Implementing a logical sudoku solver

Jouke van der Maas & Koen Keune

March 27, 2013

1 Overview

During this project, a sudoku solver was implemented. Only logical solving strategies were used; there are no brute-force methods. The underlying principle to the solve method is that once possible numbers have been calculated for each square, they can only be removed. Removal of possible numbers is achieved by a strategy. By trying strategies repeatedly, one of two states can always be achieved. From these states, a new number can be entered and the process can repeat.

2 Implementation

The implementation is based around the use of strategies, implementations of the interface `Strategy`. The solver, sudoku representation and UI also support sudokus of other sizes (i.e. a 6x6 sudoku), but no support was added for loading these from a file.

2.1 Terminology

A sudoku consists of nine rows, columns and squares, called *containers*. Each of these contains nine *cells*. Cells can have a value (numbers one through nine) or a set of up to nine possibilities.

2.2 End-states

The solver uses strategies to reduce the sudoku to one of the two end-states. The first end-state from which a new number can be entered is the *Single Possibility rule*. This rule states that if a given cell has only one possible

number, that number should be entered. The second end-state is the *Only Square rule*. This rule states that if a given number can be entered in only one cell within a container (row, column or square), that number should be entered. All other strategies implemented for the project only remove possibilities from cells, leading to one of these two end-states.

2.3 Used Strategies

Following are the strategies that are used in this particular order, based on complexity.

2.3.1 Basic Strategy

The most basic solving strategy (implemented in `OneOfEachStrategy`). Simply removes values that have already been entered for each row, column and square.

2.4 Locked Strategy

Searches for 2 or 3 possibilities in a column that are the same and don't occur in another place in the square. If one is found, remove those possibilities on every other place on the same column.

2.5 Double-Locked Strategy

Searches for possibilities that occurs in one row (or column) in a square and another row (or column) in another square so that blocks possibilities in those rows (or columns) in the third square.

2.6 Naked Group Strategy

Searches for groups of cells that contain the same possibilities. For example, a group of three cells that have the same three possibilities (n cells with the same n possibilities). When these are found, it is guaranteed the same possibility cannot occur anywhere else within the container, so those can be removed.

2.7 Hidden Twin Strategy

Similar to the naked group. When two possibilities are found in only two cells, and these cells are the same, nothing else can go in those two cells. This strategy is more complex than the naked group, so only twins are considered.

2.8 Forcing Chains Strategy

Looks at the consequences of filling the possibilities of a cell with two possible values. If both possibilities lead to the same value elsewhere, than that value must be entered. This is the most complex strategy and can slow the solver down quite a bit.

3 Results

On the provided file `big5` (containing 10.000 sudokus), the solver performs with a accuracy of 99.9%:

```
Puzzles: 10000
Solved: 9989 (99.9%)
Invalid puzzles: 0
Time: 72.26s (7.23ms average per puzzle)
```

The accuracy on harder puzzles (such as the provided `top100`) is similar, but a lot slower:

```
Puzzles: 100
Solved: 99 (99.0%)
Invalid puzzles: 0
Time: 51.91s (519.10ms average per puzzle)
```

Although this seems to be caused mainly by a few puzzles, that needs a lot of the more complicated strategies to solve. Without these puzzles the performance is much better:

```
Puzzles: 98
Solved: 97 (99.0%)
Invalid puzzles: 0
Time: 16.69s (170.27ms average per puzzle)
```

Overall the solver can solve most puzzles (about 99% of the harder ones) and solves them pretty fast on average.