days if it is a very hard problem.