Programmation en logique BE 1

2. Démarrage relies.pl

```
chemin(X,Y) :- relies(X,Y).
chemin(X,Y) :- relies(X,Z), chemin(Z,Y).
relies(l,n).
relies(p,l).
```

3. Manipulation de listes : somme

```
somme([],0).
somme([Debut | Fin],Somme) :-
   somme(Fin, S),
   Somme is Debut + S .
```

4. Manipulation de listes : fusion

```
fusion(List,[],List).
fusion([],List,List).
fusion([P1 |L1], [P2 | L2], [P1 |L3]) :-
   P1 < P2,
   fusion(L1, [P2 | L2], L3).

fusion([P1 |L1], [P2 | L2], [P2 |L3]) :-
   P2 =< P1,
   fusion([P1 | L1], L2, L3).
```

5. Manipulation de listes : inverse

```
inverse(L1, L2) :- inverse(L1, [], L2).
inverse([], Etsil, Etsil).
inverse([P|List], List2, Etsil) :-
   inverse(List, [P | List2], Etsil).

inverse1([],[]) .
inverse1([P |L1],L2) :-
   inverse1(L1, L3),
   append(L3, [P], L2).
```

6. Systèmes de réécriture : dérivation symbolique

```
derive(U+V, X, DU + DV) :-
   derive(U, X, DU),
   derive(V, X, DV).
derive(U-V, X, DU - DV) :-
   derive(U, X, DU),
   derive(V, X, DV).

derive(U*V, X, DU*V+DV*U) :-
   derive(U, X, DU),
   derive(V, X, DV).

derive(U/V, X, (DU*V-DV*U)/V^2) :-
   derive(U, X, DU),
   derive(V, X, DV).

derive(U^P, X, P*U^(P-1)*DU) :-
   derive(U, X, DU).

derive(X,X,1).
derive(Y,X,0) :-
   atomic(Y),
   Y \== X.

simplification(Exp, Res) :-
   Exp =.. [_, A1, A2],
   number(A1),
   number(A2),
   Res is Exp.

simplification(Exp, Res) :-
   Exp =.. [Op, A1, A2],
   simplification(A1, Res1),
   simplification(A2, Res2),
   SExp =.. [Op, Res1, Res2],
   Exp \== SExp,
   simplification(SExp, Res).

simplification(0*_, 0).
simplification(_*0, 0).

simplification(0+E, E).
simplification(E+0, E).

simplification(1*E, E).
simplification(E*1, E).

simplification(E,E).

deriver(U, X, Res) :-
   derive(U, X, Exp),
   simplification(Exp, Res).
```

7. Coloriage

```
colorie :-
   tous_les_pays(Liste),
   colorie(Liste).

colorie(Liste) :-
   colorie(Liste, Resultat),
   affiche(Resultat).

colorie(Liste, Resultat) :-
   colorie(Liste, [], Resultat).

colorie([], Precedents, Precedents).

colorie([Prem|Reste], Precedents, Resultat) :-
   couleur(Coul),
   not(incompatible(Prem, Coul, Precedents)),
   colorie(Reste, [[Prem, Coul]|Precedents], Resultat).

incompatible(Pays, Couleur, [[Voisin, Couleur]|_]) :-
   voisins(Pays, Voisin).
incompatible(Pays, Couleur, [_|Reste]) :-
   incompatible(Pays, Couleur, Reste).

tous_les_pays(L) :-
   setof(Pays, Autre^voisins(Pays, Autre), L).

affiche([]).
affiche([[Pays,Couleur] |Reste]) :-
   writef('%w -> %w\n', [Pays, Couleur]),
   affiche(Reste).

voisins(X,Y) :- voisin(X,Y).
voisins(X,Y) :- voisin(Y,X).

couleur(bleu).
couleur(rouge).
couleur(vert).
```

Programmation par contraintes

1. Somme des éléments d’une liste

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

somme([],0).
somme([Debut | Fin],Somme) :-
   somme(Fin, S),
   Somme #= Debut + S.
```

2. Coloriage

```
colorie :-
   tous_les_pays(Liste),
   colorie(Liste).

colorie(Liste) :-
   findall([V,_],member(V,Liste),L),
   bagof(G, D^member([D,G],L), Al),
   Al ins 1..6,
   contraindre(L),
   labeling([], Al),
   affiche(L).

contraindre([]).

contraindre([[PremPays, PremCoul]| Reste]) :-
   contraindreVoisin(PremPays, PremCoul, Reste),
   contraindre(Reste).

contraindreVoisin(_, _, []).

contraindreVoisin(PremPays, PremCoul, [[Pays, Coul] | Reste]) :-
   voisins(PremPays, Pays),
   PremCoul #\= Coul,
   contraindreVoisin(PremPays, PremCoul, Reste).

contraindreVoisin(PremPays, PremCoul, [[Pays, _] | Reste]) :-
   not(voisins(PremPays, Pays)),
   contraindreVoisin(PremPays, PremCoul, Reste).

tous_les_pays(L) :-
   setof(Pays, Autre^voisins(Pays, Autre), L).

affiche([]).
affiche([[Pays,CouleurIndex] |Reste]) :-
   Couleurs = [bleu, rouge, vert, blanc, jaune],
   nth1(CouleurIndex, Couleurs, Couleur),
    writef('%w -> %w\n', [Pays, Couleur]),
    affiche(Reste).

