SAARLAND UNIVERSITY ARTIFICIAL INTELLIGENCE
Prof. Dr. Jörg Hoffmann, Dr. Daniel Fišer, Daniel Höller, Dr. Sophia Saller

# Practical Exercise Sheet 1

Deadline Friday, May 27, 23:59

**About the submission of this sheet.**

- You might submit the solutions to exercises in groups of up to 3 students.
- All students of a group need to be in the same tutorial.
- Hand in the solution **in CMS** and using "Team Groupings".
- Solutions need to be packaged into a `.zip` file that contains a single folder with name: `AI2022_PE1_mat1_mat2_mat3`, where `mat1, mat2, mat3` are the matriculation numbers of the students who submit together.
- This folder must contain the following files:
  - `authors.txt` listing names and matriculation numbers of all students of your group. Use one line per student and no spaces: Name;Matriculation number.
  - The Z3 files containing your solutions. **Do not rename** the Z3 template files.
  - Your `sudoku.log` file with the results.
- Do not add any other folder or sub folder, this means place all files directly into `AI2022_PE1_mat1_mat2_mat3`. Do not place any file outside of this folder.

## Exercise 1: CSP Modeling in Z3. (20 Points)

This exercise sheet is accompanied with source files provided through a separate `zip` archive. You can download this archive from the AI CMS under the *Materials* category.
The sheet is about modeling a Sudoku puzzle (see below) as CSP, and solving it using a CSP solver. To do so, we will be using the Z3 theorem prover.[1][2] In Linux, you can install Z3 with the command `sudo apt-get install z3`. To run Z3, you have to create a text file in Z3's input language, which you can then pass as command line argument to the Z3 executable. Printing additional information about the solving process can be done through the `-st` option, e.g., `z3 -st path/to/csp.z3`. The result of Z3 (`sat` or `unsat`) will be printed to the console.
An overview of the Z3 language fragment relevant for this sheet is shown in Table 1. As the input language of this solver allows to represent strictly more general problems than CSPs, **you must not use any statement that is not shown in this table**. An example file (the Coloring Australia example from the lecture) is available in CMS.

[1] `https://github.com/Z3Prover/z3`
[2] An introduction can be found in `https://www.philipzucker.com/z3-rise4fun/guide.html`.

Encode the following generalized Sudoku puzzle as CSP using the Z3 language fragment depicted in Table 1. When testing intermediate states of your encoding, please be aware that **solving models not including all constraints might take much more time than solving the fully specified one.**
The goal of this puzzle is to fill the empty cells in the board with numbers from 1 to 9 complying with the constraints listed below. We use $\langle i, j \rangle$ to denote the value of the cell with $x = i$ and $y = j$. An illustration of the following constraints can be found in Figure 1.

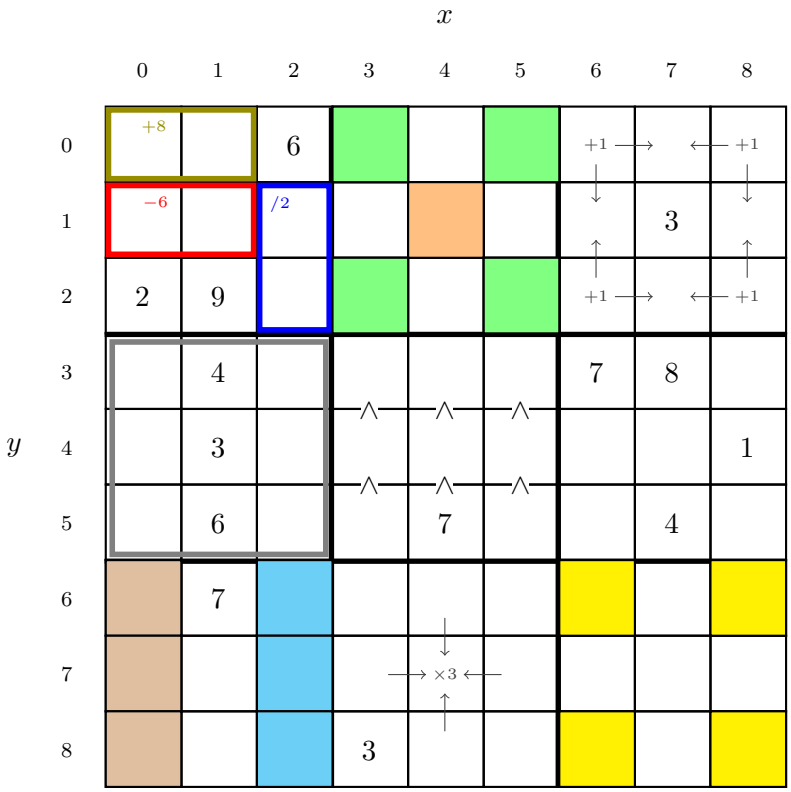


Figure 1: Illustration of the generalized Sudoku puzzle.

