**Reducing Sudoku to SAT**

The typical Sudoku puzzle that one finds in newspapers or pastime books consists of a square grid of size 9, containing preset values in some of its 81 cells. The grid is subdivided into 9 3x3 boxes. The objective of the puzzle is to fill each blank cell so that each column, each row, and each box contains all the digits from 1 to 9.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

For this project you will develop a SudokuToSAT reducer that receives an input puzzle and writes out a SAT instance in the .cnf format. Your reducer will:

1. read an input Sudoku instance from a file,
2. translate the input Sudoku instance into an equivalent SAT instance

Here are the details.

**Step 1**

Your program will read from the command line the name of a file, and then read from this file the input Sudoku puzzle (instance). An input file for a Sudoku instance contains:

* Zero or more comment lines at the beginning of the file, each starting with the character ‘c’, followed by
* a line with “3 3” that indicates the dimensions of each box in the puzzle, followed by
* a sequence of 81 integers in the range from 0 to 9. These integers are the values in Sudoku grid in row-major order, with a 0 indicating a cell that does not have a preset value.

**Step 2**

Your program will then translate the input Sudoku instance into an equivalent SAT instance.

**Translating a Sudoku instance into an equivalent SAT instance**

The constraints of **a Sudoku puzzle can be modeled using a boolean formula** in conjunctive normal form (cnf). There are several ways to translate or encode a Sudoku problem into an equivalent SAT problem. The remaining of this section presents one of these encodings.

Create 729 (9x9x9) different boolean variables vi,j,k, where i, j and k take on values from 1 to 9. Intuitively, the boolean variable vi,j,k is set to true if and only if the cell at row i and column j takes the value k. Recall the constraints on a Sudoku puzzle solution:

1. Each row must have all the digits 1 through 9,

2. Each column must have all the digits 1 through 9,

3. Each 3x3 box must have all the digits 1 through 9,

4. Each cell must have exactly one value in the range 1 through 9.

Constraints 1, 2, and 3 are similar except that each refers to a different subset of cells.

We now present a SAT encoding for Constraint 1. Constraint 1, when applied to a row i, can be viewed as a conjunct (and) of the following nine constraints:

1.1. The number 1 is the value of exactly one cell in row i.

1.2. The number 2 is the value of exactly one cell in row i.

...

1.9. The number 9 is the value of exactly one cell in row i.

We will only address Constraint 1.1 since the other eight constraints can be translated in analogous manner. Constraint 1.1, in turn, can be broken down as a conjunct of two constraints:

1.1.A. At least one cell of row i has the value 1

1.1.B. At most one cell of row i has the value 1

Constraint 1.1.A, for row i, yields the propositional clause:

(vi,1,1  or vi,2,1 or vi,3,1 or vi,4,1 or vi,5,1 or vi,6,1 or vi,7,1 or vi,8,1 or vi,9,1)

Constraint 1.1.B, for row i, yields a subformula comprising 36 clauses of the form

((not vi,j,1) or (not vi,k,1))

combined together with the *and* operator, for all distinct combinations of j and k, j and k in the range 1 through 9. Can you think of a mathematical formula to find the number of such distinct combinations?

A total of 37 (i.e. 1 + 36 ) clauses from the translation of Constraint 1.1 for row i value 1.

A total of 37 \* 9 = 333 clauses from Constraint 1 for row i.

A total of 333 \* 9 = 2997 clauses from Constraint 1.

Constraints 2 and 3 can be encoded in a similar manner.

For constraints 1, 2 and 3 we get a total of 333 \* (9 + 9 + 9) = 8991.

Constraint 4, for the cell at row i and column j, is encoded as a conjunction of two constraints:

4.1. The cell at row i and column j has at least one value in the range 1 through 9

4.2. The cell at row i and column j has at most one value in the range 1 through 9

Constraint 4.1, for the cell at row i and column j, yields the propositional clause:

(vi,j,1  or vi,j,2 or vi,j,3 or vi,j,4 or vi,j,5 or vi,j,6 or vi,j,7 or vi,j,8 or vi,j,9)

Constraint 4.2, for the cell at row i and column j, yields a subformula comprising 36 clauses of the form

((not vi,j,x) or (not vi,j,y))

combined together with the *and* operator, for all distinct combinations of x and y, x and y in the range 1 through 9.

