COMP90048 proj2 hongfeiy1 **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) ...	0.00	PASS	1.0/1.0	
3	puzzle_solution(inout) ...	0.06	PASS	1.0/1.0	
4	puzzle_solution(inout) ...	0.06	PASS	1.0/1.0	
5	puzzle_solution(inout) ...	0.02	PASS	1.0/1.0	
6	puzzle_solution(inout) ...	1.22	PASS	1.0/1.0	
7	puzzle_solution(inout) ...	1.51	PASS	1.0/1.0	

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

```
/* COMP30020 Declarative Programming Assignment 2
 * By Hongfei Yang <hongfeiy1@student.unimelb.edu.au>.
 *
 * This program solves math puzzles of square grid of squares,
 * each to be filled in with a single digit 1-9 to satisfy
 * constraints Defined by these:
 *    1. each row and each column contains no repeated digits;
 *    2. all squares on the diagonal line from upper left to lower right
 *       contain the same value; and
 *    3. 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
 *
 * This program solves the problems by first resolving the diagonal part
 * of this puzzle. If the diagonal has any ground terms (which is an
 * integer), it will fill the entire diagonal with this number. If
 * there is no ground terms, this program then fills the puzzle with
 * numbers from 1-9 only as its constraints, trying to fill other part
 * of the puzzle of other constraints.
 *
 * This program will try to fill the puzzle row by row, starting from
 * the top row to the bottom row. Each row must be filled with
 * constraints of no repeating digits and the 0th index of the row is
 * the product or the sum of the rest of the numbers in the row.
 *
 * After filling a puzzle, the program will transpose the puzzle to
 * check for vertical coloumn constranints. Same constranints applied to
 * the vertical coloumn. The final solved matrix needs to satisfy both
 * constraints.
 *
 */


:- ensure_loaded(library(clpfd)).

/*
 * Main part of the programm. This function solves the puzzle by
 * applying constraints of diagonal rules, row rules and column rules.
 *
 * puzzle_solution(Puzzle)
 * True if Puzzle is a square grid of integers that has a solution.
 */
puzzle_solution([]).
puzzle_solution([Heading|Body]) :-

    %% first fill the diagonal part of the puzzle.
    fill_diagonal([Heading|Body], [Heading|R_Body]),

    %% then fill the body part of the puzzle
    fill_puzzle(R_Body, R_body),

    %% transpose the puzzle
    transpose([Heading|R_body], [_Heading_t|R_Body_t]),

    %% then check for column constraints to get the final solved puzzle
    correct_row_puzzle(R_Body_t),

    Body = R_body.
```

```
/*
 * fill_diagonal(PrefilledPuzzle, ResultPuzzle).
 *
 * Fill the diagonal part of the puzzle. The first argument is the
 * original puzzle, the second argument is the puzzle with diagonal
 * filled in.
 *
 * The diagonal is filled using numbers from get_diagonal. Diagonals
 * will be filled row by row.
 */
fill_diagonal([Heading|Body], [Heading|R_body]) :-
    get_diagonal([Heading|Body], Diagonal),
    fill_diag_row(Body, 1, R_body, Diagonal).

/*
 * fill_diag_row(Original_puzzle, Index, Result_puzzle, Diagonal)
 *
 * Fill the diagonal composition of Original_puzzle at each row,
 * with Diagonal being the value to be filled, Index is the index
 * of diagonal in this row, to get Result_puzzle.
 *
 */
fill_diag_row([], _, [], _).
fill_diag_row([Row|Rest], A, [R_row|R_rest], Diagonal) :-

    %% replace the diagonal part in each row
    replace(Row, A, Diagonal, R_row),
    A1 is A + 1,

    %% the rest of the rows must be filled as well
    fill_diag_row(Rest, A1, R_rest, Diagonal).


/*
 * Replace a list with Elt at index marked by Index
 * replace(original_list, Index, Elemt, Result_row)
 */

replace([], _, _, []).
replace([Head|Tail], Index, Elt, [R_head|R_tail]) :-
    Index =:= 0 ->
        R_head = Elt,
        R_tail = Tail

    ;   Index1 is Index - 1,
        R_head = Head,
        replace(Tail, Index1, Elt, R_tail).

/*
 * Get the diagonal value of a prefilled puzzle.
 * Diagoal_value may be an integer, or could be a range of integers from
 * 1-9
 * get_diagonal(prefilledPuzzle, Diagonal_value).
 */
get_diagonal([_Heading|Body], Diagonal) :-
    get_diagonal(Body, 1, Diagonal).

%!  If no diagonal value found, diagonal value can be anything from 1-9
get_diagonal([], _, Diagonal) :-
```

```
    member(Diagonal, [1,2,3,4,5,6,7,8,9]).
%!  otherwise diagonal is defined in the puzzle already
get_diagonal([Row|Rest], A, Diagonal) :-
    nth0(A, Row, D0),

    (ground(D0) ->
        Diagonal = D0
    ;   A1 is A + 1,
        get_diagonal(Rest, A1, Diagonal)
    ).


/*
 * Check if one row satisfy the puzzle constraints.
 * True if the 0th element of the row is the product or sum
 * of the rest of the number in row.
 */
correct_row([Head|Tail]) :-
    sum_list(Tail, Head).
correct_row([Head|Tail]) :-
    product_list(Tail, Head).

/**
 * product_list(List, Product).
 * True if Product is the product of the all member in List
 * without the 0th element.
 */
product_list([], 1).
product_list([Head|Tail], Product) :-
    product_list(Tail, Rest),
    Product is Head * Rest.

/**
 * correst_row_puzzle(Puzzle)
 *
 * True if Puzzle has unique elements in each row
 * and each row satifisfies the product / sum rules
 */
correct_row_puzzle([]).
correct_row_puzzle([CurrRow|Rest]) :-
    correct_row(CurrRow),
    unique_elt_row(CurrRow),
    correct_row_puzzle(Rest).

/**
 * fill_row(Original_row, Result_row)
 *
 * Fill Original_row, with numbers from 1-9 if
 * there is any empty slots to get Result_row.
 */
fill_row([], []).
fill_row([Head|Tail], [R_head|R_tail]) :-
    (\+ ground(Head) ->
        member(R_head, [1,2,3,4,5,6,7,8,9])
    ;   R_head is Head),
    fill_row(Tail, R_tail).

/**
 * unique_elt_row(Row).
```

```
 *
 * True if Row has no repeat elements.
 */
unique_elt_row([]).
unique_elt_row([Head|Tail]) :-
    \+ member(Head, Tail),
    unique_elt_row(Tail).

/**
 * fill_puzzle(Original_puzzle, Result_puzzle)
 *
 * Fill Original_puzzle with digits to form Result_puzzle.
 * Each row must be filled with non-repeated digits from 1-9.
 * The 0th element of each row must be the sum or product
 * of the rest elements in the same row.
 */
fill_puzzle([], []).
fill_puzzle([Row|Rest], [[R_row_heading|R_row_body]|R_rest]) :-
    length(Rest, N),
    length(R_rest, N),
    fill_row(Row, [R_row_heading|R_row_body]),
    correct_row([R_row_heading|R_row_body]),
    unique_elt_row(R_row_body),
    fill_puzzle(Rest, R_rest).
```