COMP30020 proj2 xuliny **TESTLOG** Page 1/1

```
Beginning tests at Thu 17 Oct 2019 18:36:49 AEDT
Num Test                                Secs Status    Score    Remark
--- ----                                ---- ------    -----    ------
  1 puzzle_solution(inout) ...          0.01   PASS   1.0/1.0
  2 puzzle_solution(inout) ...          0.00   PASS   1.0/1.0
  3 puzzle_solution(inout) ...          0.00   PASS   1.0/1.0
  4 puzzle_solution(inout) ...          0.01   PASS   1.0/1.0
  5 puzzle_solution(inout) ...          0.00   PASS   1.0/1.0
  6 puzzle_solution(inout) ...          0.01   PASS   1.0/1.0
  7 puzzle_solution(inout) ...          0.01   PASS   1.0/1.0

Total tests executed: 7
Total correctness : 7.00 / 7.00 = 100.00%
Marks earned : 100.00 / 100.00
Completed tests at Thu 17 Oct 2019 18:36:49 AEDT
```

```
%  File     : proj2.pl
%  Author   : XuLin Yang 904904 <xuliny@student.unimelb.edu.au>
%  Origin   : Thu Sep 26 14:54:20 2019
%  Purpose  : Prolog program to solve the math puzzle
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%|This code is for providing the implementation for the math puzzle solver.
%% It is defined in one main predicate:
%%     puzzle_solution(+Puzzle).
%%     puzzle_solution/1.
%%
%% The program is for solving the "Math Puzzle" with a n*n squares of grids
%% and a number for each row and column (a graphical representation is given
%% below). With constraints:
%%      1. the left-top to bottom right diagonal requires same digit
%%      2. each column sums or products to the given row number
%%      3. each row sums or products to the given row number
%%      4. each row's digits are different
%%      5. each column's digits are different
%%      6. each cell of the puzzle matrix is 1-9 digit
%%
%% example use:
%% ?- Puzzle=[[0,14,10,35],[14,_,_,_],[15,_,_,_],[28,_,1,_]],
%% |    puzzle_solution(Puzzle).
%% Puzzle = [[0, 14, 10, 35], [14, 7, 2, 1], [15, 3, 7, 5], [28, 4, 1, 7]] ;
%% false.
%% The input Puzzle data structure is made up of an ignored top left corner,
%% bounded header numbers, puzzle matrix with partially bounded or unbounded
%% cells.
%% Note: 1) Some cells of the puzzle can be given a digit in the input.
%%       Otherwise input has "_" for the predicate to find a digit for unbound
%%       cell.
%%       2) The first list of Puzzle is the column number for each math puzzle
%%       column ("0" has no meaning just for distinguishing a cell place).
%%       The rest lists' first elemens are the row number for each row. The
%%       rest is the matrix to be found digits for.
%%       3) In the above example, [14,10,35] in first list is the column
%%       numbers. [14,15,18] is the first element of the rest lists is the row
%%       numbers. A digit 1 is given in row 3 column 2.
%%       Represent it graphically:
%%       +---+---+---+---+                    +---+---+---+---+
%%       |  0| 14| 10| 35|                    |  0| 14| 10| 35|
%%       +---+---+---+---+                    +---+---+---+---+
%%       | 14|  _|  _|  _|                    | 14|  7|  2|  1|
%%       +---+-----------+  == solved to =>   +---+-----------+
%%       | 15|  _|  _|  _|                    | 15|  3|  7|  5|
%%       +---+-----------+                    +---+-----------+
%%       | 18|  _|  1|  _|                    | 18|  4|  1|  7|
%%       +---+-----------+                    +---+-----------+
%%
%% The program approachs the solution by:
%%      0. unpack the input data structure to a comfortable data structure for
%%      me to process the solution
%%      1. unifies diagonal to the same digit
%%      2. generate each row that satisfies constraint 3) 4)
%%      Note: it unifies all variables in Matrix with digit in range [1,9] at
%%      this step.
