Université Paris 7 - Master 1 Informatique - Programmation logique par contraintes

Examen du 6 janvier 2012 - Durée : 2 heures

Informations : Tous les documents sont autorisés. Le barème est donné à titre indicatif et peut être modifié.

Exercice 1 (**4 points**)
Considérez la contrainte $X < Y \ \wedge \ Y < Z \ \wedge \ Z \leq X$ avec domaines $D(X) = D(Y) = D(Z) = [1...1000]$.

- Est-ce que cette contrainte a une solution ?
  Correction : non
- Rendez la contrainte borne-consistante en considérant une par une les contraintes simples. Faites uniquement les premiers 6 pas. Combien de pas sont nécessaires avant de s'arrêter ?
  Correction : On considère d'abord $X < Y$: $D(X)$ devient $[1...999]$, car pour $X = 1000$ il n'y a pas de valeur de Y qui rend $X < Y$ vrai. $D(Y)$ devient $[2...1000]$. On considère ensuite $Y < Z$: $D(Y)$ devient $[2...999]$ et $D(Z)$ devient $[3...1000]$. On considère ensuite $Z \leq X$. $D(Z)$ devient $[3...999]$ et $D(X) = [3...999]$. On reconsidère $X < Y$: $D(X) = [3...998]$ et $D(Y) = [4...999]$. On reconsidère $Y < Z$: $D(Y) = [4...998]$ et $D(Z) = [5...999]$. On reconsidère $Z \leq X$: $D(Z) = [5...998]$ et $D(X) = [5...998]$. On remarque qu'à chaque fois qu'on considère les trois contraintes les bornes supérieures diminuent de 1 et les bornes inférieures augmentent de 2. Donc après $331*3$ d'autres pas on obtient $D(X) = [666...666]$, $D(Y) = [665...666]$ et $D(Z) = [666...666]$. Un pas après le domaine de X devient vide et on s'arrête. En tout : 1000 pas.
- Si on demande à GPROLOG `?- X #< Y, Y #< Z, Z #=< X.` la réponse est donnée après environ 16 secondes ! Pourquoi ?
  Correction : Puisque les domaines ne sont pas spécifiés, par défaut GPROLOG prend les domaines $D(X) = D(Y) = D(Z) = [0...268435455]$ où 268435455 est `fd_max_integer`. Ensuite il fait environ 268435455 pas dans l'algorithme de bornes-consistance (pour se rendre compte que la contrainte n'est pas satisfaisable) ce qui prend beaucoup de temps.
- Dans YAP on fait d'abord `?- use_module(library(clpr)).` Ensuite on demande `?- {X < Y, Y < Z, Z =< X}.` La réponse est donnée quasiment instantanément. Pourquoi ?
  Correction : clpr travaille sur les réels où le solveur de contraintes se rend compte très rapidement qu'il n'y a pas de solution.
  Faute courant : Ne pas considérer la borne inférieur et la borne supérieur.

Exercice 2 (**4 points**)
Minimiser Z par rapport à $X \geq 0, Y \geq 0, Z \geq 0$ et

$$\begin{aligned} 2*X - 2*Y + 3*Z &\leq 1 \\ 2*X + 2*Y + 3*Z &= 2 \\ 5*X - 6*Y &= 3 \end{aligned}$$

Indication : Si vous pivotez une variable, il est conseillé de commencer avec X.

Correction : Le problème n'est pas en forme simplex. On le modifie : Minimiser Z par rapport à $X \geq 0, Y \geq 0, Z \geq 0, S \geq 0$ et

$$2*X - 2*Y + 3*Z + S = 1$$

$$\begin{aligned} 2*X+2*Y+3*Z &= 2 \\ 5*X-6*Y &= 3 \end{aligned}$$

Il n'est pas en forme simplex de base. On essaye de trouver une solution de base. On prend trois variables artificielles A, B et C (une pour chaque équation et on essaie de minimiser $A+B+C$ (où on remplace chaque variable par sa partie droite). On obtient : Minimiser $6-9*X+6*Y-6*Z-S$ par rapport à $X \geq 0, Y \geq 0, Z \geq 0, S \geq 0, A \geq 0, B \geq 0, C \geq 0$ et

$$\begin{aligned} A &= 1-2*X+2*Y-3*Z-S \\ B &= 2-2*X-2*Y-3*Z \\ C &= 3-5*X+6*Y \end{aligned}$$

