LAAI project code module 1

Gian Marco Baroncini

1 Black Cards

```
% exercise 1: Nel mazzo di 52 carte ve ne sono 26 rosse e 26
% nere. Il mazzo viene separato in due mazzetti: il
% primo di 25 carte, il secondo di 27.
% Se il primo mazzetto contiene 12 carte rosse,
% quante carte nere ci sono nel secondo mazzetto?


% Nc is the total number of cards
% Nfd is the number of cards in first deck
% Nfdred is the number of red cards in first deck
% Black_cards is the variable which will contain the answer: black cards in second deck


n_cards(Nc, Nfd, Nfdred, Black_cards) :-
 0 is Nc mod 2,
   Nfd =< Nc,
   Nfdred =< Nc / 2,
   Black_cards is ((Nc / 2) - (Nfd - Nfdred)).


% n_cards(52, 25, 12, Black_cards).
```

2 The Maze

```
% 2. Il labirinto
% Nel disegno seguente vedete lo schema di un
% labirinto. Massimo entra nel labirinto in alto a
% sinistra (dove c la freccia) ed esce a destra (in
% corrispondenza dellaltra freccia), seguendo le
% linee tratteggiate e non passando mai pi di una
% volta dallo stesso pallino rosso.
% Sapendo che la distanza tra due pallini  di 10
% metri, qual  la distanza massima che
% Massimo potr percorrere?

labirinto.png

:-use_module(library(lists)).

% data set of labyrinth

link(one, two).
link(two, three).
link(one, five).
link(two, six).
link(three, seven).
link(four, five).
link(five, six).
link(six, seven).
link(seven, eight).
link(four, nine).
link(five, ten).
link(six, eleven).
link(seven, twelve).
link(eight, thirteen).
link(nine, ten).
link(ten, eleven).
link(eleven, twelve).
link(twelve, thirteen).
link(ten, fourteen).
link(eleven, fifteen).
link(twelve, sixteen).
link(fourteen, fifteen).
link(fifteen, sixteen).

% starting point = 1, ending point = 13
% Visited captures the constraint "only one passage for each point"

path_starting(Visited, Visited_final, Path) :-
    link(one, X),
    not(member(X, Visited)),
    append(Visited, [one, X], Visited_new),
    sub_path(X, _, Visited_new, Visited_final, Path).
sub_path(X, thirteen, Visited, Visited_final, Path) :-
    link(X, thirteen),
    not(member(thirteen, Visited)),
    append(Visited, [thirteen], Path),
    length(Path, N),
    Visited_final is (N - 1)*10.

    % the -1 remove the +10 of the first insertion 'append([one, X]...'
    % because for the alg it counts 20 but is only one movement

sub_path(X, Y, Visited, Visited_final, Path) :-
    link(X, Y),
    not(member(Y, Visited)),
    append(Visited, [Y], Visited_new),
    sub_path(Y, _, Visited_new, Visited_final, Path).

sub_path(X, Y, Visited, Visited_final, Path) :-
    link(Y, X),
    not(member(Y, Visited)),
    append(Visited, [Y], Visited_new),
    sub_path(Y, _, Visited_new, Visited_final, Path).



% print all solutions
% findall(Visited_final, path_starting([], Visited_final, Path), Sol), max_list(Sol, Max).

% print solutions with path
% path_starting([], Visited_final, Path).
```

