Constraint Programming: Assignment no. 1

Marco Kuhlmann, M.Sc., Dipl.-Inform. Guido Tack

http://www.ps.uni-sb.de/courses/cp-ss07/

Exercise 1.1 (Download Gecode/J)

Download and install Gecode/J from the Gecode website. You find packages for all major operating systems under

`http://www.gecode.org/gecodej/download.html`.

Get yourself acquainted with the documentation, which can be found at

`http://www.gecode.org/gecodej/documentation.html`.

Exercise 1.2 (Send More Money)

Here's the famous Send More Money problem (SMM). Consider the equation

```
SEND + MORE = MONEY.
```

Your task is to find distinct digits for the letters such that the equation holds. (Neither `S` nor `M` may be zero.)

a) Solve the problem with pencil[1] and paper. Can you solve it without guessing, i.e. just by inference? What is the solution?

b) In the Gecode/J distribution, you find a sample implementation of SMM. Try to compile and run it and familiarise yourself with Gist (the Gecode Interactive Search Tool). What are the contents of the constraint store after propagation has reached its fixpoint for the first time? What does this tell you about the power of propagation for this example?

c) Can you find a branching heuristic that produces a search tree smaller than the one produced by the first-fail strategy?

[1] Any other writing device will do well, too.

Exercise 1.3 (Send Most Money)

Modify the implementation from the previous exercise to obtain a problem solver for the *Send Most Money* example (SMM+) presented in the lecture. This problem has more than one solution. In this exercise, we will use branch-and-bound search to find the best one.

a) Introduce a new problem variable `money`. Constrain `money` to be the sum of the letters `MONEY`. How big can money be at most, i.e. which initial domain can you give for the variable `money`?

b) For branch-and-bound search, our script has to implement a method

   ```
   public void constrain(Space bestSol)
   ```

   that takes the currently best solution `bestSol` as an argument and posts the additional constraint that every solution found from now on must be better than `bestSol`. In our case, you will have to post the constraint `money > bestSol.money`.

   In addition, you have to add the line `opt.bab = true;` to your `main` method to switch on the optimization search engine.

Hints: Have a look at the `constrain` and `main` methods of the example `Photo.java` in the Gecode/J distribution.

Exercise 1.4 (Sudoku)

Write a program to solve Sudoku puzzles.

A Sudoku puzzle consists of a 9 by 9 array of squares. Each square should contain one of the digits 1, 2, . . . , 9. The rules of the puzzle are that in a solution, each row should contain all the different digits, each column should contain all the different digits, and each major 3 by 3 block should contain all the different digits. An instance of a Sudoku puzzle is a square with some of the digits pre-filled. A valid Sudoku instance will have exactly one solution.

Write a program that solves Sudoku instances using Gecode/J. A good starting point can be taking one of the examples and modifying it, calling it `Sudoku.java` instead. Proceed as follows:

- First you must decide what the variables are. A good and natural model is to have one `IntVar` for each square in the puzzle, with the domain 1...9. Represent this as a `VarArray<IntVar>` of length 81.

- Functions which help you accessing fields and extracting rows and columns are provided on the course webpage.
- The constraints follow from the rules. The `distinct` constraint can be useful here, look it up in the documentation.
- Write a custom `toString()` method for your class, so that you can see the results.
- Remember to implement the constructor `Sudoku(Boolean share, Sudoku s)`, and to call `super()` in the main constructor.

Experiment with different heuristics and propagator strengths. You can change the propagator strength of distinct, by giving one of `ICL_DEF`, `ICL_VAL`, `ICL_BND`, and `ICL_DOM` as the last argument. See if there is any difference in the sizes of the search-trees.

On the course webpage, you will find a file containing several Sudoku instances, represented as 9 by 9 arrays of integers. Each non-zero integer represents a pre-filled square. Your program should be able to solve these instances.