%%      3. then do the same thing to each column to satisfy constraint 2) 5)
```

```
%%      4. ensure each cell in the n*n puzzle is ground.
%%
%% The program assumes:
%%      1. The input puzzle matrix is 2*2, 3*3 or 4*4 size (i.e.: n range from
%%         2-4 inclusively).
%%      2. The puzzle has at least one solution or returns false if not
%%         solvable when a proper Puzzle is given.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% According to https://github.com/SWI-Prolog/swipl-devel/issues/281
% Put below at the top of the file to avoid illegal multibyte sequence warning
% caused by code in library clpfd.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
:- encoding(utf8).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%           SWI PrologâM-^@M-^Ys ConstraintLogic Programming Library
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
:- ensure_loaded(library(clpfd)).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%           Major predicate
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% puzzle_solution(+Puzzle).
%% It takes a Puzzle and unifies it so that the Puzzle satisfies the
%% constraints listed above. The predicate holds when the Puzzle is the
%% representation of a solved maths puzzle. Otherwise the result is false.
puzzle_solution/1.
puzzle_solution(Puzzle) :-
    % step 0: unpack the input data structure
    unpack_puzzle(Puzzle, RowNumbers, ColumnNumbers, Matrix),

    % step 1: unifies diagonal to the same digit for constraint 1)
    same_diagonal_digits(Matrix),

    % step 2: unifies each row with digits that satisfies constraint 3) 4)
    maplist(generate_and_validate_row, RowNumbers, Matrix),

    % step 3: unifies each column with digits that satisfies constraint 2) 5)
    validate_columns(Matrix, ColumnNumbers),

    % step 4: check each cell in Matrix is ground to satisfy constraint 6)
    maplist(label, Matrix).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   step 0: unpack the input data structure helper predicates
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% unpack_puzzle(+Puzzle, -RowNumbers, -ColumnNumbers, -Matrix).
%% It takes a Puzzle input data structure as mentioned above. Holds when
%% Puzzle input is unpacked to:
%%   1. row numbers: list of number for each row digits to sum or product to
%%   2. column numbers: list of number for each column digits to sum or
%%                      product to.
%%   3. matrix of the puzzle: the n*n List of bounded or unbound cells.
:- discontiguous unpack_puzzle/4.
unpack_puzzle(Puzzle, RowNumbers, ColumnNumbers, Matrix) :-
    Puzzle = [FirstList|RestLists],
```

```
    FirstList = [_|ColumnNumbers],
    % 1 is because index starts from 1
    % split RowNumbers and puzzle Matrix from RestLists
    lists_nth1(1, RestLists, RowNumbers, Matrix).

%% lists_nth1(+N, +Lists, -Elem, -Rest).
%% This is nth1/4 version for a list of lists. It takes N which is the 1-based
%% index of elment in the list to be selected repeadly from Lists which is a
%% list of lists. Unifies the list of selected elements to Elem and lefted list
%% of lists to Rest.
%% Note is assumes Lists be a n*n matrix
:- discontiguous lists_nth1/4.
lists_nth1(_, [], [], []).
lists_nth1(N, [L|Lists], [Elem|Elements], [Rest|Rests]) :-
    nth1(N, L, Elem, Rest),
    lists_nth1(N, Lists, Elements, Rests).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   step 1: unifies diagonal to the same digit for constraint 1)
%           helper predicates
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% same_diagonal_digits(+Matrix).
%% It takes a n*n Matrix. Holds when the matrix's top-left to bottom-right
%% diagonal has the same 1-9 digits.
%% Note: It only unifies unbounded elements on diagonal with digits 1-9 for
%% better performance purpose.
:- discontiguous same_diagonal_digits/1.
same_diagonal_digits(Matrix) :-
    diagonal(Matrix, Diagonal),
    digitize_list(Diagonal),
    same(Diagonal).

%% diagonal(+Matrix, -Diagonal).
%% It takes a Matrix which is a n*n list of lists. Unify Diagonal as a list
%% which is the top-left to bottom-right diagonal of the Matrix.
