TD de *Prolog et programmation par contraintes* n° 4

(Correction)

Exercice 1 [**Carrés latins**] Un carré latin est une matrice $n \times n$ qui contient les nombres de 1 à n^2, disposés de telle façon que, pour chaque ligne et chaque colonne de la matrice, la somme des nombres de la ligne ou de la colonne en question donne le même résultat.

Par exemple, voici un carré latin de taille 3:

8	3	4
1	5	9
6	7	2

Le but de cet exercice est d'implementer un algorithme "générer et tester" pour engendrer les carrés latins.

Une configuration est une liste de n^2 éléments, qui contient une permutation des entiers de 1 à n^2. Les n premiers éléments de la liste réprésentent la première ligne de la matrice, les éléments du $n+1$-ème au $(2*n)$-ième la deuxième ligne et ainsi de suite.

Donc, la matrice ci-dessus est réprésentée par la liste `[8,3,4,1,5,9,6,7,2]`.

1. Définir un prédicat `genere(+N,-L)` qui réussit ssi `L` est la liste `[1,2,...,N`^{2}`]`.
2. Définir un prédicat `perm(+L,-G)` qui réussit ssi la liste `G` est une permutation de la liste `L`.
3. Définir un prédicat `nombre_latin(+N,-M)` qui réussit ssi `M` est la somme des éléments d'une ligne d'un carré latin de taille `N` (il s'agit évidemment de n'importe quelle ligne, ou colonne; la division entière s'écrit `//`, infixe)

   Définir les prédicats:

   - `somme_ligne(+L,+N,+I,-R)` qui réussit ssi la sommme des éléments de la `I`-ème ligne de la configuration `L` d'un carré latin de taille `N` (c.à.d. des éléments $(N*(I-1))+1$,...,$N*I$ de la liste `L`) est `R`.
   - `somme_colonne(+L,+N,+I,-R)`, l'analogue pour la somme des éléments de la `I`-ème colonne.

   Une définition de `somme_ligne` **et** `somme_colonne` **se trouve dans** `/ens/bucciare/sommes.pl`**.**
4. Définir un prédicat `test_lignes(+L,+N,+V)` qui réussit ssi `L` est une configuration pour un carré latin de taille `N`, tel que la somme des éléments sur chaque ligne de la matrice réprésentée par `L` est `V`. On supposera que `L` est effectivement une configuration.
5. Définir un prédicat `test_colonnes(L,N,V)`, analogue au précedent.
6. Définir un prédicat `carre_latin(+N,-L)` qui réussit si `L` est (la réprésentation d') un carré latin de taille `N`.
7. Définir une requête pour trouver toutes les configurations qui représentent des carrés latins de taille 3. Quel est leur nombre?
8. Pour les carrés latins de taille 4, l'approche "générer et tester" montre ses limites. Pourquoi? Définir un prédicat `carre_latin_4(L)` pour trouver les carrés latins de taille 4, plus efficace.

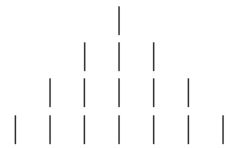
Correction :

```
concat([],L,L).
concat([A|G],L,[A|R]):-concat(G,L,R).

liste(1,[1]).
liste(N,L):-N>1, M is N-1, liste(M,G), concat(G,[N],L).

genere(N,L):-M is N*N, liste(M,L).

perm([],[]).
perm([X|L],Z):-perm(L,W),insertion(X,W,Z).

insertion(X,L,[X|L]).
insertion(X,[Y|L],[Y|G]):-insertion(X,L,G).

nombre_latin(N,M):-M is (N*(N*N+1))// 2.

somme_premiers(L,0,0).
somme_premiers([X|L],N,R):-P is N-1,somme_premiers(L,P,S),R is S+X.

elimine(L,0,L).
elimine([X|L],N,J):-P is N-1, elimine(L,P,J).

somme_ligne(L,N,1,R):-somme_premiers(L,N,R).
somme_ligne(L,N,M,R):-elimine(L,N,J),P is M-1,somme_ligne(J,N,P,R).

somme_colonne(L,N,M,R):-P is M-1, elimine(L,P,J),somme_colonne_aux(J,N,R).

somme_colonne_aux([X|L],N,X):-length(L,P),P<N.
somme_colonne_aux([X|L],N,R):-length(L,P),P>=N, elimine([X|L],N,J),
somme_colonne_aux(J,N,Q),R is X+Q.

lignes_ok(L,N,V):-ligne_aux(L,N,N,V).
ligne_aux(L,N,0,V).
ligne_aux(L,N,M,V):-somme_ligne(L,N,M,V),P is M-1,ligne_aux(L,N,P,V).


colonnes_ok(L,N,V):-colonne_aux(L,N,N,V).
colonne_aux(L,N,0,V).
colonne_aux(L,N,M,V):-somme_colonne(L,N,M,V),P is M-1,colonne_aux(L,N,P,V).

carre_latin(N,L1):-nombre_latin(N,M),genere(N,L),perm(L,L1),lignes_ok(L1,N,M),colonnes_ok(L1,N,M).

==========================OPTIMISE

concat([],L,L).
concat([A|G],L,[A|R]):-concat(G,L,R).

liste(1,[1]).
liste(N,[N|L]):-N>1, M is N-1, liste(M,L).

genere(N,L):-M is N*N, liste(M,L).

perm([X],[X]).
```

```
perm([X|L],Z):-perm(L,W),insertion(X,W,Z).

insertion(X,L,[X|L]).
insertion(X,[Y|L],[Y|G]):-insertion(X,L,G).

nombre_latin(N,M):-M is (N*(N*N+1))// 2.

member(X,[X|_]):-!.
member(X,[_|L]):-member(X,L).

enlever(X,[X|L],L):-!.
enlever(X,[Y|L],[Y|Q]):-enlever(X,L,Q).

ligne_possible(1,M,L,[M],Q):-member(M,L),enlever(M,L,Q).
ligne_possible(N,M,[X|L],[X|F],R):-N>1,X<M,P is M-X,Q is N-1,
ligne_possible(Q,P,L,F,R).

ligne_possible(N,M,[X|L],F,[X|R]):-N>1,ligne_possible(N,M,L,F,R).

choix_ligne(L,N,R):-nombre_latin(N,M),
ligne_possible(N,M,L,L1,R1),
ligne_possible(N,M,R1,L2,R2),
ligne_possible(N,M,R2,L3,R3),
perm(L1,C1),perm(L2,C2),perm(L3,C3),perm(R3,C4),
concat(C1,C2,P1),
concat(P1,C3,P2),
concat(P2,C4,R).

elimine(L,0,[],L).
elimine([X|L],N,[X|K],J):-P is N-1, elimine(L,P,K,J).

somme_colonne(L,N,M,R):-P is M-1, elimine(L,P,J),somme_colonne_aux(J,N,R).

somme_colonne_aux([X|L],N,X):-length(L,P),P<N.
somme_colonne_aux([X|L],N,R):-length(L,P),P>=N, elimine([X|L],N,J),
somme_colonne_aux(J,N,Q),R is X+Q.

colonnes_ok(L,N,V):-colonne_aux(L,N,N,V).
colonne_aux(L,N,0,V).
colonne_aux(L,N,M,V):-somme_colonne(L,N,M,V),P is M-1,colonne_aux(L,N,P,V).

carre_latin_4(R):-nombre_latin(4,M),genere(4,L),
choix_ligne(L,4,R),colonnes_ok(R,4,M).

/* [16,15,14,13,12,11,10,9,8,7,6,5,4,3,2,1] */
```

