Bien dans l'ensemble, revoir le dernier exercice. Attention, à bien utiliser le labeling qu'une fois que toutes les contraintes sont posées !

Très bon compte-rendu.

Rapport Travaux Pratiques :
Programmation par Contraintes
- TP 1 :

Découverte de la bibliothèque de contraintes à domaines finis

Nicolas Desfeux
Aurélien Texier

23 février 2011

Table des matières

1 De Prolog à Prolog+ic

1.1 Des contraintes sur les arbres

Question 1.1 L'objectif est ici de définir un prédicat permettant d'effectuer un choix parmi des ensembles. Voici le code de nos prédicats, ainsi que les tests associés.

Listing 1 – "Prédicat choixCouleur"

```
1
2  couleurVoiture(rouge).
3  couleurVoiture(vert(clair)).
4  couleurVoiture(gris).
5  couleurVoiture(blanc).
6
7  couleurBateau(vert(_)).
8  couleurBateau(noir).
9  couleurBateau(blanc).
10
11
12 choixCouleur(CouleurBateau,CouleurVoiture) :-  couleurVoiture(CouleurVoiture),
13                                                couleurBateau(CouleurBateau),
14                                                CouleurVoiture=CouleurBateau.
15
16 /* Tests
17 [eclipse 78]: choixCouleur(CouleurBateau,CouleurVoiture).
18
19 CouleurBateau = vert(clair)
20 CouleurVoiture = vert(clair)
21 Yes (0.00s cpu, solution 1, maybe more) ? ;
22
23 CouleurBateau = blanc
24 CouleurVoiture = blanc
25 Yes (0.00s cpu, solution 2)
26 */
```

Bien

Question 1.2 Prolog parcourt les branches de l'arbre des possibilités et il coupe les branches qui ne s'unifient pas avec les contraintes.
Il s'arrête quand il a parcouru toutes les possibilités, et donc il génère toutes les solutions possibles, toutes celles qui remplissent les contraintes.

Quelles sont les variables et les domaines que prolog sait manipuler ? Qual est le rôle de l'unification ?

1.2 Prolog ne comprend pas les Maths (mais il a une bonnne calculatrice)

Question 1.3 Voici le code du prédicat isBetween :

Listing 2 – "Prédicat isBetween"

```
1 isBetween(Var,Min,_Max) :-  Var=Min.
2 isBetween(Var,Min,Max) :-   \=(Min,Max),
3                             is(M,Min+1),
4                             isBetween(Var,M,Max).
```

Tests ??

Question 1.4 Voici le code du prédicat commande, et les tests associés :

Listing 3 – "Prédicat commande"

```
1  nbMaxR(10000).
2  nbMinR(5000).
3  nbMinC(9000).
4  nbMaxC(20000).
5
6  commande(NbResistance,NbCondensateur) :-  nbMaxR(NbMaxR),nbMinR(NbMinR)
       ,nbMaxC(NbMaxC),nbMinC(NbMinC),
7                                            isBetween(NbResistance,NbMinR
                                                 ,NbMaxR),
8                                            isBetween(NbCondensateur,
                                                 NbMinC,NbMaxC),
9                                            NbResistance>=NbCondensateur.
10
11 /* Tests
12 [eclipse 79]: commande(NbResistance,NbCondensateur).
13
14 NbResistance = 9000
15 NbCondensateur = 9000
16 Yes (7.73s cpu, solution 1, maybe more) ? ;
17
18 NbResistance = 9001
19 NbCondensateur = 9000
20 Yes (7.74s cpu, solution 2, maybe more) ? ;
21
22 NbResistance = 9001
23 NbCondensateur = 9001
24 Yes (7.74s cpu, solution 3, maybe more) ?
25 */
```

Bien

Estimation du nombre de solutions ?

Le test de commande permet de valider le fonctionnement de *isBetween/2*.

Question 1.5 Le temps de réponse n'est pas négligeable puisqu'il est de 7.73 secondes.
En utilisant la trace de l'exécution, on constate que l'on est ici dans un contexte de "Generate and Test", qui est dans ce cas beaucoup plus coûteux en temps.
Il génère les solutions qui sont dans les intervalles données (il y en a beaucoup !), puis il teste celles qui remplissent la condition >=.
Comme on peut le constater dans le dessin de l'arbre de recherche Porlog ci-joint (figure 1 page 4), Prolog rencontre dans son arbre de recherche beaucoup d'échecs puisqu'il génère tout, et c'est cela la cause de la perte de beaucoup de temps d'exécution.

OK

Question 1.6 On réitère l'essai mais en mettant le prédicat >= avant le isBetween.
Nous sommes ici dans un cas de figure de "Constraints and Generate" On voit alors ici que Prolog ne peut pas trouver de solution car pour remplir la condition sur le >=, il y a une infinité de solutions puisqu'il ne sait pas encore dans quel intervalle travailler. Donc il nous répond qu'il y a une faute d'instanciation. Dans le cas d'avant, le isBetween permettait de lui donner un nombre fini de solutions pour les deux variables, et après il pouvait alors trouver les solutions

Pas très rigoureux dans la forme...

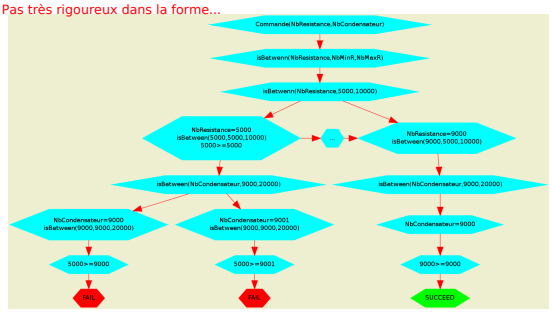

FIGURE 1 – Arbre Prolog 1

qui remplissaient le >=. Nous sommes pour le commandeBis dans un cas de figure de "Generate and Test", puisqu'il génère les solutions sans regarder ce qu'il a après comme condition.

-> Prolog ne sait pas "raisonner" avec des variables mathématiques, il peut juste faire des calculs, comme avec une calculatrice

1.3 Le solveur ic à la rescousse

Question 1.7 Nous implémentons ici le prédicat *commande2/2* en remplaçant isBetween par des commandes ic.

Listing 4 – "prédicat Commande"

```
1 commande2(NbResistance,NbCondensateur) :-  nbMaxR(NbMaxR),nbMinR(NbMinR)
      ,nbMaxC(NbMaxC),nbMinC(NbMinC),
2                                            NbResistance #:: NbMinR..
                                                 NbMaxR,
3                                            NbCondensateur #:: NbMinC..
                                                 NbMaxC,
4                                            NbResistance #>=
                                                 NbCondensateur.
5
6 /* Tests
7 [eclipse 92]: commande2(NbResistance,NbCondensateur).
8
9 NbResistance = NbResistance{9000 .. 10000}
10 NbCondensateur = NbCondensateur{9000 .. 10000}
11
12
13 Delayed goals:
14         NbCondensateur{9000 .. 10000} - NbResistance{9000 .. 10000} #=< 0
```

```
Yes (0.00s cpu)
*/
```

Ici, la réponse d'eclipse est que les intervalles réponses pour les 2 variables sont [9000,10000].

Sens : s'il y a des solutions alors elles sont dans ces intervalles

Question 1.8 Nous implémentons ici le prédicat *commande3/2*, basé sur *commande2/2*, en utilisant *labeling*.

Listing 5 – "prédicat Commande"

```
commande3(NbResistance,NbCondensateur) :-
    ,nbMaxC(NbMaxC),nbMinC(NbMinC),
                                        nbMaxR(NbMaxR),nbMinR(NbMinR)
                                        NbResistance #:: NbMinR..
                                            NbMaxR,
                                        NbCondensateur #:: NbMinC..
                                            NbMaxC,
                                        NbResistance #>=
                                            NbCondensateur,
                                        labeling([NbResistance,
                                            NbCondensateur]).

/* Tests
[eclipse 91]: commande3(NbResistance,NbCondensateur).

NbResistance = 9000
NbCondensateur = 9000
Yes (0.00s cpu, solution 1, maybe more) ? ;

NbResistance = 9001
NbCondensateur = 9000
Yes (0.00s cpu, solution 2, maybe more) ? ;

NbResistance = 9001
NbCondensateur = 9001
Yes (0.00s cpu, solution 3, maybe more) ? ;

NbResistance = 9002
NbCondensateur = 9000
Yes (0.00s cpu, solution 4, maybe more) ?

*/
```

Ici, on voit clairement que le temps de réponse est vraiment plus rapide (moins d'un centième de seconde), et ceci est dû au fait que le labeling, à savoir le solveur ic, regarde toutes les contraintes avant de générer les solutions. Ce qui signifie qu'ici, il a réduit à l'avance les intervalles de solutions minimales avant de générer les réponses. C'est un gain de temps énorme pour le cas ici présent puisque l'arbre de recherche de Prolog (figure 2, page 6) ne rencontre jamais d'échecs car il se construit dans les bonnes intervalles.

OK

idem, il faut faire apparaître les domaines et les contraintes posées

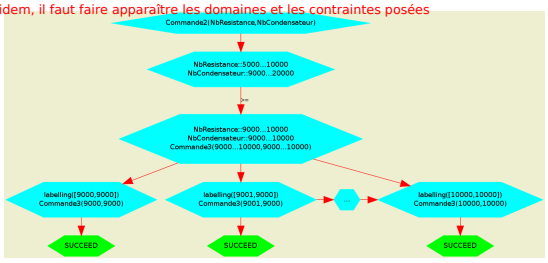

FIGURE 2 – Arbre Prolog 2

2 Zoologie

Pour répondre aux questions, nous avons implémenté le prédicat *chapie/4*.

