DM 509 Programming Languages

Fall 2010 Project (Part 1)

Department of Mathematics and Computer Science
University of Southern Denmark

November 22, 2010

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 overall project and get access to the final written exam.

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 second 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 using Blackboard's Assignment Hand-In functionality. Delivering by e-mail or to the teacher is only considered acceptable in case Blackboard cannot be used.

Deadline

Deliverable: December 6, 12:00

The Problem

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

- In each row or column, the nine numbers have to be from the set $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ and they must all be different.
- For each of the nine non-overlapping 3x3 blocks, the nine numbers have to be from the set $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ and they must all be different.

The following two figures show a Sudoku puzzle and its solution.

		8	1			5		
9	6			8				
	5			3	6		9	8
		7		6	9		1	2
		6	8		7	3		
8	1		4	5		6		
4	2		6	7			8	
				4			6	3
		5			8	7		

3	7	8	1	9	4	5	2	6
9	6	4	2	8	5	1	3	7
1	5	2	7	3	6	4	9	8
5	4	7	3	6	9	8	1	2
2	9	6	8	1	7	3	5	4
8	1	3	4	5	2	6	7	9
4	2	1	6	7	3	9	8	5
7	8	9	5	4	1	2	6	3
6	3	5	9	2	8	7	4	1

The Input

For input to your program, the Sudoku 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 two types:

- A term `x` signifies an empty cell, i.e., a cell that your solver needs to find a number for.
- A numeric term signifies a cell whose value is fixed, i.e., the corresponding cell needs to have this number in the solution that you find.

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

```
[[x, x, 8, 1, x, x, 5, x, x],
 [9, 6, x, x, 8, x, x, x, x],
 [x, 5, x, x, 3, 6, x, 9, 8],
 [x, x, 7, x, 6, 9, x, 1, 2],
 [x, x, 6, 8, x, 7, 3, x, x],
 [8, 1, x, 4, 5, x, 6, x, x],
 [4, 2, x, 6, 7, x, x, 8, x],
 [x, x, x, x, 4, x, x, 6, 3],
 [x, x, 5, x, x, 8, 7, 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:

```
[[3, 7, 8, 1, 9, 4, 5, 2, 6],
 [9, 6, 4, 2, 8, 5, 1, 3, 7],
 [1, 5, 2, 7, 3, 6, 4, 9, 8],
 [5, 4, 7, 3, 6, 9, 8, 1, 2],
 [2, 9, 6, 8, 1, 7, 3, 5, 4],
 [8, 1, 3, 4, 5, 2, 6, 7, 9],
 [4, 2, 1, 6, 7, 3, 9, 8, 5],
 [7, 8, 9, 5, 4, 1, 2, 6, 3],
 [6, 3, 5, 9, 2, 8, 7, 4, 1]]
```

The Task

Implement a predicate `solve/2` that takes an unsolved Sudoku 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 in the design section of your report. Then implement and test it.

You could for example choose one of the following approaches:

- Use a generate-and-test approach, i.e., enumerate solution candidates and test them until you find a solution.

- Generate constraints from the input and use constraint programming to solve them.
- Implement solving rules as used by human players and use brute-force only as a last resort.

The Foundations

There is a number of built-in predicates that you might find useful when building a Sudoku 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 Sudoku puzzles (`show/1`, works both on puzzles in input and in output form).

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

Example Output

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

```
Please enter puzzle as matrix:
[[x, x, 8, 1, x, x, 5, x, x],
 [9, 6, x, x, 8, x, x, x, x],
 [x, 5, x, x, 3, 6, x, 9, 8],
 [x, x, 7, x, 6, 9, x, 1, 2],
 [x, x, 6, 8, x, 7, 3, x, x],
 [8, 1, x, 4, 5, x, 6, x, x],
 [4, 2, x, 6, 7, x, x, 8, x],
 [x, x, x, x, 4, x, x, 6, 3],
 [x, x, 5, x, x, 8, 7, x, x]]
```

6

```
+-------+-------+-------+
| x x 8 | 1 x x | 5 x x |
| 9 6 x | x 8 x | x x x |
| x 5 x | x 3 6 | x 9 8 |
+-------+-------+-------+
| x x 7 | x 6 9 | x 1 2 |
| x x 6 | 8 x 7 | 3 x x |
| 8 1 x | 4 5 x | 6 x x |
+-------+-------+-------+
| 4 2 x | 6 7 x | x 8 x |
| x x x | x 4 x | x 6 3 |
| x x 5 | x x 8 | 7 x x |
+-------+-------+-------+
Setting up constraints ... DONE
Solving constraints    ... DONE
+-------+-------+-------+
| 3 7 8 | 1 9 4 | 5 2 6 |
| 9 6 4 | 2 8 5 | 1 3 7 |
| 1 5 2 | 7 3 6 | 4 9 8 |
+-------+-------+-------+
| 5 4 7 | 3 6 9 | 8 1 2 |
| 2 9 6 | 8 1 7 | 3 5 4 |
| 8 1 3 | 4 5 2 | 6 7 9 |
+-------+-------+-------+
| 4 2 1 | 6 7 3 | 9 8 5 |
| 7 8 9 | 5 4 1 | 2 6 3 |
| 6 3 5 | 9 2 8 | 7 4 1 |
+-------+-------+-------+

yes
```