AIT Semantic and Declarative Technologies Course

Assignment 1: Solving a Sudoku puzzle

Description

This task is related to the well-known Sudoku puzzle.

A *Sudoku grid* is square matrix consisting of m rows and m columns where m itself is a square number, $m = k * k$. We refer to the elements of the matrix as the *fields* of the grid. The Sudoku grid subdivides to square *sub-grids* of k rows and k columns.

For the mathematically minded, the the top left fields of sub-grids have (row,column) coordinates $(i*k, j*k)$, for $i = 0, \ldots, k-1$ and $j = 0, \ldots, k-1$; assuming that the top left field of the whole grid has (row,column) coordinates $(0, 0)$.

An example grid for $k = 2$ is shown below, the sub-grid borders are indicated by double lines.

2	3	4	1
4		2	
	2	1	4
	4		2

Given a Sudoku grid, the task is to find a solution by filling in the empty fields with values between 1 and m in such a way that all values in each row, column and subgrid are different.

Data representation

The Sudoku grid is given as a list of rows, each row being a list of integers. You can assume (i.e. you don't have to check) that the Sudoku grid supplied to your program is correct in the following sense: it consists of m rows, each row is a list of m fields, and each field is an integer between 0 and m, where $m = k * k$. You can also assume that $1 \leq k \leq 10$, although probably you will not make use of this assumption.

The above example is supplied to your program as the following Prolog data structure:

```
[[2,3,  4,1],
 [4,0,  2,0],

 [0,2,  1,4],
 [0,4,  0,2]]
```

The integer 0 represents a field which is not yet filled in. Positive integers represent fields that are filled in.

Definitions

A Sudoku grid is said to be *inconsistent* if it contains a row or a column or a sub-grid, which, in turn, contains two occurrences of the same positive integer. A Sudoku grid is said to be *consistent* if it is not inconsistent.

The term *area* is used to denote any of the rows, columns or sub-grids. Using this terminology, we can state that a Sudoku grid is consistent if no area contains two occurrences of the same positive integer.

The above example grid is consistent. The following example grid is inconsistent, because the bottom-right sub-grid contains two occurrences of integer 1.

```
[[2,3,  0,0],
 [1,0,  2,3],

 [3,2,  1,4],
 [4,0,  0,1]]
```

A Sudoku grid is said to be a **refinement** of another Sudoku grid, if the former can be obtained by *filling in* (zero or more) fields of value 0 in the latter, i.e. replacing the 0 value by a positive integer.

A Sudoku grid is said to be **fully filled in**, if all its fields contain positive integers.

A Sudoku grid is said to be **complete**, if it is fully filled in and consistent.

Solving a sudoku grid means finding all its complete refinements.

Note that the a grid may be consistent and yet it may have no solutions. The following is an example of such a grid.

```
[[2,3,  0,0],
 [0,0,  0,0],

 [0,0,  1,0],
 [0,0,  4,0]]
```

No positive integer occurs more than once within any area (as each positive integer occurs exactly once in the whole grid), but any value between 1 and 4 assigned to field in row 1, column 3 would make the grid inconsistent.

The assignment

The assignment consists of three parts, the last is optional

1. Write a Prolog predicate `consistent/1` which takes a Sudoku grid and succeeds if, and only if, the grid is consistent. If the grid is consistent, the predicate should succeed exactly once.

   A Sudoku grid is represented by a Prolog term which is a list whose elements represent the rows of the grid in top-down order. Each row is represented by a list of integers, the fields in the given row, in left-to-right order.

   The predicate has the following specification (head comment):

   ```
   % consistent(+SGrid): For all areas of the Sudoku grid SGrid it holds
   % that all positive integers in the area are distinct.
   ```

2. Using the generate-and-test approach and relying on the predicate `consistent/1`, implement the following predicate for solving a Sudoku puzzle:

   ```
   % sudoku0(+Grid0, ?Grid): Grid is a complete refinement of the Sudoku grid Grid0
   ```

3. (Optional) Provide a more efficient implementation for the `sudoku0/2` predicate and name it `sudoku/2`

   In increasing order of ambition and efficiency you can consider the following approaches:

   a) **Fused generate-and-test:** Refine the input grid by filling in a single field, and immediately check the result for consistency. Continue this process until a (fully) filled in refinement is obtained.

   b) **Fill in fields that are unique:** Try to find empty fields that can only be assigned a single positive value while keeping the grid consistent.

   c) **Find integers that can be uniquely placed in an area:** Try to find integers that can only be placed into a single field of an area while keeping the grid consistent.

   d) **Keep the braching factor of the search tree low:** If you run out of deterministic refinements (such as listed as the two preceding items), try to find a field assigning which leads to the smallest number of consistent refinements.

Sample runs

```
| ?- consistent([[1]]).
yes
| ?- consistent([[2,3, 4,1],
                 [4,0, 2,0],
                 [0,2, 1,4],
                 [0,4, 0,2]]
               ).
yes
| ?- consistent([[2,3, 0,0],
                 [1,0, 2,3],
                 [3,2, 1,4],
                 [4,0, 0,1]]
               ).
no
| ?- consistent( [[0,0, 0,0], [0,0, 0,0], [0,0, 0,0], [0,0, 0,0]]).
yes
| ?- sudoku0([[2,3, 4,1],[4,0, 2,0],[0,2, 1,4],[0,4, 0,2]], Grid).
Grid = [[2,3,4,1],[4,1,2,3],[3,2,1,4],[1,4,3,2]] ? ;
no
```

Notes

- The following example shows how obtain the integer square root `K` of a square number `M` using the built-in predicate `is/2`:

  ```
  K is integer(sqrt(M))
  ```

- You may find useful some predicates in SICStus Prolog `library(lists)`, e.g. `transpose/2` and `sublist/5`. In solving this assignment you can freely use predicates from this library.

  In SWI Prolog `transpose/2` is part of `library(clpfd)`. You can load this library for the purpose of using `transpose/2`, but no other predicates should be used from `library(clpfd)`. Also, `sublist/5` is not available in SWI, you can use `list_sub/5` from Homework P4 instead.

  Ask the instructors if you would like to use other libraries.

  Any predicate you wrote as part of an earlier homework may be re-used here. Possible examples: `chop/3` from Homework P4 and `pairings/2` from Homework P3.

- Instructions for submitting solutions will be announced soon.