Listing 6 – "Prédicat chapie"

```
nbChatsMax(1000).
nbPiesMax(1000).

chapie(Chats,Pies,Pattes,Tetes) :-
                                        nbChatsMax(NbChatsMax),
                                        nbPiesMax(NbPiesMax),
                                        Chats #:: 0..NbChatsMax,
                                        Pies #:: 0..NbPiesMax,
                                        Pattes #= Chats*4+Pies*2,
                                        Tetes #= Chats+Pies,
                                        labeling([Chats,Pies]).
```

Question 1.9 Il faut donc 3 pies et 14 pattes pour totaliser 5 têtes et deux chats.

Listing 7 – "Prédicat chapie - test"

```
[eclipse 52]: chapie(2,Pies,Pattes,5).

Pies = 3
Pattes = 14
Yes (0.00s cpu)
```

Question 1.10 Ici, il y a donc une infinité de solutions. La première donnée par le solveur est 0,0,0,0 puis 1,1,6,2, etc.

Vraiment ???

Listing 8 – "Prédicat chapie - test"

```
[eclipse 4]: chapie(Chats,Pies,Pattes,Tetes),Pattes#=Tetes*3.

Chats = 0
Pies = 0
Pattes = 0
Tetes = 0
Yes (0.00s cpu, solution 1, maybe more) ? ;

Chats = 1
Pies = 1
Pattes = 6
Tetes = 2
Yes (0.00s cpu, solution 2, maybe more) ? ;

Chats = 2
Pies = 2
Pattes = 12
Tetes = 4
Yes (0.00s cpu, solution 3, maybe more) ?
```

3 Le "OU" en contraintes

Question 1.11 Les deux prédicats ont ici exactement la même attitude pour ce cas de figure.

Listing 9 – "Prédicat vabs - vabs2"

```
vabs1(Val,AbsVal) :- Val#=AbsVal,
                                        AbsVal#>=0,
                                        labeling([Val,AbsVal]).
vabs1(Val,AbsVal) :- Val*(-1)#=AbsVal,
                                        AbsVal#>0,
                                        labeling([Val,AbsVal]).

vabs2(0,0).
vabs2(Val,AbsVal) :-    Val#=AbsVal
                                        or
                                        Val*(-1)#=AbsVal,
                                        AbsVal#>0,
                                        labeling([Val,AbsVal]).

/* Tests
[eclipse 5]: vabs1(-2,Y).

Y = 2
Yes (0.00s cpu)
[eclipse 6]: vabs2(-2,Y).

Y = 2
Yes (0.00s cpu)
```

JAMAIS de labeling dans un prédicat qui définit une contrainte. Le labeling c'est uniquement lorsque vous cherchez des solutions.


```
[eclipse 7]: vabs1(X,3).

X = 3
Yes (0.00s cpu, solution 1, maybe more) ? ;

X = -3
Yes (0.00s cpu, solution 2)
[eclipse 8]: vabs2(X,3).

X = X[-1.0Inf .. 1.0Inf]

Delayed goals:
        #=(X[-1.0Inf .. 1.0Inf], 3, _180{[0, 1]})
        #=(-(X[-1.0Inf .. 1.0Inf]), 3, _302{[0, 1]})
        -(_302{[0, 1]}) - _180{[0, 1]} #=< -1
Yes (0.00s cpu)
*/
```

Etonnant, vu votre code j'attends une erreur d'instanciation des variables, lors de l'appel au prédicat labeling...

Ici, seul la première version du prédicat donne les deux résultats corrects attendus.

La deuxième version est correcte : les contraintes sont posées ! Un labeling fait au bon moment vous redonnera bien les solutions attendues !

Question 1.12 Nous n'utiliserons pas les tests que le premier de la place. Pour la première version du prédicat, cela donne bien toutes les solutions possibles, d'abord pour les X positifs puis pour les négatifs. Pour la deuxième version, le prédicat donne d'abord les solution 0,0, puis les solutions négatives pour X, puis les positives.

Ce qu'il fallait voir c'était que seule la version 1 donne des solutions. La version 2 nécessite un labeling.

Question 1.13 Définition du prédicat *faitListe/4* :

Listing 10 – "Prédicat faitListe"

```
faitListe(VarListe,Taille,Min,Max) :-    dim(VarListe,[Taille]),
        (for(I,1,Taille),param(VarListe,Min,Max) do
        (Elem #:: Min..Max,
        indomain(Elem),
        VarListe[I] #= Elem
        )
        ).

/* Test
[eclipse 45]: faitListe(L,2,1,2).

L = [](1, 1)
Yes (0.00s cpu, solution 1, maybe more) ? ;

L = [](1, 2)
Yes (0.00s cpu, solution 2, maybe more) ? ;

L = [](2, 1)
Yes (0.00s cpu, solution 3, maybe more) ? ;

L = [](2, 2)
Yes (0.00s cpu, solution 4)
```

Indentez mieux...

faitListe ne doit pas instancier les variables, seulement poser les contraintes.

```
Autre test
[eclipse 46]: faitListe([](1,6,3),T,2,8).

No (0.00s cpu)

*/
```

Les résultats du test sont logiques, puisque 1 n'est pas entre Min et Max !

Question 1.14 Définition du prédicat *suite/1* :

Listing 11 – "Prédicat suite"

```
suite(ListVar) :-       dim(ListVar,[Taille]),
                                (for(I,1,Taille-2),param(ListVar) do
                                        (Temp #= ListVar[I+1],
                                        vabs1(Temp,VabsDeuxElem),
                                        ListVar[I+2] #= VabsDeuxElem
                                            - ListVar[I]
                                        )
                                ).
/* Test
[eclipse 47]: suite([](3,-5,2)).

Yes (0.00s cpu)

Autre test
[eclipse 48]: suite([](2,2,3)).

No (0.00s cpu)
*/
```

is

Question 1.15 Cette requête permet de visualiser les suites formées par le prédicat avec les 2 premiers éléments de cette suite compris entre 1 et 10. Effectivement, il semble y avoir périodicité entre le premier et le dixième élément (et donc une période de 9).

Listing 12 – "Réquête"

```
% dim(L,[10]),L[1]#=1,L[2]#=2,suite(L).
% dim(L,[10]),L[1]#=3,L[2]#=7,suite(L).
/*
 dim(L,[10]),
        Elem1#::1..10,Elem2#::1..10,
        indomain(Elem1),indomain(Elem2),
        L[1]#=Elem1,L[2]#=Elem2,
        suite(L).
*/
/*
 dim(L,[10]),
        Elem1#::1..10,Elem2#::1..10,
        indomain(Elem1),indomain(Elem2),
```

Utilisez plutôt labeling !!!

Il fallait utiliser faitListe


```
        L[1]#=Elem1,L[2]#=Elem2,
                L[1]#\=L[10],
        suite(L).
dim(L,[10]),
        Elem1#::1..100,Elem2#::1..100,
        indomain(Elem1),indomain(Elem2),
        L[1]#=Elem1,L[2]#=Elem2,
                L[1]#\=L[10],
        suite(L).
No (0.94s cpu)

No (93.06s cpu)

*/
```

Requête vérifiant que la suite est périodique de période 9 (vérifie qu'il n'existe aucune suite L ayant son premier et son dixième élément différents avec les deux premiers éléments de la suite compris entre 1 et 10).
Prolog répond bien non à cette requête. On réitère alors avec les deux premiers éléments de la suite compris entre 1 et 100.
La réponse est toujours non.
Enfin, on essaie avec les deux premiers éléments de la suite compris entre 1 et 1000. Après plus d'une minute, Prolog répond toujours non.

OK

Pour conclure, dans le prédicat suite, on remarquera que l'on est dans de la propagation et non du backtracking puisque l'on construit la suite au fur et à mesure du parcourt de l'arbre à l'exécution. Il n'y a donc pas besoin de faire appel au prédicat labeling.