voisins(X,Y) :- voisin(X,Y).
voisins(X,Y) :- voisin(Y,X).
```

3. Ordonnancement

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

tache(a, 7, []).
tache(b, 3, [a]).
tache(c, 1, [b]).
tache(d, 8, [a]).
tache(e, 2, [d,c]).
tache(f, 1, [d,c]).
tache(g, 1, [d,c]).
tache(h, 3, [f]).
tache(j, 2, [h]).
tache(k, 1, [e,g,j]).
tache(fin, 0, [a,b,c,d,e,f,g,h,j,k]).

ordo(LT) :-
    findall([Nom, _], tache(Nom, _, _),LT),
    contraindre(LT,LT),
    transpose(LT, [Nom, Dates]),
    Dates ins 1..30,
    member([fin, DebFin], LT),
    labeling([], [DebFin]),
    affiche(LT).

contraindre([], _).

contraindre([[PT, DPT] | Reste], LT) :-
    tache(PT, _, ListePrecedent),
    contraindrePrecedent(DPT, ListePrecedent, LT),
    contraindre(Reste, LT).

contraindrePrecedent(_, [], _).

contraindrePrecedent(DPT, [Premier | Suivant], LT) :-
    member([Premier, Debut], LT),
    tache(Premier, Duree, _),
    DPT #>= Debut + Duree,
    contraindrePrecedent(DPT, Suivant, LT).

affiche([]).
affiche([[Nom,Date] |Reste]) :-
    writef('%w -> %w\n', [Nom, Date]),
    affiche(Reste).
```

4. N Reines

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

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
   Constraint posting.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

n_queens(N, Qs) :-
        length(Qs, N),
        Qs ins 1..N,
        safe_queens(Qs).

safe_queens([]).
safe_queens([Q|Qs]) :- safe_queens(Qs, Q, 1), safe_queens(Qs).

safe_queens([], _, _).
safe_queens([Q|Qs], Q0, D0) :-
        Q0 #\= Q,
        abs(Q0 - Q) #\= D0,
        D1 #= D0 + 1,
        safe_queens(Qs, Q0, D1).


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
   Animation.

   For each N of the domain of queen Q, a reified constraint of the form

      Q #= N #<==> B

   is posted. When N vanishes from the domain, B becomes 0. A frozen
   goal then emits PostScript instructions for graying out the field.
   When B becomes 1, the frozen goal emits instructions for placing
   the queen. On backtracking, the field is cleared.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

animate(Qs) :- animate(Qs, Qs, 1).

animate([], _, _).
animate([_|Rest], Qs, N) :-
        animate_(Qs, 1, N),
        N1 #= N + 1,
        animate(Rest, Qs, N1).

animate_([], _, _).
animate_([Q|Qs], C, N) :-
        freeze(B, queen_value_truth(C,N,B)),
        Q #= N #<==> B,
        C1 #= C + 1,
        animate_(Qs, C1, N).

queen_value_truth(Q, N, 1) :- format("~w ~w q\n", [Q,N]).
queen_value_truth(Q, N, 0) :- format("~w ~w i\n", [Q,N]).
queen_value_truth(Q, N, _) :- format("~w ~w c\n", [Q,N]), false.


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
   PostScript definitions.

   Sample instructions, with these definitions loaded:

   2 init   % initialize a 2x2 board
   1 1 q    % place a queen on 1-1
   1 2 q
   1 2 c    % remove the queen from 1-2
   2 2 i    % gray out 2-2
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

postscript -->
    "/init { /N exch def 340 N div dup scale -1 -1 translate \
            /Palatino-Roman 0.8 selectfont 0 setlinewidth \
            1 1 N { 1 1 N { 1 index c } for pop } for } bind def \
     /r { translate 0 0 1 1 4 copy rectfill 0 setgray rectstroke } bind def \
     /i { gsave 0.5 setgray r grestore } bind def \
     /q { gsave r 0.5 0.28 translate (Q) dup stringwidth pop -2 div 0 moveto \
          1 setgray show grestore } bind def \
     /c { gsave 1 setgray r grestore } bind def\n".

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
   Communication with gs.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

show(N, Options, Qs) :-
        N #> 0,
        n_queens(N, Qs),
        open(pipe('gs -dNOPROMPT -g680x680 -dGraphicsAlphaBits=2 -r144 -q'),
             write, Out, [buffer(false)]),
        tell(Out),
        phrase(postscript, Ps),
        format("~s ~w init\n", [Ps,N]),
        call_cleanup((animate(Qs),labeling(Options, Qs)), close(Out)).


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
   Examples:

   ?- n_queens(8, Qs), labeling([ff], Qs).
   Qs = [1, 5, 8, 6, 3, 7, 2, 4] ;
   Qs = [1, 6, 8, 3, 7, 4, 2, 5] .

   ?- n_queens(100, Qs), labeling([ff], Qs).
   Qs = [1, 3, 5, 57, 59, 4, 64, 7, 58|...] .

   ?- show(8, [ff], Qs).
   Qs = [1, 5, 8, 6, 3, 7, 2, 4] .

   ?- show(N, [ff], Qs).
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
```