Guillaume Lahaie
LAHG04077707

TP2

INF2160, groupe 30
Date de remise: 26 avril 2013

Voici le listing du fichier Agencement.pl:

```
/*********************************************************************
* Cours : INF2160
* Session : Hiver 20013
* Objet : Travail pratique 2
* Titre : Module de gestion d'un agencement
*
* Auteur : Bernard Lefebvre
*
* Modifié par Guillaume Lahaie
*             LAHG04077707
*             Dernière modification: 17 avril 2013
*
*             Les prédicats lesIds,lesComposantsDuType, estSrictementInclus
*             et les voisins ont été définis pour le TP2. Les prédicats
*             lesComposantsDuType2 et sousListe ont aussi été définis
*             pour la réalisation des prédicats mentionnés.
*********************************************************************/

laDimension(Idc,D) :- composant(_,Idc,D,_,_).
laPosition(Idc,P) :- composant(_,Idc,_,P,_).
leType(Idc,Ty) :- composant(_,Idc,_,_,Ty).

% lesComposantsDuBloc(Bloc,Compos)
% unifie Compos à la liste des composants localisés dans ce bloc
lesComposantsDuBloc([axCmp(_,Composant)|AxCmps],[Composant|Composants]) :-
                    lesComposantsDuBloc(AxCmps,Composants).
lesComposantsDuBloc([],[]).

% faireBloc(Axe,Composants,Bloc)
% unifie Bloc à une bloc formé à l'aide d'une liste de composants sur l'axe Axe
faireBloc(Axe,[C|Cs],[axCmp(Axe,C)|AxCmps]) :- faireBloc(Axe,Cs,AxCmps).
faireBloc(_,[],[]).

% intersectionBlocs(Bloc1,Bloc2,BlocI)
% unifie BlocI à intersection de Bloc1 et de Bloc2, les composants communs aux 2 blocs
% se retrouvent dans l'intersection indépendemment de l'axe sur lequel ils se trouvent
intersectionBlocs([axCmp(Axe,C)|AxCmps],Bloc,[axCmp(Axe,C)|AxCmpsI]) :-
lesComposantsDuBloc(Bloc,Compos),
member(C,Compos), !, intersectionBlocs(AxCmps,Bloc,AxCmpsI).
intersectionBlocs([_|AxCmps],Bloc,BlocI) :-
intersectionBlocs(AxCmps,Bloc,BlocI).
```

```
intersectionBlocs([],_,[]).

% unionBlocs(Bloc1,Bloc2,BlocU)
% unifie BlocI à l'union de Bloc1 et de Bloc2, les composants des 2 blocs
% se retrouvent dans l'union une seule fois indépendemment de l'axe sur lequel ils se trouvent
unionBlocs([axCmp(_,C)|AxCmps],Bloc,BlocU) :-
lesComposantsDuBloc(Bloc,Compos),
member(C,Compos), !, unionBlocs(AxCmps,Bloc,BlocU).
unionBlocs([axCmp(Axe,C)|AxCmps],Bloc,[axCmp(Axe,C)|AxCmpsU]) :-
unionBlocs(AxCmps,Bloc,AxCmpsU).
unionBlocs([],Bloc,Bloc).

estCategorie(meuble,Idc) :- composant(meuble,Idc,_,_,_).
estCategorie(electro,Idc) :- composant(electro,Idc,_,_,_).

laX(Idc,X) :- laPosition(Idc,position(X,_,_)).

laY(Idc,Y) :- laPosition(Idc,position(_,Y,_)).

laZ(Idc,Z) :- laPosition(Idc,position(_,_,Z)).

laHauteur(Idc,H) :- laDimension(Idc,(H,_,_)).

laLargeur(Idc,L) :- laDimension(Idc,(_,L,_)).

laProfondeur(Idc,P) :- laDimension(Idc,(_,_,P)).

% leComposant(IdAgencement,IdComposant)
% unifie IdComposant à un identifaint de composant de l'agencement d'identifiant