ATTENTION : vous faites un équivalent du labeling avec le prédicat indomain/1

4 Code Complet, avec tests

Listing 13 – "test"

```
:- lib(ic).

%Question 1.1

couleurVoiture(rouge).
couleurVoiture(vert(clair)).
couleurVoiture(gris).
couleurVoiture(blanc).

couleurBateau(vert(_)).
couleurBateau(noir).
couleurBateau(blanc).


choixCouleur(CouleurBateau,CouleurVoiture) :-   couleurVoiture(CouleurVoiture
    ),
                                                couleurBateau(CouleurBateau),
                                                CouleurVoiture=CouleurBateau.

/* Tests
[eclipse 78]: choixCouleur(CouleurBateau,CouleurVoiture).

CouleurBateau = vert(clair)
CouleurVoiture = vert(clair)
Yes (0.00s cpu, solution 1, maybe more) ? ;

CouleurBateau = blanc
CouleurVoiture = blanc
Yes (0.00s cpu, solution 2)
*/

%Question 1.3

isBetween(Var,Min,_Max) :-      Var=Min.
isBetween(Var,Min,Max) :-       \=(Min,Max),
                                is(M,Min+1),
                                isBetween(Var,M,Max).

%Question 1.4

nbMaxR(10000).
nbMinR(5000).
nbMinC(9000).
nbMaxC(20000).

commande(NbResistance,NbCondensateur) :-        nbMaxR(NbMaxR),nbMinR(NbMinR)
    ,nbMaxC(NbMaxC),nbMinC(NbMinC),
                                                isBetween(NbResistance,NbMinR
                                                    ,NbMaxR),
```



```
                                                isBetween(NbCondensateur,
                                                    NbMinC,NbMaxC),
                                                NbResistance>=NbCondensateur.

/* Tests
[eclipse 79]: commande(NbResistance,NbCondensateur).

NbResistance = 9000
NbCondensateur = 9000
Yes (7.73s cpu, solution 1, maybe more) ? ;

NbResistance = 9001
NbCondensateur = 9000
Yes (7.74s cpu, solution 2, maybe more) ? ;

NbResistance = 9001
NbCondensateur = 9001
Yes (7.74s cpu, solution 3, maybe more) ?
*/

%Question 1.6

commandeBis(NbResistance,NbCondensateur) :-     nbMaxR(NbMaxR),nbMinR(NbMinR)
    ,nbMaxC(NbMaxC),nbMinC(NbMinC),
                                                NbResistance>=NbCondensateur,
                                                isBetween(NbResistance,NbMinR
                                                    ,NbMaxR),
                                                isBetween(NbCondensateur,
                                                    NbMinC,NbMaxC).


/* Test
[eclipse 82]: commandeBis(NbResistance,NbCOndensateur).
instantiation fault in NbResistance >= NbCOndensateur
Abort

*/

%Question 1.7

commande2(NbResistance,NbCondensateur) :-       nbMaxR(NbMaxR),nbMinR(NbMinR)
    ,nbMaxC(NbMaxC),nbMinC(NbMinC),
                                                NbResistance #:: NbMinR..
                                                    NbMaxR,
                                                NbCondensateur #:: NbMinC..
                                                    NbMaxC,
                                                NbResistance #>=
                                                    NbCondensateur.

/* Tests
[eclipse 92]: commande2(NbResistance,NbCondensateur).

NbResistance = NbResistance{9000 .. 10000}
```

```
NbCondensateur = NbCondensateur[9000 .. 10000]


Delayed goals:
        NbCondensateur[9000 .. 10000] - NbResistance[9000 .. 10000] #=< 0
Yes (0.00s cpu)
*/

%Question 1.8

commande3(NbResistance,NbCondensateur) :-       nbMaxR(NbMaxR),nbMinR(NbMinR)
    ,nbMaxC(NbMaxC),nbMinC(NbMinC),
                                                NbResistance #:: NbMinR..
                                                    NbMaxR,
                                                NbCondensateur #:: NbMinC..
                                                    NbMaxC,
                                                NbResistance #>=
                                                    NbCondensateur,
                                                labeling([NbResistance,
                                                    NbCondensateur]).

/* Tests
[eclipse 91]: commande3(NbResistance,NbCondensateur).

NbResistance = 9000
NbCondensateur = 9000
Yes (0.00s cpu, solution 1, maybe more) ? ;

NbResistance = 9001
NbCondensateur = 9000
Yes (0.00s cpu, solution 2, maybe more) ? ;

NbResistance = 9001
NbCondensateur = 9001
Yes (0.00s cpu, solution 3, maybe more) ? ;

NbResistance = 9002
NbCondensateur = 9000
Yes (0.00s cpu, solution 4, maybe more) ?

*/

%Zoologie
%Question 1.9

nbChatsMax(1000).
nbPiesMax(1000).

chapie(Chats,Pies,Pattes,Tetes) :-      nbChatsMax(NbChatsMax),
                                        nbPiesMax(NbPiesMax),
                                        Chats #:: 0 .. NbChatsMax,
                                        Pies #:: 0 .. NbPiesMax,
                                        Pattes #= Chats*4+Pies*2,
```



```
                                        Tetes #= Chats+Pies,
                                        labeling([Chats,Pies]).
/*
[eclipse 52]: chapie(2,Pies,Pattes,5).

Pies = 3
Pattes = 14
Yes (0.00s cpu)
*/

% Question 1.10

/*
[eclipse 4]: chapie(Chats,Pies,Pattes,Tetes),Pattes#=Tetes*3.

Chats = 0
Pies = 0
Pattes = 0
Tetes = 0
Yes (0.00s cpu, solution 1, maybe more) ? ;

Chats = 1
Pies = 1
Pattes = 6
Tetes = 2
Yes (0.00s cpu, solution 2, maybe more) ? ;

Chats = 2
Pies = 2
Pattes = 12
Tetes = 4
Yes (0.00s cpu, solution 3, maybe more) ?
*/

% Question 1.11

vabs1(Val,AbsVal) :- Val#=AbsVal,
                                        AbsVal#>=0,
                                        labeling([Val,AbsVal]).
vabs1(Val,AbsVal) :- Val*(-1)#=AbsVal,
                                        AbsVal#>0,
                                        labeling([Val,AbsVal]).

vabs2(0,0).
vabs2(Val,AbsVal) :-    Val#=AbsVal
                                    or
                                        Val*(-1)#=AbsVal,
                                        AbsVal#>0,
                                        labeling([Val,AbsVal]).

/* Tests
[eclipse 5]: vabs1(-2,Y).
```



```
Y = 2
Yes (0.00s cpu)
[eclipse 6]: vabs2(-2,Y).

Y = 2
Yes (0.00s cpu)


[eclipse 7]: vabs1(X,3).

X = 3
Yes (0.00s cpu, solution 1, maybe more) ? ;

X = -3
Yes (0.00s cpu, solution 2)
[eclipse 8]: vabs2(X,3).

X = X[-1.0Inf .. 1.0Inf]


Delayed goals:
        #=(X[-1.0Inf .. 1.0Inf], 3, _180{[0, 1]})
        #=(-(X[-1.0Inf .. 1.0Inf]), 3, _302{[0, 1]})
        -(_302{[0, 1]}) - _180{[0, 1]} #=< -1
Yes (0.00s cpu)
*/

% Question 1.13

faitListe(VarListe,Taille,Min,Max) :-   dim(VarListe,[Taille]),
                                        (for(I,1,Taille),param(VarListe,Min,Max) do
                                        (Elem #:: Min..Max,
                                        indomain(Elem),
                                        VarListe[I] #= Elem
                                        )
                                    ).

/* Test
[eclipse 45]: faitListe(L,2,1,2).

L = []{1, 1}
Yes (0.00s cpu, solution 1, maybe more) ? ;

L = []{1, 2}
Yes (0.00s cpu, solution 2, maybe more) ? ;

L = []{2, 1}
Yes (0.00s cpu, solution 3, maybe more) ? ;

L = []{2, 2}
Yes (0.00s cpu, solution 4)

Autre test
```



```
[eclipse 46]: faitListe([]{1,6,3},T,2,8).

No (0.00s cpu)

*/

% Question 1.14

suite(ListVar) :-       dim(ListVar,[Taille]),
                                        (for(I,1,Taille-2),param(ListVar) do
                                        (Temp #= ListVar[I+1],
                                        vabs1(Temp,VabsDeuxElem),
                                        ListVar[I+2] #= VabsDeuxElem
                                            - ListVar[I]
                                        )
                                        ).
/* Test
[eclipse 47]: suite([]{3,-5,2}).

Yes (0.00s cpu)

Autre test
[eclipse 48]: suite([]{2,2,3}).

No (0.00s cpu)
*/

% Question 1.15
% dim(L,[10]),L[1]#=1,L[2]#=2,suite(L).
% dim(L,[10]),L[1]#=3,L[2]#=7,suite(L).
/*
 dim(L,[10]),
        Elem1#::1..10,Elem2#::1..10,
        indomain(Elem1),indomain(Elem2),
        L[1]#=Elem1,L[2]#=Elem2,
        suite(L).
*/
/*

 dim(L,[10]),
        Elem1#::1..10,Elem2#::1..10,
        indomain(Elem1),indomain(Elem2),
        L[1]#=Elem1,L[2]#=Elem2,
                L[1]#\=L[10],
        suite(L).
dim(L,[10]),
        Elem1#::1..100,Elem2#::1..100,
        indomain(Elem1),indomain(Elem2),
        L[1]#=Elem1,L[2]#=Elem2,
                L[1]#\=L[10],
        suite(L).
No (0.94s cpu)
```

```
298 No (93.06s cpu)
299
300 */
```

Excellent travail

Rapport Travaux Pratiques : Programmation par Contraintes - TP 3 : **Contraintes Logiques**

Nicolas Desfeux
Aurélien Texier

15 mars 2011

Dans ce T.P., nous allons utiliser la programmation par contraintes pour résoudre un problème d'ordonnancement de tâches à effectuer sur deux machines.
Dans un premier temps, nous définirons les prédicats qui fixent les domaines dans lesquels nous travaillerons, puis nous ajouterons les contraintes liées au fait que les tâches doivent être effectuées seulement après que certaines autres soient faites. Enfin, nous finirons par une dernière contraintes qui vise à empêcher que deux tâches se fassent simultanément sur la même machine.

Question 3.1 Nous définissons ici un prédicat *taches(?Taches)* qui unifie Taches au tableau des tâches.