3 The Resistance

```
:-use_module(library(lists)).

% .defined as x, - as y

letter(x, Found, Updated) :-append(Found, ["E"], Updated).
letter(y, Found, Updated) :-append(Found, ["T"], Updated).
letter(x, y, Found, Updated) :-append(Found, ["A"], Updated).
letter(x, x, Found, Updated) :-append(Found, ["I"], Updated).
letter(y, x, Found, Updated) :-append(Found, ["N"], Updated).
letter(y, y, Found, Updated) :-append(Found, ["M"], Updated).
letter(y, x, x, Found, Updated) :-append(Found, ["D"], Updated).
letter(y, y, x, Found, Updated) :-append(Found, ["G"], Updated).
letter(y, x, y, Found, Updated) :-append(Found, ["K"], Updated).
letter(y, y, y, Found, Updated) :-append(Found, ["O"], Updated).
letter(x, y, x,Found, Updated) :-append(Found, ["R"], Updated).
letter(x, x, x, Found, Updated) :-append(Found, ["S"], Updated).
letter(x, x, y, Found, Updated) :-append(Found, ["U"], Updated).
letter(x, y, y, Found, Updated) :-append(Found, ["W"], Updated).
letter(y, x, x, x, Found, Updated) :-append(Found, ["B"], Updated).
letter(y, x, y, x, Found, Updated) :-append(Found, ["C"], Updated).
letter(x, x, y, x, Found, Updated) :-append(Found, ["F"], Updated).
letter(x, x, x, x, Found, Updated) :-append(Found, ["H"], Updated).
letter(x, y, y, y, Found, Updated) :-append(Found, ["J"], Updated).
letter(x, y, x, x, Found, Updated) :-append(Found, ["L"], Updated).
letter(x, y, y, x, Found, Updated) :-append(Found, ["P"], Updated).
letter(y, y, x, y, Found, Updated) :-append(Found, ["Q"], Updated).
letter(x, x, x, y, Found, Updated) :-append(Found, ["V"], Updated).
letter(y, x, x, y, Found, Updated) :-append(Found, ["X"], Updated).
letter(y, x, y, y, Found, Updated) :-append(Found, ["Y"], Updated).
letter(y, y, x, x, Found, Updated) :-append(Found, ["Z"], Updated).




%dictionary(["H","E","L","L"]).
%dictionary(["H","E","L","L","O"]).
%dictionary(["O","W","O","R","L","D"]).
%dictionary(["W","O","R","L","D"]).
%dictionary(["T","E","S","T"]).

dictionary(["D","U"]).
dictionary(["T","E","T","E"]).
dictionary(["B","A","C"]).
dictionary(["D","U","C"]).
dictionary(["B","A","N","N"]).


find_match([], Found, Result) :-append(Found, [], Result).

find_match([X|L], Found, Result) :-
    letter(X, Found, Updated),
    find_match(L, Updated, Result).

find_match([X,Y|L], Found, Result) :-
    letter(X, Y, Found, Updated),
    find_match(L, Updated, Result).

find_match([X,Y,Z|L], Found, Result):-
    letter(X, Y, Z, Found, Updated),
    find_match(L, Updated, Result).

find_match([X,Y,Z,W|L], Found, Result):-
    letter(X, Y, Z, W, Found, Updated),
    find_match(L, Updated, Result).


decode_results([], FinalWords, Result_decoding) :-append(FinalWords, [],
Result_decoding).

decode_results([X|L], FinalWords, Results_decoding) :-
    findall(Found, decode_single_result(X, [], [], Found), Res),
    append(Res, FinalWords, Up_FinalWords),
    decode_results(L, Up_FinalWords, Results_decoding).


decode_single_result([], _, Words, Found) :-append(Words, [], Found).

decode_single_result([X], Memory, Words, Found) :-
    append(Memory, [X], NewWord), dictionary(NewWord), append(Words, [NewWord],
    UpWords), decode_single_result([], Memory, UpWords, Found).

decode_single_result([X,Y|L], Memory, Words, Found) :-
    append(Memory, [X], NewWord), dictionary(NewWord), append(Words, [NewWord],
    UpWords), decode_single_result([Y|L], [], UpWords, Found) ;
    append(Memory, [X], NewWord), decode_single_result([Y|L], NewWord, Words, Found).


% first sequence:
% ......-...-..---.-----.-..-..-.. ==>
% x,x,x,x,x,x,y,x,x,x,y,x,x,y,y,y,x,y,y,y,y,y,x,y,x,x,y,x,x,y,x,x
% HELL, HELLO, OWORLD, WORLD, TEST
%
% second sequence:
% -....--.-. ==> y,x,x,x,x,y,y,x,y,x
% BAC, BANN, DUC, DU TETE
%
% ?- findall(Result, find_match([INSERT_SEQUENCE_OF_x,y_HERE], [], Result),
% All_Results), decode_results(All_Results, [], Results_decoding).
```