COMP90048 proj2 dstern **LOG** Page 1/1

```
Num Test                                Secs Status    Score    Remark
--- ----                                ---- ------    -----    ------
  1 puzzle_solution(inout) ...          0.01   PASS   1.0/1.0
  2 puzzle_solution(inout) ...          0.01   PASS   1.0/1.0
  3 puzzle_solution(inout) ...          0.03   PASS   1.0/1.0
  4 puzzle_solution(inout) ...          0.06   PASS   1.0/1.0
  5 puzzle_solution(inout) ...          0.04   PASS   1.0/1.0
  6 puzzle_solution(inout) ...          0.06   PASS   1.0/1.0
  7 puzzle_solution(inout) ...          0.03   PASS   1.0/1.0

Total tests executed: 7
Total correctness : 7.00 / 7.00 = 100.00%
Marks earned : 10.50 / 10.50
```

```
:- use_module(library(clpfd)).
:- use_module(library(apply)).

% ------------------------------------------------------------------------
% AUTHORSHIP, DESCRIPTION
%
% Solver for 'Maths Puzzles', written in Prolog, as described below.
%
% David Stern
% Student at the University of Melbourne
% Master of Engineering (Software)
% October 2017
% Copyright: Â© 2017 David Stern.  All rights reserved.
% ^(Code only, NOT the project specification)
%
% ------------------------------------------------------------------------
% LIBRARY USE
%
% Uses both the apply and clpfd (Constraint logic programming over finite
% domains) libraries, as well as the SWIPL libray, for concise, efficient code.
%
% Uses the automatically loaded nth0/3 from the SWIPL lists library.
%
% Frequently uses maplist/2 from the apply library.
%
% clpfd library functions used:
% - Frequently uses the elegant arithmetic constraints #=/2, #=</2 and #>=/2.
%   (these are true relations and can be used in all directions, subsumes is/2,
%   etc.)
% - puzzle_solution uses transpose/2.
% - no_repeat_row uses all_distinct/1.
% - digit_row uses ins/2 .
% - ground_vars uses label/1 .
% - row_has_valid_heading uses sum/3.
%
% The contributors to these wonderful libraries are thanked for their
% incredibly useful code.
% ------------------------------------------------------------------------
% PROJECT SPECIFICATION: PETER SCHACHTE - COMP30020/COMP90048
% DECLARATIVE PROGRAMMING - SEMESTER 2 - 2017 - THE UNIVERSITY OF MELBOURNE
%
% "A maths puzzle is a square grid of squares, each to be filled in with a
% single digit 1âM-^@M-^S9 (zero is not permitted) satisfying these constraints:
%   - each row and each column contains no repeated digits;
%   - all squares on the diagonal line from upper left to lower right contain
%     the same value; and
%   - the heading of reach row and column (leftmost square in a row and topmost
%     square in a column) holds either the sum or the product of all the digits
%     in that row or column.
%
% Note that the row and column headings are not considered to be part of the
% row or column, and so may be filled with a number larger than a single digit.
% The upper left corner of the puzzle is not meaningful.
%
% When the puzzle is originally posed, most or all of the squares will be
% empty, with the headings filled in. The goal of the puzzle is to fill in all
% the squares according to the rules. A proper maths puzzle will have at most
% one solution.
%
```

```
% Here is an example puzzle as posed (left) and solved (right):"
%  ____ ____ ____ ____    ____ ____ ____ ____
% |    | 14 | 10 | 35 |  |    | 14 | 10 | 35 |
% | 14 |    |    |    |  | 14 |  7 |  2 |  1 |
% | 15 |    |    |    |  | 15 |  3 |  7 |  5 |
% | 28 |    |    |    |  | 28 |  1 |  1 |  7 |
%  ---- ---- ---- ----    ---- ---- ---- ----
%  (^ As shown in the Project Specification)
%
% The 'Puzzle' referred to in predicates defined below is an NxN list of lists
% of equal length that fit the specification above and below.
%
% "You will write Prolog code to solve maths puzzles. Your program should
% supply a predicate puzzle solution(Puzzle) that holds when Puzzle is the
% representation of a solved maths puzzle."
%
% "A maths puzzle (Puzzle) will be represented as a lists of lists, each of the
% same length, representing a single row of the puzzle. The first element of
% each list is considered to be the header for that row. Each element but the
% first list in the puzzle is considered to be the header of the corresponding
% column of the puzzle. The first element of the first element of the list is
% the corner square of the puzzle, and thus is ignored."
%
% Assumption 1: "When puzzle_solution/1 is called, its argument will be a
% proper list of proper lists, and all the header squares of the puzzle (plus
% the ignored corner square) are bound to integers. Some of the other squares
% in the puzzle may also be bound to integers, but the others will be unbound."
%
% Assumption 2: "This code will only be tested with proper puzzles, which have
% at most one solution. If the puzzle is not solvable, the predicate should
% fail, and it should never succeed with a puzzle argument that is not a valid
% solution."
% ------------------------------------------------------------------------------

% ------------------------------------------------------------------------------
% FINALLY, THE PROGRAM
%
% puzzle_solution(Puzzle) holds when Puzzle is the representation of a solved
% maths puzzle. Uses the predicates defined below to achieve this.
