Project Proposal

Harry Smith
November 22, 2019

KenKen Solver

I'd like to implement a solver for the game KenKen. If you aren't familiar with the game, it's like Sudoku but with a few extra complications:

1. Every game of KenKen is played on an $N \times N$ board.
2. Each row and column must contain exactly one instance of the numbers $1 \dots N$, with no repeats within a row or column.
3. The board is partitioned into a set of contiguous regions, each of which imposes a target number and an operation.
   - A region may contain more than one of the same number, subject to constraint **2**.
   - The operation is applied to all numbers in the region to get a result. The result of this applcation must be equal to the target number specified for the region.
   - The operations may be any of $+, *, -, /$. In the case of the non-commutative operators $(-, /)$, these regions typically contain only two cells and the ordering is interpreted as `larger - smaller` or `larger / smaller`.
4. Any region which contains only one cell can be trivially filled in with its target number. That is, a region of only one cell with target value 4 can only be filled in with the number 4.

I estimate that this will be somewhat more challenging to implement than a solver for Sudoku. In particular, an abstract representation of a KenKen board requires formalizing more varieties of constraints and allowing for a good deal of variance in the number of these constraints in a given game. Further, I will implement the AC-3 constraint satisfaction solver with backtracking as the algorithm to actually deduce the solutions. I suspect that the variety of possible constraints will make this implementation more difficult than Marlow's solution for Sudoku.

The parallelization will come in the form of allowing multiple puzzles to be solved at once rather than parallelizing the process of solving a single puzzle. To date, I have accumulated a set of about 1,200 puzzles with sizes ranging from 3×3 to 9×9, and their corresponding solutions in text format. With this in mind, I anticipate that my process will be the following:

1. Formalize a KenKen game in Haskell with functions for determining whether board states are legal and when the puzzle is completed.
2. Implement a parser of boards and solutions that matches my dataset.
3. Create the AC-3 solver.
4. Design a Puzzle Pool which employs the solver to generate solutions for each puzzle and does so in parallel.

If you're curious about my corpus of puzzles, I scraped them using Python[1] from `http://www.mlsite.net/neknek/play.php` and an example 3×3 puzzle looks like this:

```
# 3
+ 7 A1 B1 B2
! 1 C1
* 3 A2 A3 B3
* 6 C2 C3
```

[1] Sorry :(