SAARLAND UNIVERSITY ARTIFICIAL INTELLIGENCE
Prof. Jörg Hoffmann and Prof. Wolfgang Wahlster
Dr. Álvaro Torralba, Daniel Gnad, Marcel Steinmetz
Christian Bohnenberger, Cosmina Croitoru, Akram Elkorashy, Sophian Guidara,
Daniel Heller, Björn Mathis, Lukas Schaal, Julia Wichlacz

# Practical Sheet 1

Solutions due **May 31**, 23:59, in the Moodle system.

## Exercise 1. (20 Points)

Consider the following *generalized Sudoku* puzzle. The goal of this problem is to fill the empty cells in the board with numbers from 1 to 9 complying with the constraints listed below.

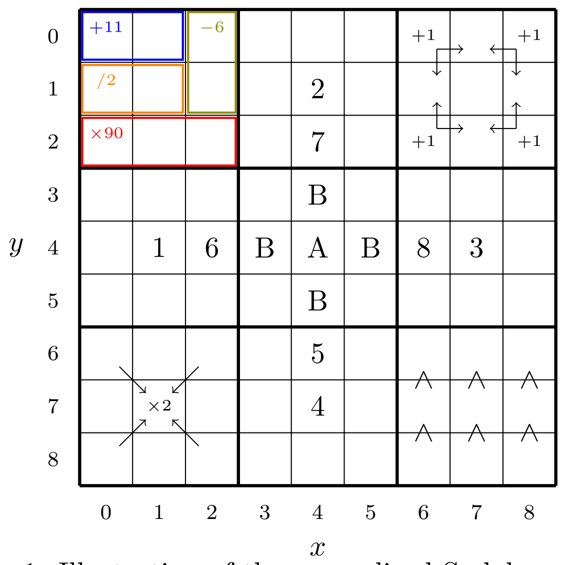

Figure 1: Illustration of the generalized Sudoku puzzle.

In what follows, we use $\langle i, j\rangle$ to denote the value of the cell with $x = i$ and $y = j$.

1. Typical Sudoku constraints: Numbers cannot be repeated in any row, column, or 3x3 square.
2. Upper left corner: The numbers must comply with the arithmetic expressions drawn in the figure. In particular:
   (a) $\langle 0, 0\rangle + \langle 1, 0\rangle = 11$

   (b) $\langle 2,0\rangle - \langle 2,1\rangle = 6$
   (c) $\frac{\langle 0,1\rangle}{\langle 1,1\rangle} = 2$
   (d) $\langle 0,2\rangle \times \langle 1,2\rangle \times \langle 2,2\rangle = 90$

3. Bottom right corner: Numbers must comply with the inequalities. In particular:
   (a) $\langle 6,6\rangle < \langle 6,7\rangle < \langle 6,8\rangle$
   (b) $\langle 7,6\rangle < \langle 7,7\rangle < \langle 7,8\rangle$
   (c) $\langle 8,6\rangle < \langle 8,7\rangle < \langle 8,8\rangle$

4. Upper right corner: For each of the corners of the 3x3 upper right square, one of the (horizontally or vertically) adjacent cells (within this square) must equal the value of the corner plus 1. Note that according to this definition of adjacency, each of the corners has exactly two adjacent cells. For example, for the upper left corner of the square, if $\langle 6,0\rangle = 4$, either $\langle 6,1\rangle = 5$ or $\langle 7,0\rangle = 5$.

5. Bottom left corner: The sum of the corners of the bottom left square must be equal to twice the center cell of the bottom left square. In other words: $\langle 0,6\rangle + \langle 0,8\rangle + \langle 2,8\rangle + \langle 2,6\rangle = 2 \times \langle 1,7\rangle$

6. Center: If the central cell "A" is an even number, then all cells labeled with "B" must be odd numbers. If "A" is an odd number, then all cells labeled with "B" must be even numbers.