Listing 1 – "taches"

```
1  taches(Taches) :-       Taches =          [](tache(3,[],m1,_),
2                                              tache(8,[],m1,_),
3                                              tache(8,[4,5],m1,_),
4                                              tache(6,[],m2,_),
5                                              tache(3,[1],m2,_),
6                                              tache(4,[1,7],m1,_),
7                                              tache(8,[3,5],m1,_),
8                                              tache(6,[4],m2,_),
9                                              tache(6,[6,7],m2,_),
10                                             tache(6,[9,12],m2,_),
11                                             tache(3,[1],m2,_),
12                                             tache(6,[7,8],m2,_)).
13
14 /*Test
15
16 [eclipse 3]: taches(T).
17
18 T = [tache(3, [], m1, _169), tache(8, [], m1, _176), tache(8, [4, 5], m1,
       _183), tache(6, [], m2, _194), tache(3, [1], m2, _201), tache(4, [1, 7],
       m1, _210), tache(8, [3, 5], m1, _221), tache(6, [4], m2, _232), tache(6,
```

OK


```
       [6, 7], m2, _241), tache(6, [9, 12], m2, _252), tache(3, [1], m2, _263),
       tache(6, [7, 8], m2, _272)]
19 Yes (0.00s cpu)
20
21 */
```

Question 3.2 Nous définissons ici un prédicat *affiche(+Taches)* qui affiche chaque élément, à savoir chaque tâche constituant le problème. Nous définiront ce prédicat à l'aide d'un itérateur.

Listing 2 – "affiche"

```
1  affiche(Taches) :-      dim(Taches,[Dim]),
2                          (for(Indice,1,Dim),param(Taches)
3                                                  do
4                                                      Elem is Taches[Indice
                                                           ],
5                                                      writeln(Elem)
6                          ).
7
8
9  /*                                                  OK
10 [eclipse 4]: taches(T),affiche(T).
11 tache(3, [], m1, _235)
12 tache(8, [], m1, _242)
13 tache(8, [4, 5], m1, _249)
14 tache(6, [], m2, _260)
15 tache(3, [1], m2, _267)
16 tache(4, [1, 7], m1, _276)
17 tache(8, [3, 5], m1, _287)
18 tache(6, [4], m2, _298)
19 tache(6, [6, 7], m2, _307)
20 tache(6, [9, 12], m2, _318)
21 tache(3, [1], m2, _329)
22 tache(6, [7, 8], m2, _338)
23
24 T = [tache(3, [], m1, _235), tache(8, [], m1, _242), tache(8, [4, 5], m1,
       _249), tache(6, [], m2, _260), tache(3, [1], m2, _267), tache(4, [1, 7],
       m1, _276), tache(8, [3, 5], m1, _287), tache(6, [4], m2, _298), tache(6,
       [6, 7], m2, _307), tache(6, [9, 12], m2, _318), tache(3, [1], m2, _329),
       tache(6, [7, 8], m2, _338)]
25 Yes (0.00s cpu)
26
27 */
```

Question 3.3 Nous définissons ici un prédicat *domaines(+Taches, ?Fin)* qui contraint chaque tâche à commencer après l'instant 0 et à finir avant Fin, variable qui correspond à l'instant où toutes les tâches sont terminées.

Listing 3 – "domaines"

```
1  domaines(Taches,Fin) :- dim(Taches,[Dim]),
2                          (for(Indice,1,Dim),param(Taches,Fin)
```



```
3                                                  do
4                                                      tache(Duree,_Nom,
                                                           _Machine,Debut)
                                                           is Taches[Indice
                                                           ],
5                                                      Debut + Duree #=< Fin
                                                           ,
6                                                      Debut #>= 0
7                          ).
8                                                  Oui
9  /*
10 [eclipse 5]: taches(T),domaines(T,10).
11
12 T = [tache(3, [], m1, _421[0 .. 7]), tache(8, [], m1, _644[0 .. 2]), tache(8,
       [4, 5], m1, _867[0 .. 2]), tache(6, [], m2, _1090[0 .. 4]), tache(3,
       [1], m2, _1313[0 .. 7]), tache(4, [1, 7], m1, _1536[0 .. 6]), tache(8,
       [3, 5], m1, _1759[0 .. 2]), tache(6, [4], m2, _1982[0 .. 4]), tache(6,
       [6, 7], m2, _2205[0 .. 4]), tache(6, [9, 12], m2, _2428[0 .. 4]), tache
       (3, [1], m2, _2651[0 .. 7]), tache(6, [7, 8], m2, _2874[0 .. 4])]
13
14 There are 12 delayed goals. Do you want to see them? (y\n)
15 Yes (0.00s cpu)
16 */
```

Question 3.4 Voici le prédicat *getVarList(+Taches, ?Fin, ?List)* qui permet de récupérer la liste des variables du problème.

Listing 4 – "getVarList"

```
1  getVarList(Taches,Fin,ListFin):- dim(Taches,[Dim]),
2                                   (for(Indice,1,Dim),fromto([],In,Out,List),
                                        param(Taches)
3                                   do
4                                       Xi is Taches[Indice],
5                                       Xi = tache(_,_,_,Debut),
6                                       Out=[Debut|In]
7                                   ),
8                                   ListFin=[Fin|List].
9                                                  Bien
10
11 /*
12 [eclipse 6]: taches(T), getVarList(T, Fin, L).
13
14 T = [](tache(3, [], m1, _238), tache(8, [], m1, _243), tache(8, [4, 5], m1,
       _248), tache(6, [], m2, _257), tache(3, [1], m2, _262), tache(4, [1, 7],
       m1, _269), tache(8, [3, 5], m1, _278), tache(6, [4], m2, _287), tache(6,
       [6, 7], m2, _294), tache(6, [9, 12], m2, _303), tache(3, [1], m2, _312),
       tache(6, [7, 8], m2, _319))
15 Fin = Fin
16 L = [Fin, _319, _312, _303, _294, _287, _278, _269, _262, _257, _248, _243,
       _238]
17 Yes (0.00s cpu)
18
```



```
19
20 */
```

Question 3.5 On définit le prédicat *solve(?Taches, ?Fin)* qui permet, en utilisant les trois prédicats précédents, de trouver un ordonnancement qui respecte les contraintes de domaines définies.
Le test effectué atteste bien la conformité du prédicat car solve rend bien comme première solution toutes les tâches avec des débuts à 0, puis il incrémente chacune comme solutions suivantes. Il unifie Fin à 8 puisque c'est la durée de la tâche la plus longue, ce qui est logique aussi.

Très bien

Listing 5 – "solve1"

```
1  solve1(Taches,Fin) :-   taches(Taches),
2                                                  domaines(Taches,Fin),
3                                                  getVarList(Taches,Fin,Liste),
4                                                  labeling(Liste),
5                                                  affiche(Taches).
6  /*
7  [eclipse 4]: solve1(Taches,Fin).
8  lists.eco  loaded in 0.00 seconds
9  tache(3, [], m1, 0)
10 tache(8, [], m1, 0)
11 tache(8, [4, 5], m1, 0)
12 tache(6, [], m2, 0)
13 tache(3, [1], m2, 0)
14 tache(4, [1, 7], m1, 0)
15 tache(8, [3, 5], m1, 0)
16 tache(6, [4], m2, 0)
17 tache(6, [6, 7], m2, 0)
18 tache(6, [9, 12], m2, 0)
19 tache(3, [1], m2, 0)
20 tache(6, [7, 8], m2, 0)
21
22 Taches = [](tache(3, [], m1, 0), tache(8, [], m1, 0), tache(8, [4, 5], m1, 0)
       , tache(6, [], m2, 0), tache(3, [1], m2, 0), tache(4, [1, 7], m1, 0),
       tache(8, [3, 5], m1, 0), tache(6, [4], m2, 0), tache(6, [6, 7], m2, 0),
       tache(6, [9, 12], m2, 0), tache(3, [1], m2, 0), tache(6, [7, 8], m2, 0))
23 Fin = 8
24 Yes (0.00s cpu, solution 1, maybe more) ? ;
25 tache(3, [], m1, 1)
26 tache(8, [], m1, 0)
27 tache(8, [4, 5], m1, 0)
28 tache(6, [], m2, 0)
29 tache(3, [1], m2, 0)
30 tache(4, [1, 7], m1, 0)
31 tache(8, [3, 5], m1, 0)
32 tache(6, [4], m2, 0)
33 tache(6, [6, 7], m2, 0)
34 tache(6, [9, 12], m2, 0)
35 tache(3, [1], m2, 0)
36 tache(6, [7, 8], m2, 0)
37
```

```
Taches = [](tache(3, [], m1, 1), tache(8, [], m1, 0), tache(8, [4, 5], m1, 0)
    , tache(6, [], m2, 0), tache(3, [1], m2, 0), tache(4, [1, 7], m1, 0),
    tache(8, [3, 5], m1, 0), tache(6, [4], m2, 0), tache(6, [6, 7], m2, 0),
    tache(6, [9, 12], m2, 0), tache(3, [1], m2, 0), tache(6, [7, 8], m2, 0))
Fin = 8
Yes (0.01s cpu, solution 2, maybe more) ? ;
tache(3, [], m1, 2)
tache(8, [], m1, 0)
tache(8, [4, 5], m1, 0)
tache(6, [], m2, 0)
tache(3, [1], m2, 0)
tache(4, [1, 7], m1, 0)
tache(8, [3, 5], m1, 0)
tache(6, [4], m2, 0)
tache(6, [6, 7], m2, 0)
tache(6, [9, 12], m2, 0)
tache(3, [1], m2, 0)
tache(6, [7, 8], m2, 0)

Taches = [](tache(3, [], m1, 2), tache(8, [], m1, 0), tache(8, [4, 5], m1, 0)
    , tache(6, [], m2, 0), tache(3, [1], m2, 0), tache(4, [1, 7], m1, 0),
    tache(8, [3, 5], m1, 0), tache(6, [4], m2, 0), tache(6, [6, 7], m2, 0),
    tache(6, [9, 12], m2, 0), tache(3, [1], m2, 0), tache(6, [7, 8], m2, 0))
Fin = 8
Yes (0.01s cpu, solution 3, maybe more) ?
*/
```

Question 3.6 On définit ici un prédicat *precedences(+Taches)* qui contraint chaque tâche à démarrer après la fin de ses tâches préliminaires.
On modifie alors solve pour prendre en compte ces contraintes.
Les résultats de solve sont bien conformes car les deux premières tâches commencent à 0 puisqu'elles n'ont besoin d'aucune autre tâche effectuée au préalable pour s'effectuer. La tâche 5 qui a besoin de la tâche 1 effectuée commence bien à 3, qui est la durée de la tâche 1. Tout est donc correct.
On peut donc en conclure que sans la contrainte des machines différentes qui va suivre, la solution optimale prendrait 38 unités de temps pour se faire (car le solveur unifie Fin à 38).