On pivote X et on choisit la première équation. On obtient :
Minimiser $3/2+(9/2)*A-3*Y+(15/2)*Z+(7/2)*S$ par rapport à $X \geq 0, Y \geq 0, Z \geq 0, S \geq 0, A \geq 0, B \geq 0, C \geq 0$ et

$$\begin{aligned} X &= 1/2-(1/2)*A+Y-(3/2)*Z-(1/2)*S \\ B &= 1+A-4*Y+S \\ C &= 1/2+Y+(5/2)*A+(15/2)*Z+(5/2)*S \end{aligned}$$

On pivote Y dans la deuxième équation :
Minimiser $3/4+(15/4)*A+(3/4)*B+(15/2)*Z+(11/4)*S$ par rapport à $X \geq 0, Y \geq 0, Z \geq 0, S \geq 0, A \geq 0, B \geq 0, C \geq 0$ et

$$\begin{aligned} X &= 3/4-(1/4)*A-(1/4)*B-(3/2)*Z-(1/4)*S \\ Y &= 1/4+(1/4)*A-(1/4)*B+(1/4)*S \\ C &= 3/4+(11/4)*A-(1/4)*B+(15/2)*Z+(9/4)*S \end{aligned}$$

On n'atteint pas 0. Donc la contrainte d'origine n'a pas de solution.
Fautes courantes : Mettre la variable additionnelle à la mauvaise place de l'inéquation. Ne pas savoir comment obtenir une forme de base. Remplacer Z par $Z^+ - Z^-$.

Exercice 3 (**4 points**)
Un nutritionniste veut mettre en place un régime alimentaire à la fois équilibré et le moins cher possible. On suppose que les seuls aliments disponibles sont les suivants :

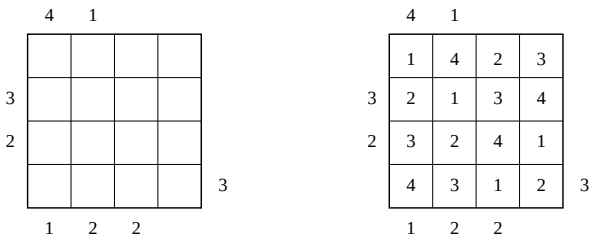
aliment	taille d'une portion	énergie (kcal)	protéine (gramme)	calcium (mg)	prix (Euros/portions)	limite (portions/jour)
flocons d'avoine	28g	110	4	2	0.3	4
lait	221ml	160	8	285	0.9	8
poulet	100g	205	32	12	2.4	3
oeufs	2	160	13	54	1.3	2
tarte au citron	170g	420	4	22	2	2
boeuf-carotte	260g	260	14	80	1.9	2

Le régime alimentaire doit contenir au moins 2000 kcal d'énergie, 55g de protéine et 800 mg de calcium. Une certaine variété est imposé (car qui voudra manger 10 portions de boeuf-carotte par jour ?) et pour chaque aliment un nombre limite de portions par jour est donc fixé (le nombre de portions/jour peut être une fraction). Le nutritionniste se pose la question : Quel est le régime alimentaire satisfaisant qui est le **moins** cher ?

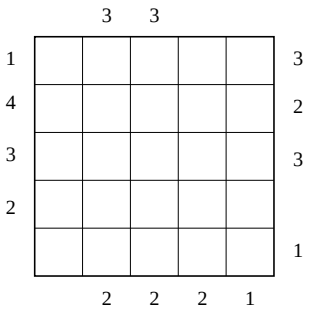
– Aidez le nutritionniste en **modélisant** le problème comme un CSP. Il vous est demandé de **modéliser** et non pas de **résoudre** le problème.
Correction : C'est un problème d'optimisation sur les réels. On considère 6 **variables, une pour chaque aliment,** $x_1, x_2, x_3, x_4, x_5, x_6$ **(leur valeur correspond au nombre de portions pas jour de cet aliment) et le problème est :**
Minimiser $0.3 * x_1 + 0.9 * x_2 + 2.4 * x_3 + 1.3 * x_4 + 2 * x_5 + 1.9 * x_6$ **par rapport à** $x_1 \geq 0 \wedge x_2 \geq 0 \wedge x_3 \geq 0 \wedge x_4 \geq 0 \wedge x_5 \geq 0 \wedge x_6 \geq 0 \wedge x_1 \leq 4 \wedge x_2 \leq 8 \wedge x_3 \leq 3 \wedge x_4 \leq 2 \wedge x_5 \leq 2 \wedge x_6 \leq 2 \wedge 110 * x_1 + 160 * x_2 + 205 * x_3 + 160 * x_4 + 420 * x_5 + 260 * x_6 \geq 2000 \wedge 4 * x_1 + 8 * x_2 + 32 * x_3 + 13 * x_4 + 4 * x_5 + 14 * x_6 \geq 55 \wedge 2 * x_1 + 285 * x_2 + 12 * x_3 + 54 * x_4 + 22 * x_5 + 80 * x_6 \geq 800$