% IdAgencement
leComposant(Ida,Idc) :-
agencement(Ida,IdCs),
member(Idc,IdCs).

% estVoisin(Dir,Axe,C1,C2)
% retourne vrai si C1 est un voisin dans la direction Dir de C2 sur l'axe Axe
estVoisin(droite,Axe,C1,C2) :- estVoisin(gauche,Axe,C2,C1).
estVoisin(gauche,Axe,C1,C2) :-
laX(C1,X1),
laX(C2,X2),
laY(C1,Y1),
laY(C2,Y2),
laZ(C1,Z1),
laZ(C2,Z2),
laLargeur(C2,L2),
(Axe = x -> Xl2 is X2 + L2, Xl2 == X1, Y1 == Y2, Z1 == Z2;
    Yl2 is Y2 - L2, Yl2 == Y1, X1 == X2, Z1 == Z2).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% lesVoisins(Dir,Axe,Ag,Compo,Voisins)
% Dir est une direction
% Axe est un axe
% Ag est un agencement
% Compo est un composant
% Voisins s'unifie à la liste des voisins de Compo sur l'axe Axe et dans la
% direction Dir. Ici, la liste des voisins comprend tous les blocs liés dans
% la direction donnée. Par exemple, si b1 est un voisin de b2 et b2 est un
% voisin de Compo, alors b1 et b2 seront dans la liste Voisins.
%
%On tente donc de trouver un voisin de Compo à partir de la liste des composants
%de l'agencement donné. Si on trouve un voisin, on cherche alors les voisins de
%ce composant dans la liste originale, dans la même direction donnée (sinon, on
%pourrait obtenir une récurision sans fin). On ajoute ce résultat à la liste Voisins,
%et on continue ensuite à traiter le reste de la liste
%
%Exemple d'utilisation:
%| ?- lesVoisins(droite,y,ag,b3,Bs).
%Bs = [b1] ?
%yes
%| ?- lesVoisins(gauche,x,ag,b4,Bs).
%Bs = [b2,b1] ?
%yes
lesVoisins(Dir,Axe,Ag,Compo,Voisins) :- agencement(Ag,L),
                                        lesVoisins(Dir,Axe,Ag,L,Compo, Voisins),!.

lesVoisins(_,_,_,[],_,[]).
lesVoisins(Dir,Axe,Ag,[X|Xs],Compo,[X|Ys]) :- estVoisin(Dir,Axe,Compo,X),
                                lesVoisins(Dir,Axe,Ag,X,Voisins),append(Voisins,Zs,Ys),
                                lesVoisins(Dir,Axe,Ag,Xs,Compo,Zs).
lesVoisins(Dir,Axe,Ag,[X|Xs],Compo,Ys) :- \+ estVoisin(Dir,Axe,Compo,X),
                                                lesVoisins(Dir,Axe,Ag,Xs,Compo,Ys).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% leBloc(Ax,Ag,Compo,Bloc)
% Axe est un axe
% Ag est un agencement
% Bloc s'unfie au bloc sur l'axe Axe qui contient le composant Compo
leBloc(Axe,Ag,Compo,Bloc) :-
lesVoisins(gauche,Axe,Ag,Compo,B1),
lesVoisins(droite,Axe,Ag,Compo,B2),
append(B1,[Compo|B2],B12),
faireBloc(Axe,B12,Bloc).

% sontDansMemeBloc(Ag,C1,C2)
% Axe est un axe
% C1 et C2 sont des composants
% le prédicat est vrai si et seulement si C1 et C2 font partie d'un même bloc
sontDansMemeBloc(Ag,C1,C2) :-
```

```
leBloc(x,Ag,C1,Bx),
leBloc(y,Ag,C1,By),
lesComposantsDuBloc(Bx,Cx),
lesComposantsDuBloc(By,Cy),
(member(C2,Cx);member(C2,Cy)).

% estDansBlocs(Blocs,C)
% retourne vrai si le composant C se trouve dans un des blocs de la liste de blocs Blocs
estDansBlocs([Bloc|_],C) :-
lesComposantsDuBloc(Bloc,Compos),
member(C,Compos),!.
estDansBlocs([_|Blocs],C) :-