Listing 6 – "precedences"

```
precedences(Taches) :-  dim(Taches,[Dim]),
                        (for(Indice,1,Dim),param(Taches)
                        do
                                Elem is Taches[Indice],
                                Elem = tache(_D,Noms,_M,Debut),
                                (foreach(I,Noms),param(Debut,Taches)
                                do
                                        tache(Duree2,_N,_M,Debut2) is Taches[
                                            I],
                                        Debut #>= Debut2+Duree2
                                )
                        ).
```

Bien


```
solve(Taches,Fin) :-    taches(Taches),
                                                domaines(Taches,Fin),
                                                precedences(Taches),
                                                getVarList(Taches,Fin,Liste),
                                                labeling(Liste),
                                                affiche(Taches).

/*
[eclipse 44]: solve(T, Fin).
tache(3, [], m1, 0)
tache(8, [], m1, 0)
tache(8, [4, 5], m1, 6)
tache(6, [], m2, 0)
tache(3, [1], m2, 3)
tache(4, [1, 7], m1, 22)
tache(8, [3, 5], m1, 14)
tache(6, [4], m2, 6)
tache(6, [6, 7], m2, 26)
tache(6, [9, 12], m2, 32)
tache(3, [1], m2, 3)
tache(6, [7, 8], m2, 22)

T = [](tache(3, [], m1, 0), tache(8, [], m1, 0), tache(8, [4, 5], m1, 6),
    tache(6, [], m2, 0), tache(3, [1], m2, 3), tache(4, [1, 7], m1, 22),
    tache(8, [3, 5], m1, 14), tache(6, [4], m2, 6), tache(6, [6, 7], m2, 26),
     tache(6, [9, 12], m2, 32), tache(3, [1], m2, 3), tache(6, [7, 8], m2,
    22))
Fin = 38

*/
```

Question 3.7 Enfin, on définit le prédicat *conflits(+Taches)* qui impose que, sur une machine, deux tâches ne se déroulent pas en même temps.
On modifie solve de la même manière qu'à la question précédente pour obtenir une solution du problème prenant en compte cette dernière contrainte.
Enfin, avec cette dernière contrainte, on obtient une solution qui dure un peu plus longtemps, 43 unités de temps.

Listing 7 – "conflits"

```
conflits(Taches) :- dim(Taches,[Dim]),
                    (for(Indice,1,Dim),param(Taches,Dim)
                    do
                            Elem is Taches[Indice],
                            I2 is Indice+1,
                            /*Il faut assurer que les indices soient
                                differents, sinon on va se retrouver a
                                comparer deux fois la meme taches*/
                            (for(I,I2,Dim),param(Taches,Elem)
                            do
                                    Elem2 is Taches[I],
```



```
                                                machinesDifferentes(Elem,Elem2)
                                )
                        ).

machinesDifferentes(tache(_,_,M1,_),tache(_,_,M2,_)):- \=(M1,M2),!.
machinesDifferentes(tache(Duree,_,M,Debut),tache(Duree2,_,M,Debut2)) :- ((
    Debut #>= Debut2+Duree2) or (Debut+Duree #=< Debut2)).
```

TB

```
solve2(Taches,Fin) :-   taches(Taches),
                                                domaines(Taches,Fin),
                                                precedences(Taches),
                                                conflits(Taches),
                                                getVarList(Taches,Fin,Liste),
                                                labeling(Liste),
                                                affiche(Taches).
/*
[eclipse 10]: solve2(Taches,Fin).
tache(3, [], m1, 0)
tache(8, [], m1, 29)
tache(8, [4, 5], m1, 9)
tache(6, [], m2, 0)
tache(3, [1], m2, 6)
tache(4, [1, 7], m1, 25)
tache(8, [3, 5], m1, 17)
tache(6, [4], m2, 12)
tache(6, [6, 7], m2, 31)
tache(6, [9, 12], m2, 37)
tache(3, [1], m2, 9)
tache(6, [7, 8], m2, 25)

Taches = [](tache(3, [], m1, 0), tache(8, [], m1, 29), tache(8, [4, 5], m1,
    9), tache(6, [], m2, 0), tache(3, [1], m2, 6), tache(4, [1, 7], m1, 25),
    tache(8, [3, 5], m1, 17), tache(6, [4], m2, 12), tache(6, [6, 7], m2, 31)
    , tache(6, [9, 12], m2, 37), tache(3, [1], m2, 9), tache(6, [7, 8], m2,
    25))
Fin = 43
Yes (0.01s cpu, solution 1, maybe more) ? ;
tache(3, [], m1, 1)
tache(8, [], m1, 29)
tache(8, [4, 5], m1, 9)
tache(6, [], m2, 0)
tache(3, [1], m2, 6)
tache(4, [1, 7], m1, 25)
tache(8, [3, 5], m1, 17)
tache(6, [4], m2, 12)
tache(6, [6, 7], m2, 31)
tache(6, [9, 12], m2, 37)
tache(3, [1], m2, 9)
tache(6, [7, 8], m2, 25)

Taches = [](tache(3, [], m1, 1), tache(8, [], m1, 29), tache(8, [4, 5], m1,
    9), tache(6, [], m2, 0), tache(3, [1], m2, 6), tache(4, [1, 7], m1, 25),
```



```
    tache(8, [3, 5], m1, 17), tache(6, [4], m2, 12), tache(6, [6, 7], m2, 31)
    , tache(6, [9, 12], m2, 37), tache(3, [1], m2, 9), tache(6, [7, 8], m2,
    25))
Fin = 43
Yes (0.01s cpu, solution 2, maybe more) ? ;
tache(3, [], m1, 2)
tache(8, [], m1, 29)
tache(8, [4, 5], m1, 9)
tache(6, [], m2, 0)
tache(3, [1], m2, 6)
tache(4, [1, 7], m1, 25)
tache(8, [3, 5], m1, 17)
tache(6, [4], m2, 12)
tache(6, [6, 7], m2, 31)
tache(6, [9, 12], m2, 37)
tache(3, [1], m2, 9)
tache(6, [7, 8], m2, 25)

Taches = [](tache(3, [], m1, 2), tache(8, [], m1, 29), tache(8, [4, 5], m1,
    9), tache(6, [], m2, 0), tache(3, [1], m2, 6), tache(4, [1, 7], m1, 25),
    tache(8, [3, 5], m1, 17), tache(6, [4], m2, 12), tache(6, [6, 7], m2, 31)
    , tache(6, [9, 12], m2, 37), tache(3, [1], m2, 9), tache(6, [7, 8], m2,
    25))
Fin = 43
Yes (0.01s cpu, solution 3, maybe more) ?
*/

/*
[eclipse 11]: solve2(Taches,42).

No (0.00s cpu)
*/
```

Question 3.8 Oui, la solution est la meilleure ! Prolog résoud les contraintes en incrémentant le début des tâches, jusqu'à obtenir le respect des contraintes. Comme il incrémente, la première trouvée est candidate pour la solution optimale au problème, à savoir l'ordonnancement des tâches au plus tôt. Seulement, rien ne nous assure que la première est la meilleure, car cela dépend de l'ordre de la déclaration des tâches au début, dans le prédicat *taches(?Taches)*.
Pour vérifier donc, nous avons, dans nos tests de la question précédente, effectué une requête de *solve2* avec Fin à 42 (au lieu de la solution donnée qui est 43), et le solveur nous répond No, ce qui justifie bien qu'il n'y a pas de solution plus courte pour ce problème d'ordonnancement. Donc nous avons eu dans notre cas la solution optimale.

Oui !

A priori, il n'y a aucune raison pour que la première solution soit la meilleure. Pour garantir que la première solution est la meilleure, il faut commencer l'énumération par la variable Fin. Ainsi, vous chercherez des solutions en instanciant Fin par ordre croissant -> La première solution sera donc la meilleure.