Fautes courantes : Écrire un programme GPROLOG. Cela n'était pas demandé et en plus ça ne marche pas, car on a besoin ici de variables réelles. Oublier les contraintes qui disent que chaque variable doit être supérieure ou égale à 0**.**

Exercice 4 (**8 points**)
Des gratte-ciel sont construits sur une grille NxN. Chaque gratte-ciel a un nombre d'étages correspondant à sa taille (de 1 à N). Dans chaque colonne et chaque ligne chaque taille apparaît exactement une fois. Les nombres sur les bords de la grille indiquent combien de gratte-ciel peuvent être vus en regardant de ce point vers la ligne ou la colonne correspondante. Un gratte-ciel est visible, si **tous** les gratte-ciel devant lui sont plus petits. Voici l'exemple d'une grille et de sa solution :



Par exemple, dans la deuxième ligne du bas en regardant de la gauche on peut voir 2 gratte-ciel : celui d'hauteur 3 et celui d'hauteur 4. Voici un deuxième exemple d'une grille (5x5) :



– Donnez un programme en GPROLOG qui résout le deuxième exemple. Il vous est demandé d'écrire un programme pas de résoudre vous-même le problème.
Indications : Vous pouvez utiliser le prédicat prédéfini `fd_cardinality(+List, ?Count)` où `List` est une liste de contraintes et `Count` le nombre de contraintes de `List` satisfaites. Dans les contraintes, vous pouvez utiliser la conjonction `#/\`. Par exemple, `fd_cardinality([3 #< 4 #/\ 4 #< 5, 3 #< 2],1)` est vrai.
Corrigé :

```
% Une contrainte de visibilité
```

```
contrainte([A1,A2,A3,A4,A5],C) :-
   fd_cardinality([A5 #> A4 #/\ A5 #> A3  #/\ A5 #> A2 #/\ A5 #> A1,
                   A4 #> A3 #/\ A4 #> A2  #/\ A4 #> A1,
                   A3 #> A2 #/\ A3 #> A1, A2 #> A1],C).

gratteciel(L) :-
   L = [Line1,Line2,Line3,Line4,Line5],
   Line1 = [X11,X12,X13,X14,X15],
   Line2 = [X21,X22,X23,X24,X25],
   Line3 = [X31,X32,X33,X34,X35],
   Line4 = [X41,X42,X43,X44,X45],
   Line5 = [X51,X52,X53,X54,X55],
   fd_domain(Line1,1,5),fd_domain(Line2,1,5),
   fd_domain(Line3,1,5),fd_domain(Line4,1,5),
   fd_domain(Line5,1,5),
   fd_all_different(Line1),fd_all_different(Line2),
   fd_all_different(Line3),fd_all_different(Line4),
   fd_all_different(Line5),
   fd_all_different([X11,X21,X31,X41,X51]),
   fd_all_different([X12,X22,X32,X42,X52]),
   fd_all_different([X13,X23,X33,X43,X53]),
   fd_all_different([X14,X24,X34,X44,X54]),
   fd_all_different([X15,X25,X35,X45,X55]),
% Le premier gratte-ciel est toujours visible. Il faut donc enlever 1
% de chaque nombre de gratte-ciel visible.
   contrainte(Line1,0), contrainte(Line2,3), contrainte(Line3,2),
   contrainte(Line4,1),
   contrainte([X12,X22,X32,X42,X52],2),
   contrainte([X13,X23,X33,X43,X53],2),
   contrainte([X15,X14,X13,X12,X11],2),
   contrainte([X25,X24,X23,X22,X21],1),
   contrainte([X35,X34,X33,X32,X31],2),
   contrainte([X55,X54,X53,X52,X51],0),
   contrainte([X52,X42,X32,X22,X12],1),
   contrainte([X53,X43,X33,X23,X13],1),
   contrainte([X54,X44,X34,X24,X14],1),
   contrainte([X55,X45,X35,X25,X15],0),
   fd_labeling(Line1),fd_labeling(Line2),fd_labeling(Line3),
   fd_labeling(Line4),fd_labeling(Line5).
```