estDansBlocs(Blocs,C).

% lesBlocs(Axe,Ag,Compos,Blocs)
% Axe est x ou y
% Ag est un agencement
% Compos est une liste de composants
% unifie Blocs à la liste des blocs sur l'axe formés avec chacun des composants de compos
lesBlocs(_,_,[],[]).
lesBlocs(Axe,Ag,[Compo|Compos],Blocs) :-
lesBlocs(Axe,Ag,Compos,Blocps),
(estDansBlocs(Blocps,Compo) -> Blocs = Blocps;
    leBloc(Axe,Ag,Compo,Bloc),
    Blocs = [Bloc|Blocps]).

% lesBlocs(Axe,Ag,Blocs)
% Axe est x ou y
% Ag est un agencement
% unifie Blocs à la liste des blocs sur l'axe
lesBlocs(Axe,Ag,Blocs) :-
agencement(Ag,Compos),
lesBlocs(Axe,Ag,Compos,Blocs).

% lesBlocs(Ag,Blocs)
% Ag est un agencement
% unifie Blocs à la liste des blocs de l'agencement
% Cetet liste est obtenue en prenant les listes correspondants à chaque axe
% et en éliminant les blocs d'un axe qui sont inclus strictement dans les blocs de l'autre axe
lesBlocs(Ag,Blocs) :-
lesBlocs(x,Ag,BlocsX),
lesBlocs(y,Ag,BlocsY),
elimineBlocs(BlocsY,BlocsX,BlocsYp),
elimineBlocs(BlocsX,BlocsY,BlocsXp),
append(BlocsXp,BlocsYp,Blocs).

elimineBlocs([BlocY|BlocsY],BlocsX,Blocs) :-
elimineBloc(BlocY,BlocsX),!,
elimineBlocs(BlocsY,BlocsX,Blocs).
elimineBlocs([BlocY|BlocsY],BlocsX,[BlocY|Blocs]) :-
```

```
elimineBlocs(BlocsY,BlocsX,Blocs).
elimineBlocs([],_,[]).

elimineBloc(BlocY,[BlocX|_]) :-
estStrictementInclus(BlocY,BlocX),!.
elimineBloc(BlocY,[_|BlocsX]) :-
elimineBloc(BlocY,BlocsX).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%estStrictementInclus(BlocY,Blocx):
%BlocX et BlocY sont des listes de blocs
%Le prédicat vérifie si tous les composants de BlocX sont inclus dans
%les blocs de BlocX, tel que BlocX contient au moins un autre composant
%qui n'est pas contenu dans BlocY (selon la définition ensembliste d'un
%ensemble strictement inclus dans un autre ensemble).
%On obtient donc la liste des composants des deux blocs, et ensuite on compare
%on vérifie si la liste des composants de BlocX est une sous-liste de la liste
%de BlocY, et aussi si la liste des composants de BlocY n'est pas une sous-liste
%de BlocX.
%
%Je définie ici le prédicat sousListe pour vérifier si une liste est une
%sous-liste d'une autre liste. On pourrait aussi utiliser subset, défini
%par prolog, mais je préfère le faire moi-même.
%
%Exemple d'utilisation:
%| ?- bloc(1,B1),bloc(7,B7),estStrictementInclus(B1,B7).
%B1 = [axCmp(x,b1),axCmp(x,b2),axCmp(x,b4)],
%B7 = [axCmp(x,b1),axCmp(x,b2),axCmp(y,b3),axCmp(x,b4)] ?
%yes
estStrictementInclus(By,Bx) :- lesComposantsDuBloc(By, Cys),
                               lesComposantsDuBloc(Bx, Cxs),
                               sousListe(Cys,Cxs),\+sousListe(Cxs,Cys).

%sousListe(L1,L2):
%L1 et L2 sont deux listes, de n'importe quel type d'éléments
%sousListe vérifie si la liste L1 est une sous-liste de L2.
%Exemples d'utilisation:
%| ?- sousListe([1,2,3],[1,2,3,4,5,6]).
%yes
%| ?- sousListe([1,2,3],[1,3,4,5,6]).
%no