1 Code Complet, avec l'ensemble des tests

Listing 8 – "TP3"

```
1  :-lib(ic).
2  :-lib(ic_symbolic).
3
4  taches(Taches) :-       Taches =        [](tache(3,[],m1,_),
5                                          tache(8,[],m1,_),
6                                          tache(8,[4,5],m1,_),
7                                          tache(6,[],m2,_),
8                                          tache(3,[1],m2,_),
9                                          tache(4,[1,7],m1,_),
10                                         tache(8,[3,5],m1,_),
11                                         tache(6,[4],m2,_),
12                                         tache(6,[6,7],m2,_),
13                                         tache(6,[9,12],m2,_),
14                                         tache(3,[1],m2,_),
15                                         tache(6,[7,8],m2,_)).
16
17 /*Test
18
19 [eclipse 3]: taches(T).
20
21 T = [tache(3, [], m1, _169), tache(8, [], m1, _176), tache(8, [4, 5], m1,
       _183), tache(6, [], m2, _194), tache(3, [1], m2, _201), tache(4, [1, 7],
       m1, _210), tache(8, [3, 5], m1, _221), tache(6, [4], m2, _232), tache(6,
       [6, 7], m2, _241), tache(6, [9, 12], m2, _252), tache(3, [1], m2, _263),
       tache(6, [7, 8], m2, _272)]
22 Yes (0.00s cpu)
23
24 */
25
26 affiche(Taches) :-       dim(Taches,[Dim]),
27                          (for(Indice,1,Dim),param(Taches)
28                                                  do
29                                                          Elem is Taches[Indice
                                                                ],
30                                                          writeln(Elem)
31                                                  ).
32
33
34 /*
35 [eclipse 4]: taches(T),affiche(T).
36 tache(3, [], m1, _235)
37 tache(8, [], m1, _242)
38 tache(8, [4, 5], m1, _249)
39 tache(6, [], m2, _260)
40 tache(3, [1], m2, _267)
41 tache(4, [1, 7], m1, _276)
42 tache(8, [3, 5], m1, _287)
43 tache(6, [4], m2, _298)
44 tache(6, [6, 7], m2, _307)
```



```
45 tache(6, [9, 12], m2, _318)
46 tache(3, [1], m2, _329)
47 tache(6, [7, 8], m2, _338)
48
49 T = [tache(3, [], m1, _235), tache(8, [], m1, _242), tache(8, [4, 5], m1,
       _249), tache(6, [], m2, _260), tache(3, [1], m2, _267), tache(4, [1, 7],
       m1, _276), tache(8, [3, 5], m1, _287), tache(6, [4], m2, _298), tache(6,
       [6, 7], m2, _307), tache(6, [9, 12], m2, _318), tache(3, [1], m2, _329),
       tache(6, [7, 8], m2, _338)]
50 Yes (0.00s cpu)
51
52 */
53
54
55 domaines(Taches,Fin) :- dim(Taches,[Dim]),
56                         (for(Indice,1,Dim),param(Taches,Fin)
57                                                 do
58                                                         tache(Duree,_Nom,
                                                               _Machine,Debut)
                                                               is Taches[Indice
                                                               ],
59                                                         Debut + Duree #=< Fin
                                                               ,
60                                                         Debut #>= 0
61                                                 ).
62
63 /*
64 [eclipse 5]: taches(T),domaines(T,10).
65
66 T = [tache(3, [], m1, _421{0 .. 7}), tache(8, [], m1, _644{0 .. 2}), tache(8,
       [4, 5], m1, _867{0 .. 2}), tache(6, [], m2, _1090{0 .. 4}), tache(3,
       [1], m2, _1313{0 .. 7}), tache(4, [1, 7], m1, _1536{0 .. 6}), tache(8,
       [3, 5], m1, _1759{0 .. 2}), tache(6, [4], m2, _1982{0 .. 4}), tache(6,
       [6, 7], m2, _2205{0 .. 4}), tache(6, [9, 12], m2, _2428{0 .. 4}), tache
       (3, [1], m2, _2651{0 .. 7}), tache(6, [7, 8], m2, _2874{0 .. 4})]
67
68 There are 12 delayed goals. Do you want to see them? (y\n)
69 Yes (0.00s cpu)
70 */
71
72
73 getVarList(Taches,Fin,ListFin):- dim(Taches,[Dim]),
74                                  (for(Indice,1,Dim),fromto([],In,Out,List),
                                       param(Taches)
75                                  do
76                                          Xi is Taches[Indice],
77                                          Xi = tache(_,_,_,Debut),
78                                          Out=[Debut|In]
79                                  ),
80                                  ListFin=[Fin|List].
81
82
83 /*
```



```
84 [eclipse 6]: taches(T), getVarList(T, Fin, L).
85
86 T = [](tache(3, [], m1, _238), tache(8, [], m1, _243), tache(8, [4, 5], m1,
       _248), tache(6, [], m2, _257), tache(3, [1], m2, _262), tache(4, [1, 7],
       m1, _269), tache(8, [3, 5], m1, _278), tache(6, [4], m2, _287), tache(6,
       [6, 7], m2, _294), tache(6, [9, 12], m2, _303), tache(3, [1], m2, _312),
       tache(6, [7, 8], m2, _319))
87 Fin = Fin
88 L = [Fin, _319, _312, _303, _294, _287, _278, _269, _262, _257, _248, _243,
       _238]
89 Yes (0.00s cpu)
90 */
91
92 solve(Taches,Fin) :-    taches(Taches),
93                                          domaines(Taches,Fin),
94                                          precedences(Taches),
95                                          getVarList(Taches,Fin,Liste),
96                                          labeling(Liste),
97                                          affiche(Taches).
98
99 precedences(Taches) :-  dim(Taches,[Dim]),
100                        (for(Indice,1,Dim),param(Taches)
101                        do
102                                Elem is Taches[Indice],
103                                Elem = tache(_D,Noms,_M,Debut),
104                                (foreach(I,Noms),param(Debut,Taches)
105                                do
106                                        tache(Duree2,_N,_M,Debut2) is Taches[
                                               I],
107                                        Debut #>= Debut2+Duree2
108                                )
109                        ).
110
111 /*
112 [eclipse 44]: taches(T), solve(T, Fin).
113 tache(3, [], m1, 0)
114 tache(8, [], m1, 0)
115 tache(8, [4, 5], m1, 6)
116 tache(6, [], m2, 0)
117 tache(3, [1], m2, 3)
118 tache(4, [1, 7], m1, 22)
119 tache(8, [3, 5], m1, 14)
120 tache(6, [4], m2, 6)
121 tache(6, [6, 7], m2, 26)
122 tache(6, [9, 12], m2, 32)
123 tache(3, [1], m2, 3)
124 tache(6, [7, 8], m2, 22)
125
126 T = [](tache(3, [], m1, 0), tache(8, [], m1, 0), tache(8, [4, 5], m1, 6),
       tache(6, [], m2, 0), tache(3, [1], m2, 3), tache(4, [1, 7], m1, 22),
       tache(8, [3, 5], m1, 14), tache(6, [4], m2, 6), tache(6, [6, 7], m2, 26),
        tache(6, [9, 12], m2, 32), tache(3, [1], m2, 3), tache(6, [7, 8], m2,
       22))
```



```
127 Fin = 38
128 */
129
130 conflits(Taches) :- dim(Taches,[Dim]),
131                     (for(Indice,1,Dim),param(Taches,Dim)
132                     do
133                             Elem is Taches[Indice],
134                             I2 is Indice+1,
135                             /*Il faut assurer que les indices soient
                                    differents, sinon on va se retrouver a
                                    comparer deux fois la meme taches*/
136                             (for(I,I2,Dim),param(Taches,Elem)
137                             do
138                                     Elem2 is Taches[I],
139                                     machinesDifferentes(Elem,Elem2)
140                             )
141                     ).
142
143 machinesDifferentes(tache(_,_,M1,_),tache(_,_,M2,_)):- \=(M1,M2),!.
144 machinesDifferentes(tache(Duree,_,M,Debut),tache(Duree2,_,M,Debut2)) :- ((
       Debut #>= Debut2+Duree2) or (Debut+Duree #=< Debut2)).
145
146 solve2(Taches,Fin) :-   taches(Taches),
147                                          domaines(Taches,Fin),
148                                          precedences(Taches),
149                                          conflits(Taches),
150                                          getVarList(Taches,Fin,Liste),
151                                          labeling(Liste),
152                                          affiche(Taches).
153 /*
154 [eclipse 10]: solve2(Taches,Fin).
155 tache(3, [], m1, 0)
156 tache(8, [], m1, 29)
157 tache(8, [4, 5], m1, 9)
158 tache(6, [], m2, 0)
159 tache(3, [1], m2, 6)
160 tache(4, [1, 7], m1, 25)
161 tache(8, [3, 5], m1, 17)
162 tache(6, [4], m2, 12)
163 tache(6, [6, 7], m2, 31)
164 tache(6, [9, 12], m2, 37)
165 tache(3, [1], m2, 9)
166 tache(6, [7, 8], m2, 25)
167
168 Taches = [](tache(3, [], m1, 0), tache(8, [], m1, 29), tache(8, [4, 5], m1,
       9), tache(6, [], m2, 0), tache(3, [1], m2, 6), tache(4, [1, 7], m1, 25),
       tache(8, [3, 5], m1, 17), tache(6, [4], m2, 12), tache(6, [6, 7], m2, 31)
       , tache(6, [9, 12], m2, 37), tache(3, [1], m2, 9), tache(6, [7, 8], m2,
       25))
169 Fin = 43
170 Yes (0.01s cpu, solution 1, maybe more) ? ;
171 tache(3, [], m1, 1)
172 tache(8, [], m1, 29)
```

```
tache(8, [4, 5], m1, 9)
tache(6, [], m2, 0)
tache(3, [1], m2, 6)
tache(4, [1, 7], m1, 25)
tache(8, [3, 5], m1, 17)
tache(6, [4], m2, 12)
tache(6, [6, 7], m2, 31)
tache(6, [9, 12], m2, 37)
tache(3, [1], m2, 9)
tache(6, [7, 8], m2, 25)

Taches = [](tache(3, [], m1, 1), tache(8, [], m1, 29), tache(8, [4, 5], m1,
    9), tache(6, [], m2, 0), tache(3, [1], m2, 6), tache(4, [1, 7], m1, 25),
    tache(8, [3, 5], m1, 17), tache(6, [4], m2, 12), tache(6, [6, 7], m2, 31)
    , tache(6, [9, 12], m2, 37), tache(3, [1], m2, 9), tache(6, [7, 8], m2,
    25))
Fin = 43
Yes (0.01s cpu, solution 2, maybe more) ? ;
tache(3, [], m1, 2)
tache(8, [], m1, 29)
tache(8, [4, 5], m1, 9)
tache(6, [], m2, 0)
tache(3, [1], m2, 6)
tache(4, [1, 7], m1, 25)
tache(8, [3, 5], m1, 17)
tache(6, [4], m2, 12)
tache(6, [6, 7], m2, 31)
tache(6, [9, 12], m2, 37)
tache(3, [1], m2, 9)
tache(6, [7, 8], m2, 25)

Taches = [](tache(3, [], m1, 2), tache(8, [], m1, 29), tache(8, [4, 5], m1,
    9), tache(6, [], m2, 0), tache(3, [1], m2, 6), tache(4, [1, 7], m1, 25),
    tache(8, [3, 5], m1, 17), tache(6, [4], m2, 12), tache(6, [6, 7], m2, 31)
    , tache(6, [9, 12], m2, 37), tache(3, [1], m2, 9), tache(6, [7, 8], m2,
    25))
Fin = 43
Yes (0.01s cpu, solution 3, maybe more) ?
*/

/*
[eclipse 11]: solve2(Taches,42).

No (0.00s cpu)
*/
```