Constraint 4 results in a total of (1 + 36) \* 81 = 2997.

Any preset values that appear in a Sudoku puzzle lead to additional clauses in its translation. For example, if the cell in row 3 and column 7 has the preset value 6, we add the clause (v3,7,6) to the formula.

A Sudoku puzzle translates in this manner into a SAT instance with 729 variables and approximately 12000 clauses: i.e. 8991 + 2997 = 11988 from all 9 rows, 9 columns, 9 boxes and 81 cells; and, a few more for the preset values.

For Step 2 of this laboratory assignment, you will write a program that converts a Sudoku puzzle (input in Step 1) into an equivalent SAT instance. Notice that the great majority of the clauses in the formulas for all Sudoku puzzles will be identical, since the only differences from one puzzle to another will be in the clauses that encode their preset values.

An example of a Sudoku input file of size 9 (i.e. 9 X 9 square grid):

3 3

5 4 0 2 0 9 0 0 1

0 0 0 5 0 0 0 0 4

0 0 7 0 0 0 9 0 0

8 0 0 0 3 0 0 6 7

0 0 0 6 0 5 0 0 0

9 3 0 0 1 0 0 0 2

0 0 1 0 0 0 6 0 0

2 0 0 0 0 6 0 0 0

3 0 0 1 0 7 0 4 9

The first line in the above example indicates that each box in this puzzle is of size 3 (i.e. 3 X 3). This puzzle should translate into 729 variables and close to 12000 clauses.

Your program will translate an instance of the Sudoku puzzle such as the above and output the computed SAT instance to a file in DIMACS format[1]. A file in DIMACS format contains:

* Zero or more comment lines at the beginning of the file, each starting with the character ‘c’.
* The first data line has the format “p cnf number-of-variables number-of clauses”. Note that variables are numbered 1 through number-of-variables.
* The specification of the clauses follows, one clause per line. A clause line ends with a ‘0’. A clause is specified by writing the numbers of the literals that occur in it. Note that the negated variable number **i** is specified as **–i**; thus, the clause (not-**x12** or **x3** or **x41**) is specified in this file format as –12 3 41 0.

It is customary to give SAT input files the .cnf extension. A sample SAT input file contains:

**c This file illustrates the DIMACS input file format, and specifies the formula**

**c (x1 or x2) and (not-x2 or x3 or x4) and (not-x1 or not-x3 or not-x4)**

**p cnf 4 3**

**1 2 0**

**-2 3 4 0**

**-1 -3 -4 0**

First five lines of the output for the instance Sudoku board is:

p cnf 729 12016

1 10 19 28 37 46 55 64 73 0

-1 -73 0

-1 -64 0

-1 -55 0

**Submission instructions**

Follow the following schedule:

Assignment 2 – Part 1: due on or before - section 1: Feb 19; section 2: Feb 20

Assignment 2 – Part 2: due on or before - section 1: Feb 19; section 2: Feb 20

Assignment 2 – Part 3: due on or before - section 1: Feb 26; section 2: Feb 27

Assignment 2 – Part 4: due on or before - section 1: Mar 1; section 2: Mar 1

Mar 1 is the hard deadline. The rest are soft deadlines. The difference between the hard and soft deadlines is only in what you will submit. Other than that all deadlines must be strictly adhered to. For the soft deadlines, it is enough if you submit the hard copy of the source code of the Part that is due. Please do submit anything else such input/output files, object code, or an execution of the code. Each such submission will be treated as the final submission of the part that is due, and graded.

By the hard deadline, submit a flash drive containing the source code, input and output files plus the object (or executable) code.

Sometimes I may ask you to resubmit after making some changes/improvements to your code. In those situations, submit both the original and the improved versions of the code.

**\*\*\* Submit your assignments in my office. Please do not hand it to me in class. \*\*\*\***

See the syllabus for additional info on how to submit a project.