DM 509 Programming Languages

Fall 2012 Project (Part 1)

Department of Mathematics and Computer Science
University of Southern Denmark

September 18, 2012

Introduction

The purpose of the project for DM509 is to try in practice the use of logic and functional programming for small but non-trivial examples. The project consists of two parts. The first deals with logic programming and the second part with functional programming.

Please make sure to read this entire note before starting your work on this part of the project. Pay close attention to the sections on deadlines, deliverables, and exam rules.

Exam Rules

This first part of the project is a part of the final exam. Both parts of the project have to be passed to pass the course.

Thus, the project must be done individually, and no cooperation is allowed beyond what is explicitly stated in this document.

Deliverables

There is one deliverable for this first part of the project: A short project report in PDF format (2-5 pages without front page and appendix) has to be delivered. This report should contain the following 7 sections:

- front page
- specification
- design
- implementation
- testing
- conclusion
- appendix including all source code

The report has to be delivered as a single PDF file electronically using Blackboard's SDU Assignment functionality.

Deadline

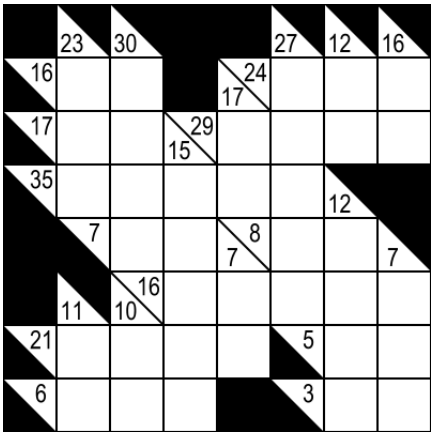
October 5, 2012, 12:00

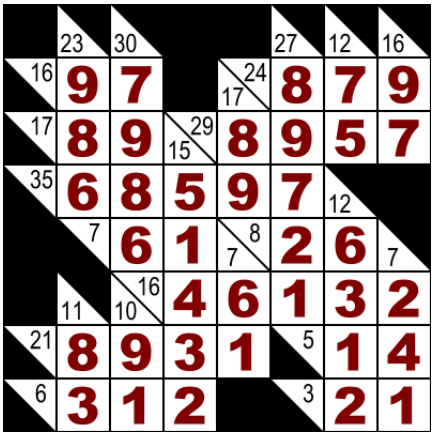
The Problem

Your task in this part of the project is to write a solver for Kakuro puzzles. Kakuro puzzles are a kind of crossword puzzles with numbers where the following two conditions have to be met:

- In each consecutive row or column of empty fields, all numbers have to be from the set $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ and they must all be different.
- For each consecutive row or column of empty fields, a number at the to the left or above of the fields specifies the sum of all these numbers.

The following two figures show a Kakuro puzzle and its solution (taken from Wikipedia, both graphics are under the GNU Free Documentation License).





The Input

For input to your program, the Kakuro puzzles are represented as Prolog terms. More specifically, they are represented as matrices (lists of rows which are lists of fields) where the fields can be one of the following four types:

- A frame block with no sum information (black in the graphics above) is represented by the term `x/x`.
- A frame block with sum information for a row (black lower left, number N in upper right) is represented by the term `x/N`, e.g., `x/16` or `x/24`.
- A frame block with sum information for a column (number N in lower left, black upper right) is represented by the term `N/x`, e.g., `23/x`.
- An empty field is represented by just the term `x`.

Thus, for our example above we obtain the following Prolog term:

```
[[x/x , 23/x, 30/x , x/x  , x/x  , 27/x, 12/x, 16/x],
 [x/16, x   , x    , x/x  , 17/24, x   , x   , x   ],
 [x/17, x   , x    , 15/29, x    , x   , x   , x   ],
 [x/35, x   , x    , x    , x    , x   , 12/x, x/x ],
 [x/x , x/7 , x    , x    , 7/8  , x   , x   , 7/x ],
 [x/x , 11/x, 10/16, x    , x    , x   , x   , x   ],
 [x/21, x   , x    , x    , x    , x/5 , x   , x   ],
 [x/6 , x   , x    , x    , x/x  , x/3 , x   , x   ]].
```

The home page of the course contains a number of possible inputs to test your program on.

The Output

The output of your solver should also be a Prolog term. The representation is similar to the one for the Input except for all `x` being replaced by the appropriate number.

Thus, for our example above we obtain the following Prolog term:

```
[[x/x , 23/x, 30/x , x/x  , x/x  , 27/x, 12/x, 16/x],
 [x/16, 9   , 7    , x/x  , 17/24, 8   , 7   , 9   ],
 [x/17, 8   , 9    , 15/29, 8    , 9   , 5   , 7   ],
 [x/35, 6   , 8    , 5    , 9    , 7   , 12/x, x/x ],
 [x/x , x/7 , 6    , 1    , 7/8  , 2   , 6   , 7/x ],
 [x/x , 11/x, 10/16, 4    , 6    , 1   , 3   , 2   ],
 [x/21, 8   , 9    , 3    , 1    , x/5 , 1   , 4   ],
 [x/6 , 3   , 1    , 2    , x/x  , x/3 , 2   , 1   ]].
```

The Task

Implement a predicate `solve/2` that takes an unsolved Kakuro puzzle as the first argument and instantiates the second argument by its solved form.