Très bon travail

Rapport Travaux Pratiques : Programmation par Contraintes - TP 4 : **Contraintes Logiques**

Nicolas Desfeux
Aurélien Texier

30 mars 2011

Dans ce T.P., nous allons utiliser la programmation par contraintes pour faire un planning pour organiser une régate, planning qui respecte certaines contraintes.

Question 4.1 Nous définissons ici un prédicat *getData(?TailleEquipes, ?NbEquipes, ?CapaBateaux, ?NbBateaux, ?NbConf)* qui unifie les variables passées en paramètres avec les données du problème.

Listing 1 – "getData"

```
1  getData(TailleEquipes,NbEquipes,CapaBateaux,NbBateaux,NbConf):-
2          TailleEquipes = [](5,5,2,1),
3          NbEquipes = 4,
4          CapaBateaux = [](7,6,5),
5          NbBateaux = 3,
6          NbConf = 3.
7
8  /* Tests
9  [eclipse 7]: getData(TailleEq,NbEq,CapaBat,NbBat,NbConf).
10
11 TailleEq = [](5, 5, 2, 1)
12 NbEq = 4
13 CapaBat = [](7, 6, 5)
14 NbBat = 3
15 NbConf = 3
16 Yes (0.00s cpu)
17 */
```

OK

Question 4.2 Nous définissons ici un prédicat *defineVars(?T,+NbEquipes,+NbConf,+NbBateaux)* qui unifie T au tableau des variables et contraint le domaine des variables.

Listing 2 – "defineVars"

```
1  defineVars(T,NbEquipes,NbConf,NbBateaux):-
```



```
2          dim(T,[NbEquipes,NbConf]),
3          (for(Ind1,1,NbEquipes),param(T,NbBateaux,NbConf)
4          do
5                  (for(Ind2,1,NbConf),param(T,Ind1,NbBateaux)
6                  do
7                          T[Ind1,Ind2] #:: 1..NbBateaux
8                  )
9          ).
10
11 /* Tests
12 [eclipse 8]: getData(TailleEq,NbEq,CapaBat,NbBat,NbConf),defineVars(T,NbEq,
       NbConf,NbBat).
13
14 TailleEq = [](5, 5, 2, 1)
15 NbEq = 4
16 CapaBat = [](7, 6, 5)
17 NbBat = 3
18 NbConf = 3
19 T = []([](_419{1 .. 3}, _488{1 .. 3}, _557{1 .. 3}), [](_628{1 .. 3}, _697{1
       .. 3}, _766{1 .. 3}), [](_837{1 .. 3}, _906{1 .. 3}, _975{1 .. 3}), [](
       _1046{1 .. 3}, _1115{1 .. 3}, _1184{1 .. 3}))
20 Yes (0.00s cpu)
21 */
```

Bien

Question 4.3 Nous définissons ici un prédicat *getVarList(+T, ?L)* qui construit la liste L des variables contenues dans le tableau T. La liste des variables contient les variables de la première colonne suivies de celles de la seconde colonne, etc.

Listing 3 – "getVarList"

```
1  getVarList(T,L):-
2          dim(T,[NbEquipes,NbConf]),
3          (for(Indice1,1,NbConf),fromto([],In,Out,L),param(T,NbEquipes)
4          do
5                  (for(Indice2,1,NbEquipes),fromto([],In2,Out2,L2),param(T,
                       Indice1)
6                  do
7                          Var is T[Indice2,Indice1],
8                          append(In2,[Var],Out2)
9                  ),
10                 append(In,L2,Out)
11         ).
12
13 /* Tests
14 [eclipse 9]:  getData(TailleEq,NbEq,CapaBat,NbBat,NbConf),defineVars(T,NbEq,
       NbConf,NbBat),getVarList(T,L).
15
16 TailleEq = [](5, 5, 2, 1)
17 NbEq = 4
18 CapaBat = [](7, 6, 5)
19 NbBat = 3
20 NbConf = 3
```

Bien


```
21 T = [](_484{1 .. 3}, _553{1 .. 3}, _622{1 .. 3}), [](_693{1 .. 3}, _762{1
       .. 3}, _831{1 .. 3}), [](_902{1 .. 3}, _971{1 .. 3}, _1040{1 .. 3}), [](
       _1111{1 .. 3}, _1180{1 .. 3}, _1249{1 .. 3}))
22 L = [_484{1 .. 3}, _693{1 .. 3}, _902{1 .. 3}, _1111{1 .. 3}, _553{1 .. 3},
       _762{1 .. 3}, _971{1 .. 3}, _1180{1 .. 3}, _622{1 .. 3}, _831{1 .. 3},
       _1040{1 .. 3}, _1249{1 .. 3}]
23 Yes (0.00s cpu)
24 */
```

Question 4.4 Nous définissons ici un prédicat *solve(?T)* qui résoud le problème des régates où seules les contraintes de domaines sont posées.

Listing 4 – "solve1"

```
1  solve1(T) :-
2          getData(_TailleEquipes,NbEquipes,_CapaBateaux,NbBateaux,NbConf),
3          defineVars(T,NbEquipes,NbConf,NbBateaux),
4          getVarList(T,L),
5          labeling(L).
6
7  /* Tests
8  [eclipse 10]: solve1(T).
9
10 T = [](1, 1, 1), [](1, 1, 1), [](1, 1, 1), [](1, 1, 1))
11 Yes (0.00s cpu, solution 1, maybe more) ? ;
12
13 T = [](1, 1, 1), [](1, 1, 1), [](1, 1, 1), [](1, 1, 2))
14 Yes (0.00s cpu, solution 2, maybe more) ? ;
15
16 T = [](1, 1, 1), [](1, 1, 1), [](1, 1, 1), [](1, 1, 3))
17 Yes (0.00s cpu, solution 3, maybe more) ?
18 */
```

Bien

Question 4.5 Nous définissons ici un prédicat *pasMemeBateaux(+T,+NbEquipes,+NbConf)* qui impose qu'une même équipe ne retourne pas deux fois sur le même bateau. On modifie ensuite le prédicat *solve* pour qu'il prenne en compte cette nouvelle contrainte.

Listing 5 – "pasMemeBateaux"

```
1  pasMemeBateaux(T,NbEquipes,NbConf):-
2          dim(T,[NbEquipes,NbConf]),
3          (for(Indice1,1,NbEquipes),param(T,NbConf)
4          do
5                  (for(Indice2,1,NbConf),fromto([],In,Out,L),param(T,Indice1
                       )
6                  do
7                          Bat is T[Indice1,Indice2],
8                          append(In,[Bat],Out)
9                  ),
10                 alldifferent(L)
11         ).
12
```

Bien


```
13 solve2(T) :-
14         getData(_TailleEquipes,NbEquipes,_CapaBateaux,NbBateaux,NbConf),
15         defineVars(T,NbEquipes,NbConf,NbBateaux),
16         pasMemeBateaux(T,NbEquipes,NbConf),
17         getVarList(T,L),
18         labeling(L).
19
20 /* Tests
21 [eclipse 11]: solve2(T).
22
23 T = [](1, 2, 3), [](1, 2, 3), [](1, 2, 3), [](1, 2, 3))
24 Yes (0.00s cpu, solution 1, maybe more) ? ;
25
26 T = [](1, 2, 3), [](1, 2, 3), [](1, 2, 3), [](1, 3, 2))
27 Yes (0.00s cpu, solution 2, maybe more) ? ;
28
29 T = [](1, 2, 3), [](1, 2, 3), [](1, 3, 2), [](1, 2, 3))
30 Yes (0.00s cpu, solution 3, maybe more) ?
31 */
```

OK

Question 4.6 Nous définissons ici un prédicat *pasMemePartenaires(+T,+NbEquipes,+NbConf)* qui impose qu'une même équipe ne se retrouve pas deux fois avec la même équipe. On modifie une nouvelle fois le prédicat *solve* pour qu'il prenne en compte cette nouvelle contrainte.