sousListe([],_).
sousListe([X|Xs],Liste) :- member(X,Liste),sousListe(Xs,Liste).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%lesComposantsDuType(IdAgencement, Ty, Compos):
%idAgencement: un agencement, fourni selon son identificateur
%Ty: le type de composant recherché
%Compos: s'unifie à la liste des composants de type Ty contenus dans l'agencement
%idAgencement. La liste Compos doit contenir tous les éléments qui s'unifient, et
```

```
%non une sous-liste. Je définie un deuxième prédicat pour vérifier la liste des
%composants, alors que lesComposantsDuType obtient la liste des composants
%d'un agencement qui est utilisé par le second prédicat.
%
%On obtient tout d'abord la liste des composants de l'agencement, et ensuite on
%vérifie chaque agencement. Si l'agencement est du type recherché, on l'insère
%dans Compos, sinon, on passe au prochain élément de la liste.
%Exemple d'utilisation:
%| ?- lesComposantsDuType(ag,bas,Bs).
%Bs = [b3,b4] ?
%yes
lesComposantsDuType(Id,Ty,Compos) :- agencement(Id,Ags),
                                     lesComposantsDuType2(Ty,Ags,Compos).


%lesComposantsDuType2(Ty,Composants,Compos).
%Ty est le type de composant recherché.
%Composants est la liste comprenants tous les composants.
%Compos s'unifie à la liste des composants du type Ty.
%Compos doit contenir tous les éléments du type recherche, et non seulement
%une sous-liste.
%exemple d'utilisation:
%| ?- lesComposantsDuType2(haut,[b1,b2,b3,b4,b5,b6,h1,h2],Compos).
%Compos = [b5,b6,h1,h2] ?
%yes
lesComposantsDuType2(Ty,[X|Xs],[X|Q]) :- composant(_,X,_,_,Ty),!,
                                          lesComposantsDuType2(Ty,Xs,Q).
lesComposantsDuType2(Ty,[_|Xs], Q) :- !,lesComposantsDuType2(Ty,Xs,Q).
lesComposantsDuType2(_,[],[]).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% possedeTypes(Ag,Types)
% Est vrai si et seulement si l'agencement Ag possède tous les types de composants
% de la liste Types
possedeTypes(Ag,[Ty|Types]) :-
lesComposantsDuType(Ty,Ag,[_|_]),
possedeTypes(Ag,Types).
possedeTypes(_,[]).



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% lesIds(Blocs,Css)
% Blocs: Liste de blocs
% Css: Liste de liste d'identificateurs de composants.
% lesIds Unifient Css avec la liste de listes ce composants des blocs de la
% liste Blocs.Il est possible aussi d'unifier la liste Blocs avec les
% blocs associés à la liste de listes de composants, toutefois l'information
```

```
% dans les blocs sera incomplète, on ne peut obtenir la direction de l'axe.
% Exemple d'utilisation:
%| ?- bloc(1,Bloc1),bloc(2,Bloc2),lesIds([Bloc1,Bloc2],Ids).
%Ids = [[b1,b2,b4],[b5,b6]],
%Bloc1 = [axCmp(x,b1),axCmp(x,b2),axCmp(x,b4)],
%Bloc2 = [axCmp(x,b5),axCmp(x,b6)] ?
%yes

lesIds([X|Xs], [C|Css]) :- lesComposantsDuBloc(X, C),lesIds(Xs,Css).
lesIds([],[]).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Exemple d'exécution des tests de Main.pl sur rayon1:

```
| ?- consult('Main').
% consulting /home/gk491589/INF2160/TP2/Main.pl...
%  consulting /home/gk491589/INF2160/TP2/Agencement.pl...
%  consulted /home/gk491589/INF2160/TP2/Agencement.pl in module user, 0 msec 27152 bytes

TESTS POUR -- lesIds(...)
=========================================================================
#1. lesIds: bloc vide: OK
#2. lesIds: liste d'un bloc avec un composant: OK
#3. lesIds: liste d'un bloc avec plusieurs composants: OK
#4. lesIds: liste de plusieurs blocs: OK
NOTE POUR CETTE PARTIE: 3.0/3

TESTS POUR -- lesComposantsDuType(...)
=========================================================================
#5. lesComposantsDuType : agencement vide, rien: OK
#6. lesComposantsDuType : agencement un composant, rien: OK
#7. lesComposantsDuType : agencement un composant, un composant: OK
#8. lesComposantsDuType : agencement 2 composants, un composant: OK
#9. lesComposantsDuType : agencement plusieurs composants, un composant: OK
#10. lesComposantsDuType : agencement plusieurs composants, plusieurs composants: OK
NOTE POUR CETTE PARTIE: 5.0/5

TESTS POUR -- estStrictementInclus(...)
=========================================================================
#11. estStrictementInclus : bloc vide: OK
#12.estStrictementInclus : bloc 6 avec 1: OK
#13. estStrictementInclus : bloc1 avec 4, blocs egaux: OK
#14. estStrictementInclus : bloc 1 avec bloc 7: OK
#15. estStrictementInclus : bloc 0 avec 1: OK
NOTE POUR CETTE PARTIE: 4.0/4

TESTS POUR -- lesVoisins(...)
```

```
================================================================
#16. lesVoisins : agencement vide: OK
#17. lesVoisins : pas voisins: OK
#18. lesVoisins : un voisin gauche sur x: OK
#19. lesVoisins : un voisin droite sur y: OK
#20. lesVoisins : desvoisins gauche sur x: OK
NOTE POUR CETTE PARTIE: 4.0/4

NOTE FINALE: 16.0/16
% consulted /home/gk491589/INF2160/TP2/Main.pl in module user, 10 msec 52000 bytes
yes
```

Voici la définition des autres tests ajoutés (testés sur swipl).

1. lesIds: Obtenir une liste de blocs à partir d'une liste de liste de composants.

```
?- lesIds(Bs,[[b1,b2,b3],[h1,h2]]).
Bs = [[axCmp(_G339, b1), axCmp(_G345, b2), axCmp(_G351, b3)], [axCmp(_G360, h1), axCmp(_G366, h2)
|    lesIds(Bs,[]).
Bs = [].
```

Ici, on peut voir que la direction des blocs est inconnue, car la fonction ne peut déterminer cette information. Pour la liste vide, on obtient une liste de blocs vide, ce qui est le résultat attendu.

---

2. lesComposantsDuBloc: Obtenir le type d'une liste de composants d'un agencement.

```
?- lesComposantsDuType(ag, Ty,[b5,b6,h1,h2]).
Ty = haut.
?- lesComposantsDuType(ag, Ty,[b1,b5,b6,h1,h2]).
false.
?- lesComposantsDuType(ag, Ty,[b6,h1,h2]).
Ty = haut.
```

Pour l'exemple 1 et 3, on obtient le bon résultat, le type de Ty est celui commun à tous les composants de la liste fournie. Toutefois, le résultat de l'exemple 3 ne vérifie pas que la liste des composants contient tous les composants d'un type de l'agencement.

Pour l'exemple 2, on obtient un bon résultat, car l'interpréteur détecte que les composants de la liste fournie ne sont pas tous du même type.

---

3. estStrictementInclus: Vérifier si un bloc est strictement inclus dans lui-même.

```
|    bloc(7,B7),estStrictementInclus(B7,B7).
false.
```

Tel qu'attendu, d'après la définition d'un ensemble strictement inclus dans un autre ensemble, un bloc n'est pas strictement inclus dans lui-même.

J'ai aussi tenté de vérifier si, à partir de la définition d'un bloc, prolog peut créer soit un bloc qui est strictement inclus dans le bloc donné (donc estStrictementInclus(Bs,bloc)), ou encore le contraire (estStrictementInclus(bloc, Bs)). Dans ces situations, prolog ne peut déterminer un résultat.

---

4. lesVoisins: Vérifier si on peut unifier Compo à partir d'une définition de Voisins

```
?- lesVoisins(gauche,x,ag,Compo,[b1]).
Compo = b2.
?- lesVoisins(gauche,x,ag,Compo,[b2,b1]).
false.
?- lesVoisins(gauche,x,ag,Compo,[b2]).
false.
```

Ici, on obtient le bon résultat pour le premier test, mais pas pour le troisième. Le prédicat peut donc être utilisé pour unifier Voisins à partir de Compo, mais pas pour unifier Compo à partir de Voisins.