Exercice 2 [**Le jeu des 16 allumettes**]

Configuration initiale:



Règle du jeu: les deux joueurs, chacun à son tour, prélèvent une quantité (non nulle) d'allumettes d'une (et une seule) rangée.

Fin du jeu: le joueur qui prélève la dernière allumette a perdu.

1. choisir une représentation des configurations du jeu.
2. définir un prédicat `move(+Config_Source,Config_Destination)` qui implémente la règle du jeu.
3. définir un prédicat `gagne(+Configuration)`, tel que le but `gagne(c).` a succes si et seulement si il existe une stratégie gagnante à partir de la configuration `c`. Voici par exemple 3 configurations gagnantes:

4. vérifier que la configuration initiale du jeu est perdante.
5. définir un prédicat `jeu(+Configuration)` qui permet à l'utilisateur de jouer contre le programme, comme premier joueur et à partir de la configuration donnée. Le programme joue une stratégie optimale (c'est à dir que, dès que le programme joue sur une configuration gagnante, le programme gagne).

Correction :

```
/* il gioco dei 16 stecchini. Una configurazione e una lista di 4 interi. All inizio [1,3,5,7].*/
/* le mosse possibili si fanno con un predicato a due posti; la macchina gioca contro un utente*/
/* al quiale si chiede la prima mossa.*/



moins(7,0). moins(7,1). moins(7,2). moins(7,3). moins(7,4). moins(7,5). moins(7,6).
moins(6,0). moins(6,1). moins(6,2). moins(6,3). moins(6,4). moins(6,5).
moins(5,0). moins(5,1). moins(5,2). moins(5,3). moins(5,4).
moins(4,0). moins(4,1). moins(4,2). moins(4,3).
moins(3,0). moins(3,1). moins(3,2).
moins(2,0). moins(2,1).
moins(1,0).


move([X,Y,Z,T],[X,Y,Z,U]):-moins(T,U).
move([X,Y,Z,T],[X,Y,U,T]):-moins(Z,U).
move([X,Y,Z,T],[X,U,Z,T]):-moins(Y,U).
move([1,Y,Z,T],[0,Y,Z,T]).
```

```
perd([1,0,0,0]).
perd([0,1,0,0]).
perd([0,0,1,0]).
perd([0,0,0,1]).
gagne([0,0,0,0]).


gagne(L):-move(L,G), perd(G).
perd(L):- not(gagne(L)).


next(X,Y):-move(X,Y),perd(Y).

montre_ligne(0):-nl,!.
montre_ligne(N):-write('|'),M is N-1,montre_ligne(M).

montre_pos([X,Y,Z,T]):-montre_ligne(X),montre_ligne(Y),montre_ligne(Z),montre_ligne(T).


bouge(X,Y):-write('de quelle rangée?'),nl,read(X),
            write('combien?'),nl,read(Y).

/* change(Current,Rangee,Combien,Result)*/

change([],[]).
change([X|L],1,N,[Y|L]):-Y is X-N.
change([X|L],M,N,[X|G]):- M>1, P is M-1, change(L,P,N,G).

verifie(L,X,Y,Z):-change(L,X,Y,Z),move(L,Z).

somma([],0).
somma([M|L],N):-somma(L,K),N is K+M.

finale(L):-somma(L,N), N=<1.

utilisateur(L):- montre_pos(L),
(finale(L)->(nl,write('fin'),nl,exit);bouge(X,Y)),
verifie(L,X,Y,Z)
,montre_pos(Z),next(Z,Next),
utilisateur(Next).
```