% Process:
% - First, the equal length of the puzzle is ascertained using maplist (equal
%   length not explicitly included in the assumptions).
% - We then transpose the Puzzle to create CPuzzle, which we use with
%   predicates designed to test only rows, needed to satisfy constraints on
%   both columns and rows.
% - We then ensure that the elements of Puzzle are digits, and then ensure
% - Constraint 0 (as above)
% - Constraint 1 (as above)
% - Constraint 2 (as above)
% Then, once the variables are limited to domains that satisfy these
% constraints, we ground them.
puzzle_solution(Puzzle) :-
    maplist(same_length(Puzzle), Puzzle),
    transpose(Puzzle, CPuzzle),
    digits(Puzzle),
    no_repeats(Puzzle), no_repeats(CPuzzle),
    equal_diagonal(Puzzle),
    valid_row_headings(Puzzle), valid_row_headings(CPuzzle),
    ground_vars(Puzzle).
```

```
% ------------------------------------------------------------------------

% ------------------------------------------------------------------------
% Constraint 0: Each square of the rows and columns must contain a single
%               digit 1-9 (no 0s). Headings must be digits
%
% digits/1 takes a Puzzle and holds when Constraint 0 holds.
% First unpacks the values of Puzzle to access the header row (Row0) and
% the puzzle's rows (Rows). Then ensures that the elements of Row0 are >= 1,
% and then ensures Rows are rows of digits, as defined above.
digits(Puzzle) :-
    Puzzle = [[_|Row0]|Rows],
    maplist(#=<(1), Row0),
    maplist(digit_row, Rows).

% digit_row/1 takes a row of a Puzzle, as described above, and holds when the
% elements of the row satisfy Constraint 0. First ensures the Heading is >= 1,
% & then sets the domain of the row elements to digits between 1-9 (inclusive).
digit_row([Heading|Row]) :-
    Heading #>= 1,
    Row ins 1..9.
% ------------------------------------------------------------------------

% ------------------------------------------------------------------------
% Constraint 1: Each row and each column contain no repeated digits.
%
% no_repeats/1 takes a Puzzle, and holds when there are no repeated digits
% in any of the rows. A transposed Puzzle can be used to check columns contain
% no repeated digits.
no_repeats(Puzzle) :-
    Puzzle = [_|Rows],
    maplist(no_repeat_row, Rows).

% no_repeat_row/1 takes a row of a Puzzle and holds when the elements of that
% row are all distinct (using all_distinct/1 from clpfd).
no_repeat_row([_|Row]) :- all_distinct(Row).
% ------------------------------------------------------------------------

% ------------------------------------------------------------------------
% Constraint 2: All squares on the diagonal are equal.
%
% equal_diagonal/1 takes a Puzzle, discards its header row, takes the first
% element of the first row (1th element) of the puzzle, sets X to the value of
% the first non-heading element of the first row, and holds when the other
% elements of the diagonal are equal to this element.
equal_diagonal([_Headerrow|[Firstrow|Rows]]) :-
    nth0(1, Firstrow, X),
    equal_diag(Rows, 2, X).

% equal_diag/1 take rows of a puzzle, an integer Nth indicating the index of
% the column that must be equal to the next argument, X, an integer.
% Holds where the nth element of Row and the nth+1 element of the next row (if
% any) is equal to X.
equal_diag([], _, _).
equal_diag([Row|Rows], Nth, X) :-
    nth0(Nth, Row, X),
    Next #= Nth + 1,
    equal_diag(Rows, Next, X).
% ------------------------------------------------------------------------
```

```
% ------------------------------------------------------------------------
% Constraint 3: Heading of each row and col holds either the sum or product of
% all digits in that row or col.
%
% valid_row_headings/1 takes a Puzzle and ensures the headings of each row is
% either the sum or the product of its elements. Used with a transposed Puzzle
% to check that the column headings are also valid.
valid_row_headings(Rows) :-
    Rows = [_|Puzzlerows],
    maplist(row_has_valid_heading, Puzzlerows).

% row_has_valid_heading/1 takes a row of a Puzzle and is true either when the
% elements are the sum of the heading, or when they are the product of the
% heading.
row_has_valid_heading([Heading|Row]) :- sum(Row, #=, Heading).
row_has_valid_heading([Heading|Row]) :- is_product(Row, 1, Heading).

% is_product/3 takes a list of elements, an accumulator value (initially 1 for
% the first call), and a value X, and holds where the product of the list
% elements, multiplied by the accumulator value, is equal to X.
is_product([], X, X).
is_product([Y|Ys], A, X) :-
    A1 #= Y * A,
    is_product(Ys, A1, X).
% ------------------------------------------------------------------------

% ------------------------------------------------------------------------
% Finally, ensure all terms are ground.
%
% ground_vars/1 takes a Puzzle and holds when all of the variables in the
% puzzle are ground.
ground_vars([_Headingrow|Rows]) :- maplist(label, Rows).
% ------------------------------------------------------------------------
```