THE UNIVERSITY OF SYDNEY

COMP3109 Assignment 2

Logic-Oriented Programming (10 marks)

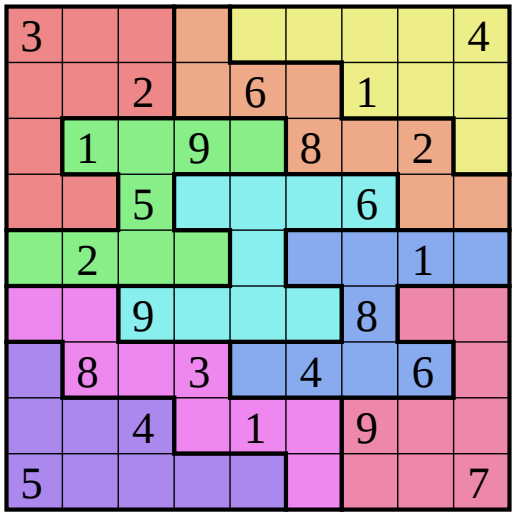
This first assignment is an **individual assignment**, and is due in Week 8 **on Friday, September 16, at 6pm**. Please pack the source code of this assignment into a zip file and submit it via eLearning. The documentation of the program should be done in form of comments. Please be verbose with your comments, i.e., explain your ideas thoroughly, saying in plain English how you have implemented the program. Not complying to documentation standards will give you less marks for your assignment.

In this assignment you will implement a Sudoku solver in Prolog. Your assignment should run on the workspaces on `edstem.com.au`.

Sudoku is a popular and contemporary class of puzzle. The objective is to fill a 9×9 grid with digits from 1 to 9. The grid is composed of nine 3×3 subgrids. A solution is *well-formed* iff (1) for each sub-grid all nine digits 1 to 9 are used and (2) in each column/row of the 9×9 grid all digits from 1 to 9 show up. An instance of the a Sudoku puzzle is given by providing a partially completed grid, for which there exists a unique and well-formed solution.

Some solver implementations for the classic Sudoku puzzle in Prolog can be looked up on the internet and there you can find sufficient explanations for implementing a solver for the classic problem in Prolog (i.e. it requires only few lines of Prolog, cf. Rosetta.org).

In this assignment the task is to write a solver for an extended version of Sudoku puzzles. The extension is called a *Jigsaw-Sudoku* which is variation of the classic puzzle. The extension permits that the sub-grids are of irregular shape. However, the same conditions apply, i.e., all digits from 1 to 9 need to show up in rows, columns, and in each sub-grid. An instance of a Jigsaw-Sudoku is given below (from WikiMedia.org):



The problem is specified by two lists: the first describes the partially completed grid, and the second

list describes the irregular shapes of the sub-grids. It consists of nine lists. Each list is a list of row-column coordinates of cells forming a sub-grid. For the example, the problem is given below:

```
problem( [[3,_,_,_,_,_,_,_,4],
          [_,_,2,_,6,_,1,_,_],
          [_,1,_,9,_,8,_,2,_],
          [_,_,5,_,_,_,6,_,_],
          [_,2,_,_,_,_,_,1,_],
          [_,_,9,_,_,_,8,_,_],
          [_,8,_,3,_,4,_,6,_],
          [_,_,4,_,1,_,9,_,_],
          [5,_,_,_,_,_,_,_,7]],
         [[[1,1],[1,2],[1,3],[2,1],[2,2],[2,3],[3,1],[4,1],[4,2]],
          [[1,4],[2,4],[2,5],[2,6],[3,6],[3,7],[3,8],[4,8],[4,9]],
          [[1,5],[1,6],[1,7],[1,8],[1,9],[2,7],[2,8],[2,9],[3,9]],
          [[3,2],[3,3],[3,4],[3,5],[4,3],[5,1],[5,2],[5,3],[5,4]],
          [[4,4],[4,5],[4,6],[4,7],[5,5],[6,3],[6,4],[6,5],[6,6]],
          [[5,6],[5,7],[5,8],[5,9],[6,7],[7,5],[7,6],[7,7],[7,8]],
          [[6,1],[6,2],[7,2],[7,3],[7,4],[8,4],[8,5],[8,6],[9,6]],
          [[6,8],[6,9],[7,9],[8,7],[8,8],[8,9],[9,7],[9,8],[9,9]],
          [[7,1],[8,1],[8,2],[8,3],[9,1],[9,2],[9,3],[9,4],[9,5]]]).
```

Task 1 (2 Mark)

Define a predicate `completegrid(S)` that holds for a sub-grid definition, if all cells of the $9x9$ time grid are defined by the sub-grid definition (i.e., sub-grids do not overlap and the union of all subgrids form the $9x9$ grid). Find either a counting argument or set argument, to define your predicate.

Invoking the predicate on the problem instance, i.e.,

```
? problem(X,S), completegrid(S).
true.
```

should return a true value. For a grid such as

```
completegrid([[[1,2],[1,2],[1,2],[2,1],[2,2],[2,3],[3,1],[4,1],[4,2]],
              [[1,4],[2,4],[2,5],[2,6],[3,6],[3,7],[3,8],[4,8],[4,9]],
              [[1,2],[1,6],[1,7],[1,8],[1,9],[2,7],[2,8],[2,9],[3,9]],
              [[3,2],[3,3],[3,4],[3,5],[4,3],[5,1],[5,2],[5,3],[5,4]],
              [[4,4],[4,5],[4,6],[4,7],[5,5],[6,3],[6,4],[6,5],[6,6]],
              [[5,6],[5,7],[5,8],[5,9],[6,7],[7,5],[7,6],[7,7],[7,8]],
              [[6,1],[6,2],[7,2],[7,3],[7,4],[8,4],[8,5],[8,6],[9,6]],
              [[6,8],[6,9],[7,9],[8,7],[8,8],[8,9],[9,7],[9,8],[9,9]],
              [[7,1],[8,1],[8,2],[8,3],[9,1],[9,2],[9,3],[9,4]]]).
false.
```

the predicate should return the false value because there are not all $9x9$ cells of the grid defined by the grid, e.g. $(1, 1)$, etc.

Note that the predicate just checks whether each cell is defined, it does not check whether the sub-grids are contiguous regions.

Task 2 (3 Mark)

Define a predicate `contiguousregion(S)` that holds for a definition of a region, if the region is contiguous (i.e., don't split into two regions with no connection via a grid cell).

For example the sub-grid

```
? contiguousgrid([[1,1],[1,2],[1,3],[2,1],[2,2],[2,3],[3,1],[4,1],[4,2]]).
true.
```

should hold because each cell is connected with another cell in the region in north/east/south/west direction on the grid and form a single region. The following subgrid is not a contiguous region because it consists of two separated contiguous regions, e.g.,

```
? contiguousgrid([[1,1],[1,2],[1,3],[1,4],[1,5],[1,6],[3,1],[4,1],[4,2]]).
false.
```

Task 3 (5 Mark)

Define a predicate `solve(G,S,X)` that solves the Jigsaw-Problem where G is the grid, S the sudoku field, and X the solution.