7. Corners: The sum of the corners of the entire board must be equal to 10. In other words: $\langle 0,0\rangle + \langle 0,8\rangle + \langle 8,0\rangle + \langle 8,8\rangle = 10$

Your task is to model the generalized Sudoku as a CSP problem and let an actual CSP solver run on it. To do so, use the solver linked in the lecture slides (`http://minion.sourceforge.net`). Go to the "Download" section and download a version fitting your operating system (in the following description we assume it to be the Linux version; we have not tested any other versions, but assume that they will be called in a similar manner). On our system it was enough to just extract the downloaded archive – the executable is located in the folder "bin" and is called "minion".

(a) Create an empty file and model the CSP for the given task within this file. The file should be called "sudoku.minion".

(b) Run minion on your file. To do so, go to the "bin" directory and execute "./minion -findallsols <path-to-your-file>". Save the output in a file called "run.log". Note: Minion should find 60 solutions to problem, if you encoded all the above constraints correctly. If the output of the solutions takes extremely long (i.e., if a huge number of solutions are found), or there is no solution, it is very likely that have a bug in your model.

Put both files into an archive called “name1-name2-name3.zip”, where “name1”, “name2”, “name3” are the family names of all authors. Additionally, add a file called “authors.txt” to the archive that contains one line per author, detailing the full name and matriculation number. To upload the archive to the Moodle system, go to the corresponding assignment, click “Add submission”, and upload it. (obviously, you are not restricted to “*.zip”, but you can use any popular format, that you like). Note that only one author per group needs to do the submission.

**IMPORTANT** Please add comments to your model, i.e., for each of the constraint types (1-7), you should have a different section in your model. Briefly explain in the comments how you encoded each of the constraints (comments can be added by using a “#”). *If you do not put any comments into your model, or your model is difficult to understand with the given comments, we will substract points.*

**General guidelines** To get an idea of what the input files for minion look like, you can find some examples in the folder “test_instances”. Additionally, you might want to consult the manual, which is in “docs/Manual.pdf”. The basic outline of the resulting file is shown in Figure 2.
Here are some additional notes:

- The domain of a variable, or the domain of all elements of a vector, is given like this: {2..7} if the domain contains all integers in the range from 2 to 7.
- If you use some vector called `vectorname`, you can access an element at position (2,7) via vectorname[7,2]. Note that as usual the vectors start at 0.
- The constraint `eq(a,b)` means that the two variables `a` and `b` must have the same values. You might replace a variable by a position in a vector (as done in the `diseq` example) or by a constant (as done in the two `eq` examples).
- The constraint `diseq(a,b)` means that the two variables `a` and `b` must have different values.
- Minion supports not only binary constraints. Check the list of constraints that appear in the manual from which we highlight:
  - The “all different” constraint: `alldiff([a,b,c])` means that all elements in the given vector (here `a`, `b`, and `c`) must have different values.
  - Arithmetic constraints such as “sumleq”, “product”, specify constraints of the type $a + b + c \leq d$, or $a \times b = c$.

```
MINION 3

**VARIABLES**
DISCRETE vectorname[yrange,xrange] {domain}
DISCRETE variablename {domain}

**SEARCH**
PRINT[vectorname]

**TUPLELIST**
validvalues 2 3
1 2 3
3 2 1

**CONSTRAINTS**
eq(vectorname[0,3],5)
eq(variablename,3)
diseq(vectorname[2,0],vectorname[0,0])
alldiff([vectorname[0,0],vectorname[0,1],vectorname[0,2]])
table([vectorname[0,0], vectorname[0,1], vectorname[0,2]], validvalues)

**EOF**
```

Figure 2: Example

  - "table" constraint: Allows to provide a set of tuples that are valid for a set of variables. For example, one can ensure that the first three elements of vectorname[0,_] are either 1, 2, 3 or 3, 2, 1. Figure 2 contains an example of this restriction. The valid values can be specified separately in the TUPLELIST section by giving it a name (validvalues) the number of valid tuples and the number of relevant variables followed by the list of valid tuples, one per line. Then, the tuple list can be used in the table constraint specifying which variables should comply with one of these tuples.

- Auxiliary variables. Sometimes, it is useful to declare additional variables (e.g. variablename in our example) that are not really part of the solution we want to obtain but can be used as auxiliary variables. For example, if you need to perform a complex operation and no constraint fits your needs, you may use an auxiliary variable to store an intermediate result that can then be used in a constraint.