– Donnez un programme en GPROLOG qui résout un problème de gratte-ciel quelconque. Pour cela, donnez d'abord une description de la représentation du problème (taille de la grille, etc.).
Indication : Vous pouvez supposer qu'il existe un prédicat `transpose(+L,?T)` qui étant donné une liste de listes `L` représentant une grille renvoie dans `T` une liste de listes avec colonnes et lignes inversés. Par exemple, `transpose([[1,2,3],[4,5,6],[7,8,9]],T)` renvoie `T = [[1,4,7],[2,5,8],[3,6,9]]`. Vous pouvez également utiliser `reverse(+L,?R)` qui renverse une liste `L`. Par exemple, `reverse([1,2,5],R)` donne `R=[5,2,1]`.
  – Commencez par écrire un prédicat qui génère une grille NxN de variables de la forme `[[...],[...],...,[...]]`

Corrigé :
On utilise un prédicat `gratteciel(?L,+N,+LineLeft,+LineRight,+ColumnUp,+ColumnDown)` qui prend comme entrée la dimension `N` et les contraintes de visibilité pour les lignes à gauche, à droite, les colonnes en haut et en bas et renvoie dans `L` une solution.

```
transpose([], []).
transpose([X|L], L1) :-
    transpose(X, [X|L], L1).

transpose([], _, []).
transpose([_|Rest], L, [T|T1]) :-
        split(L, T, Rest1),
        transpose(Rest, Rest1, T1).

split([], [], []).
split([[F|L]|Rest], [F|L1], [L|L2]) :-
     split(Rest, L1, L2).


% génère une grille de taille NxN de variables de
% la forme [[...],[...],...,[...]]
makegrid(L,N) :- makegrid(L,N,N).
makegrid([],_,0) :- !.
makegrid([X|L],N,N1) :- length(X,N),N2 is N1-1,makegrid(L,N,N2).

alldifferentlines([]).
alldifferentlines([L|Rest]) :- fd_all_different(L),alldifferentlines(Rest).

alldifferentcolums(L) :- transpose(L,L1),alldifferentlines(L1).

alldifferent(L) :- alldifferentlines(L),alldifferentcolums(L).

% génère les domaines des variables
domain([],_,_).
domain([X|L],A,B) :- fd_domain(X,A,B),domain(L,A,B).

visible1(_,[],1). % 1 is equal to true
visible1(X,[Y1|L],((X #> Y1) #/\ C)) :- visible1(X,L,C).

visible([],[]).
visible([X|L],[C|C1]) :- visible1(X,L,C),visible(L,C1).

% Constrainte sur une ligne.
% si 0, alors pas de contraintes
% sinon, on génère les contraintes pour le nombre de gratte ciel visible.
line(_,0) :- !.
line(L,X) :- reverse(L,L1),visible(L1,C),fd_cardinality(C,X).

% Contraintes sur les lignes.
% On traite la contrainte à gauche et ensuite celle de droite (symétrique
% en renversant la liste
```

```
lines([],[],[]).
lines([L|Rest],[Left1|Left],[Right1|Right]) :-
  line(L,Left1),
  reverse(L,L1),
  line(L1,Right1),
  lines(Rest,Left,Right).

% Les contraintes sur les colonnes sont symétriques. Il suffit d'utiliser
% le prédicat pour les lignes.

columns(L,ColumnUp,ColumnDown) :- transpose(L,L1),lines(L1,ColumnUp,ColumnDown).

labeling([]).
labeling([X|L]) :- fd_labeling(X),labeling(L).

% LineLeft,LineRight, ColumnUp, ColumnDown sont quatres listes d'exactement
% N éléments qui contiennent à chaque position le nombre de gratte-ciel
% visible ou 0 si ce nombre n'est pas contraint.

gratteciel(L,N,LineLeft,LineRight,ColumnUp,ColumnDown) :-
   makegrid(L,N),domain(L,1,N),
   alldifferent(L),lines(L,LineLeft,LineRight),columns(L,ColumnUp,ColumnDown),
   labeling(L).


% gratteciel(L,4,[0,3,2,0],[0,0,0,3],[4,1,0,0],[1,2,2,0]).
% gratteciel(L,5,[1,4,3,2,0],[3,2,3,0,1],[0,3,3,0,0],[0,2,2,2,1]).
```