1. Typical Sudoku constraints:
   (a) Numbers cannot be repeated in any row, column, or 3x3 square.
   (b) Cells whose values are already specified must be assigned to the respective values.

2. Top left $3 \times 3$ square: The numbers must comply with the arithmetic expressions drawn in the figure:
   (a) $\langle 0,0\rangle + \langle 1,0\rangle = 8$
   (b) $\langle 0,1\rangle - \langle 1,1\rangle = 6$
   (c) $\langle 2,2\rangle / \langle 2,1\rangle = 2$
3. Top middle $3 \times 3$ square: The values of the green cells must be either all odd or all even. Moreover, if the green cells contain odd numbers, then the orange cell must contain an even number. If the green cells contain even numbers, then the orange cell must contain an odd number.
4. Top right $3 \times 3$ square: For every corner cell of this square, one of the horizontally or vertically adjacent cells must equal the value plus 1. When for example the cell $\langle 0,3\rangle = 3$, then either $\langle 0,4\rangle = 4$ or $\langle 1,3\rangle = 4$.
5. Middle left $3 \times 3$ square: The sum of all rows in this square must be equal, i.e., $\langle 0,3\rangle + \langle 1,3\rangle + \langle 2,3\rangle = \langle 0,4\rangle + \langle 1,4\rangle + \langle 2,4\rangle = \langle 0,5\rangle + \langle 1,5\rangle + \langle 2,5\rangle$.
6. Center $3 \times 3$ square: Numbers must comply with the inequalities. More specifically:
   (a) $\langle 3,3\rangle < \langle 3,4\rangle < \langle 3,5\rangle$
   (b) $\langle 4,3\rangle < \langle 4,4\rangle < \langle 4,5\rangle$
   (c) $\langle 5,3\rangle < \langle 5,4\rangle < \langle 5,5\rangle$
7. Bottom left $3 \times 3$ square: Multiplying the sums of the two indicated columns gives an odd number: $(\langle 0,6\rangle + \langle 0,7\rangle + \langle 0,8\rangle) \times (\langle 2,6\rangle + \langle 2,7\rangle + \langle 2,8\rangle)$ must be odd.
8. Bottom middle $3 \times 3$ square: The sum of the indicated cells must be equal to three times the value of the center cell. In other words: $\langle 4,6\rangle + \langle 3,7\rangle + \langle 4,8\rangle + \langle 5,7\rangle = 3 \times \langle 4,7\rangle$
9. Bottom right $3 \times 3$ square: at most one of the yellow cells may contain a value larger than 4.

| Statement | Description |
|---|---|
| **General** | |
| `(check-sat)` | Checks whether the CSP defined up to this point is satisfiable. |
| `(declare-const var Int)` | Declares a new variable with name `var`. |
| `(assert E)` | Adds boolean expression `E` as constraint. |
| `(get-value (E))` | Prints the value of `E`, where `E` can be an arbitrary expression such as constant, variable, function, or mathematical or boolean combination thereof (must occur after `(check-sat)`). |
| `(get-model)` | Prints all variable assignments (must occur after `(check-sat)`). |
| `(echo "message")` | Prints `message` to the console. |
| `; This is a comment` | Commenting. |
| **Mathematical Expressions** | |
| $c$ | Constants $c \in \mathbb{Z}$. |
| `var` | Evaluates to the value of variable `var`. |
| `(Board x1 y2)` | Evaluates to the value of the cell with coordinates $\langle 1, 2 \rangle$ |
| `(`$\circ$ $E_1$ `...` $E_n$`)` | Evaluates to $E_1 \circ E_2 \circ \cdots \circ E_n$, where $\circ$ can be any of `+`, `-`, and `*`. |
| **Boolean Expressions** | |
| `true` | Constant for true. |
| `false` | Constant for false. |
| `(not` $E$`)` | Negation of the boolean expression $E$. |
| `(and` $E_1$ `...` $E_n$`)` | Conjunction over the boolean expressions $E_1$ to $E_n$. |
| `(or` $E_1$ `...` $E_n$`)` | Disjunction over the boolean expressions $E_1$ to $E_n$. |
| `(`$\circ$ $E_1$ `...` $E_n$`)` | Is true iff for the evaluation of the expressions $E_1$ to $E_n$, it holds that $E_1 \circ E_2$ and $E_2 \circ E_3$, ..., and $E_{n-1} \circ E_n$, where $\circ$ can be any of `<`, `<=`, `>`, `>=`, and `=`. |
| `(distinct` $E_1$ `...` $E_n$`)` | Is true iff every expression $E_1$ to $E_n$ evaluates to a different value. |

Table 1: Z3 input language fragment you may use for the CSP modeling exercises.