Keep in mind, that there are many different ways how to implement such a `solve/2` predicate. Explain your approach, implement it, and produce the report.

The Foundations

There is a number of built-in predicates that you might find useful when building a Kakuro solver:

- `var/1`, which is true if the argument is an (uninstantiated) variable
- `fd_var/1`, which is true if the argument is an (uninstantiated) constraint variable
- `number/1`, which is true if the argument is an integer or a floating point number
- `read/1`, `write/1`, and `nl/0` for input and output
- `fd_domain/3`, `fd_all_different/1`, `fd_labeling/1`, and `#=` for constraint solving

To make life easier for you, I have also defined some predicates for outputting Kakuro puzzles (`show/1`, works both on puzzles in input and in output form) and for converting Kakuro puzzles in a different notation to our representation (`convert/2`, works when the first argument is a term as used in the additional examples linked from the course home page).

Finally, there is a template available from the course home page for calling your `solve/2` predicate (see next section) using the predicate `kakuro/0`.

Example Output

The printed output when posing the query `?- kakuro.` and inputting the input from above could be:

```
Please enter puzzle as matrix:
[[x/x , 23/x, 30/x , x/x  , x/x  , 27/x, 12/x, 16/x],
 [x/16, x   , x    , x/x  , 17/24, x   , x   , x   ],
 [x/17, x   , x    , 15/29, x    , x   , x   , x   ],
 [x/35, x   , x    , x    , x    , x   , 12/x, x/x ],
 [x/x , x/7 , x    , x    , 7/8  , x   , x   , 7/x ],
 [x/x , 11/x, 10/16, x    , x    , x   , x   , x   ],
 [x/21, x   , x    , x    , x    , x/5 , x   , x   ],
 [x/6 , x   , x    , x    , x/x  , x/3 , x   , x   ]].
```

```
+-------+-------+-------+-------+-------+-------+-------+-------+
| x /x  | 23/x  | 30/x  | x /x  | x /x  | 27/x  | 12/x  | 16/x  |
+-------+-------+-------+-------+-------+-------+-------+-------+
| x /16 |       |       | x /x  | 17/24 |       |       |       |
+-------+-------+-------+-------+-------+-------+-------+-------+
| x /17 |       |       | 15/29 |       |       |       |       |
+-------+-------+-------+-------+-------+-------+-------+-------+
| x /35 |       |       |       |       |       | 12/x  | x /x  |
+-------+-------+-------+-------+-------+-------+-------+-------+
| x /x  | x /7  |       |       | 7 /8  |       |       | 7 /x  |
+-------+-------+-------+-------+-------+-------+-------+-------+
| x /x  | 11/x  | 10/16 |       |       |       |       |       |
+-------+-------+-------+-------+-------+-------+-------+-------+
| x /21 |       |       |       |       | x /5  |       |       |
+-------+-------+-------+-------+-------+-------+-------+-------+
| x /6  |       |       |       | x /x  | x /3  |       |       |
+-------+-------+-------+-------+-------+-------+-------+-------+
Setting up constraints ... DONE
Solving constraints    ... DONE
+-------+-------+-------+-------+-------+-------+-------+-------+
| x /x  | 23/x  | 30/x  | x /x  | x /x  | 27/x  | 12/x  | 16/x  |
+-------+-------+-------+-------+-------+-------+-------+-------+
| x /16 | 9     | 7     | x /x  | 17/24 | 8     | 7     | 9     |
+-------+-------+-------+-------+-------+-------+-------+-------+
| x /17 | 8     | 9     | 15/29 | 8     | 9     | 5     | 7     |
+-------+-------+-------+-------+-------+-------+-------+-------+
| x /35 | 6     | 8     | 5     | 9     | 7     | 12/x  | x /x  |
+-------+-------+-------+-------+-------+-------+-------+-------+
| x /x  | x /7  | 6     | 1     | 7 /8  | 2     | 6     | 7 /x  |
+-------+-------+-------+-------+-------+-------+-------+-------+
| x /x  | 11/x  | 10/16 | 4     | 6     | 1     | 3     | 2     |
+-------+-------+-------+-------+-------+-------+-------+-------+
| x /21 | 8     | 9     | 3     | 1     | x /5  | 1     | 4     |
+-------+-------+-------+-------+-------+-------+-------+-------+
| x /6  | 3     | 1     | 2     | x /x  | x /3  | 2     | 1     |
+-------+-------+-------+-------+-------+-------+-------+-------+

yes
```