:- discontiguous diagonal/2.
diagonal(Matrix, Diagonal) :-
    length(Matrix, Length),
    % 1 is because index starts from 1.
    % get the diagonal of matrix by recursively calls nth1 on matrix with
    % increasing index to be selected.
    recur_nth1_increased_index(Matrix, Length, 1, Diagonal).

%% recur_nth1_increased_index(+Lists, +N, +Index, -Elem).
%% It takes Lists which is a n*n matrix, N which is the size of the matrix,
%% and Index is the accumulator of the element's 1-based index to be selected
%% from cur L (head of Lists). Index (accumulator) increases at each depth
%% from 1-N. Unify Elem as the diagonal of the Lists (n*n matrix).
:- discontiguous recur_nth1_increased_index/4.
recur_nth1_increased_index([], _, _, []).
recur_nth1_increased_index([L|Lists], Length, Acc, [Elem|Elements]) :-
    Next #= Acc + 1,
    nth1(Acc, L, Elem),
    recur_nth1_increased_index(Lists, Length, Next, Elements).

%% digitize_list(+List).
%% It takes a list. Hold when the variables in List are digitized from 1 to 9.
:- discontiguous digitize_list/1.
```

```
digitize_list(List) :- List ins 1..9.

%% same(+List).
%% It takes a list. Holds when the list's elements are the same or the list is
%% empty.
:- discontiguous same/1.
same([]).
same([_]).
same([X,X|Xs]) :- same([X|Xs]).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   step 2: unifies each row with digits that satisfies constraint 3) 4)
%           helper predicates
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% generate_and_validate_row(+RowNumber, +Row).
%% It takes a row number for the row to be summed or producted to and a list of
%% digits. Bound variables in Row from [1-9]. Holds when the Row satisfies
%% constraints 3) 4) 6).
%% Note: all cells in Matrix is now unified with a digit
:- discontiguous generate_and_validate_row/2.
generate_and_validate_row(RowNumber, Row) :-
    digitize_list(Row),
    validate_row(RowNumber, Row).

%% validate_row(+RowNumber, +Row).
%% It takes a row number which is the sum or product of the digits in the Row
%% and a Row of mixture of bounded digits or unbouned variables. Holds when the
%% Row satisfies constraint 3) 4).
:- discontiguous validate_row/2.
validate_row(RowNumber, Row) :-
    all_different(Row),
    sum_or_product_to(Row, RowNumber).

%% sum_or_product_to(+Row, +RowNumber).
%% It take a Row which is a list of digits and a RowNumber. Holds when
%% satisfy constraint 3).
:- discontiguous sum_or_product_to/2.
sum_or_product_to(Row, RowNumber) :-
    sum(Row, #=, RowNumber); product_list(Row, RowNumber).

%% product_list(+List, -Product).
%% It takes a List of digits. Unify Product as the product of all digits in
%% the List.
%% Note: The product of the empty list is sensibly defined as 1 so that, when
%% building the product over several lists, empty lists blend in naturally and
%% do not change any results.
:- discontiguous product_list/2.
product_list(List, Product) :- foldl(product, List, 1, Product).

%% product(+X, +Y, -Z).
%% It takes a number X and a number Y. Unify Z as the product of X and Y.
:- discontiguous product/3.
product(X, Y, Z) :- Z #= X * Y.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   step 3: unifies each column with digits that satisfies constraint 2) 5)
%           helper predicates
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%% validate_columns(+Matrix, +ColumnNumbers).
%% It takes a puzzle Matrix which is a n*n list of lists and a list of
%% ColumnNumbers for the columns of the Matrix to summed or producted to. Holds
%% when it satisfies constraint 2) 5).
%% Note: With the help of transpose/2, validate columns in Matrix can reuse
%% validate_row/2.
:- discontiguous validate_columns/2.
validate_columns(Matrix, ColumnNumbers) :-
    transpose(Matrix, MatrixTrans),
    maplist(validate_row, ColumnNumbers, MatrixTrans).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   End by XuLin Yang, 904904
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```