Listing 6 – "pasMemePartenaires"

```
1  pasMemePartenaires(T,NbEquipes,NbConf):-
2          dim(T,[NbEquipes,NbConf]),
3          (for(Equipe1,1,NbEquipes),param(T,NbConf,NbEquipes)
4          do
5                  Indice is Equipe1+1,
6                  (for(Equipe2,Indice,NbEquipes), param(T,Equipe1,NbConf)
7                  do
8
9                          (for(Conf,1,NbConf),param(T,Equipe1,Equipe2),fromto
                               (0,In,Out,Tot)
10                         do
11                                 Bateau1 is T[Equipe1,Conf],
12                                 Bateau2 is T[Equipe2,Conf],
13                                 #=(Bateau1,Bateau2,Ans),
14                                 Out #= In + Ans
15                         ),
16                         Tot #=< 1
17                 )
18         ).
19
20 solve3(T) :-
21         getData(_TailleEquipes,NbEquipes,_CapaBateaux,NbBateaux,NbConf),
22         defineVars(T,NbEquipes,NbConf,NbBateaux),
23         pasMemeBateaux(T,NbEquipes,NbConf),
24         pasMemePartenaires(T,NbEquipes,NbConf),
25         getVarList(T,L),
26         labeling(L).
```

Bien

```
/* Tests
[eclipse 12]: solve3(T).

T = [](1, 2, 3), [](1, 3, 2), [](2, 1, 3), [](2, 3, 1))
Yes (0.00s cpu, solution 1, maybe more) ? ;

T = [](1, 2, 3), [](1, 3, 2), [](2, 3, 1), [](2, 1, 3))
Yes (0.00s cpu, solution 2, maybe more) ? ;

T = [](1, 3, 2), [](1, 2, 3), [](2, 1, 3), [](2, 3, 1))
Yes (0.00s cpu, solution 3, maybe more) ?
*/
```

Question 4.7 Nous définissons ici un prédicat *capaBateaux(+T,+TailleEquipes,+NbEquipes,+CapaBateaux,+NbB* qui vérifie que les capacités des bateaux sont respectées lors de chaque confrontation. On modifie une nouvelle fois le prédicat *solve* pour qu'il prenne en compte cette nouvelle contrainte.

Listing 7 – "capaBateaux"

```
capaBateaux(T,TailleEquipes,NbEquipes,CapaBateaux,NbBateaux,NbConf):-
        dim(T,[NbEquipes,NbConf]),
        (for(Bateau,1,NbBateaux), param(T,NbEquipes,NbConf,CapaBateaux,
            TailleEquipes)
        do
                (for(Conf,1,NbConf), param(T,NbEquipes,Bateau,CapaBateaux,
                    TailleEquipes)
                do
                        (for(Equipe,1,NbEquipes),param(T,Bateau,Conf,
                            TailleEquipes),fromto(0,In,Out,Total)
                        do
                                BateauI is T[Equipe,Conf],
                                #=(Bateau,BateauI,Cond),
                                Inc #= TailleEquipes[Equipe] * Cond,
                                Out #= In + Inc
                        ),
                        Capacite is CapaBateaux[Bateau],
                        Total #=< Capacite
                )
        ).

solve4(T) :-
        getData(TailleEquipes,NbEquipes,CapaBateaux,NbBateaux,NbConf),
        defineVars(T,NbEquipes,NbConf,NbBateaux),
        pasMemeBateaux(T,NbEquipes,NbConf),
        pasMemePartenaires(T,NbEquipes,NbConf),
        capaBateaux(T,TailleEquipes,NbEquipes,CapaBateaux,NbBateaux,NbConf),
        getVarList(T,L),
        labeling(L).

/* Tests
[eclipse 13]: solve4(T).
```

Bien


```
T = [](](1, 2, 3), [](2, 3, 1), [](3, 1, 2), [](3, 2, 1))
Yes (0.01s cpu, solution 1, maybe more) ? ;

T = [](](1, 3, 2), [](2, 1, 3), [](3, 2, 1), [](3, 1, 2))
Yes (0.01s cpu, solution 2, maybe more) ? ;

T = [](](1, 2, 3), [](3, 1, 2), [](2, 3, 1), [](1, 3, 2))
Yes (0.01s cpu, solution 3, maybe more) ?
*/
```

Question 4.8 On passe ici à un problème de taille réelle. On dispose dorénavant de 13 voiliers et de 29 équipes qui doivent effectuer la régate comportant 7 confrontations.
Le temps d'exécution étant relativement long, il nous a été proposé d'améliorer le labeling. Pour cela, nous avons mélanger la liste des varaibles obtenu après *getVarList*, en alternant simplement les grosses et les petites équipes. Le gain sur le temps d'exécution est relativement important, puisqu'il est quasiment de 10 !

Listing 8 – "Problème de taille réelle et labeling"

```
getData2(TailleEquipes,NbEquipes,CapaBateaux,NbBateaux,NbConf):-
        TailleEquipes =
            [](7,6,5,5,5,4,4,4,4,4,4,4,3,3,2,2,2,2,2,2,2,2,2,2,2,2),
        NbEquipes = 29,
        CapaBateaux = [](10,10,9,8,8,8,8,8,8,7,6,4,4),
        NbBateaux = 13,
        NbConf = 7.

solve5(T):-
        getData2(TailleEq,NbEq,CapaBat,NbBat,NbConf),
        defineVars(T,NbEq,NbConf,NbBat),
        pasMemeBateaux(T,NbEq,NbConf),
        pasMemePartenaires(T,NbEq,NbConf),
        capaBateaux(T,TailleEq,NbEq,CapaBat,NbBat,NbConf),
        getVarList(T,L),
        labeling(L).

/* Tests
[eclipse 14]: solve5(T).

T = [](](1, 2, 3, 4, 5, 6, 7), [](2, 1, 4, 3, 6, 5, 8), [](3, 4, 1, 2, 7, 8,
     5), [](4, 3, 1, 5, 2, 7, 6), [](5, 6, 2, 1, 3, 4, 9), [](2, 3, 5, 1, 4,
     9, 10), [](3, 1, 2, 6, 4, 10, 11), [](6, 5, 7, 2, 1, 3, 4), [](6, 7, 5,
     8, 2, 1, 3), [](7, 5, 6, 8, 3, 2, 1), [](7, 8, 9, 6, 1, 11, 2), [](8, 7,
     6, 9, 10, 12, 2), [](8, 9, 7, 10, 11, 1, 12), [](1, 4, 8, 3, 9, 7, 10),
     [](4, 2, 8, 10, 7, 3, 9), [](5, 8, 3, 11, 6, 9, 1), [](9, 6, 4, 5, 8, 11,
     13), [](9, 8, 10, 12, 13, 2, 11), [](9, 10, 8, 11, 12, 13, 2), [](9, 11,
     12, 13, 1, 10, 3), [](10, 9, 11, 7, 12, 3, 13), [](10, 11, 9, 12, 8, 1,
     4), [](10, 12, 13, 11, 9, 2, 3), [](11, 9, 10, 13, 8, 4, 5), [](11, 10,
     12, 9, 13, 5, 1), [](11, 12, 9, 7, 10, 13, 8), [](12, 10, 13, 7, 11, 9,
     4), [](12, 13, 11, 9, 8, 2, 6), [](13, 11, 10, 7, 9, 8, 1))
Yes (55.23s cpu, solution 1, maybe more) ?
*/
```



```
% Amelioration du labeling pour gagner du temps

getLast([A],A,[]).
getLast([A|R],B,[A|L]):- getLast(R,B,L).
```

= append(L,[B],R)

```
debutfin([A|R],A,L,B) :- getLast(R,B,L).
debutfin([A,B,C],A,[A,C,B],C).

melangListe([],[]).
melangListe([A,B,C],[A,C,B]):-!.
melangListe(L,[A,B|L2]) :- debutfin(L,A,L1,B),melangListe(L1,L2).

getVarList2(T,L):-
        dim(T,[NbEquipes,NbConf]),
        (for(Indice1,1,NbConf),fromto([],In,Out,L),param(T,NbEquipes)
        do
                (for(Indice2,1,NbEquipes),fromto([],In2,Out2,L2),param(T,
                    Indice1)
                do
                        Var is T[Indice2,Indice1],
                        append(In2,[Var],Out2)
                ),
                melangListe(L2,L3),
                append(In,L3,Out)
        ).

solve6(T):-
        getData2(TailleEq,NbEq,CapaBat,NbBat,NbConf),
        defineVars(T,NbEq,NbConf,NbBat),
        pasMemeBateaux(T,NbEq,NbConf),
        pasMemePartenaires(T,NbEq,NbConf),
        capaBateaux(T,TailleEq,NbEq,CapaBat,NbBat,NbConf),
        getVarList2(T,L),
        labeling(L).

/* Tests
[eclipse 15]: solve6(T).

T = [](](1, 2, 3, 4, 5, 6, 7), [](2, 1, 4, 3, 6, 5, 8), [](3, 4, 1, 2, 8, 7,
     9), [](4, 3, 5, 1, 2, 9, 10), [](5, 6, 2, 7, 1, 3, 4), [](6, 5, 7, 2, 1,
     4, 3), [](6, 7, 8, 5, 2, 1, 11), [](7, 5, 6, 8, 9, 1, 2), [](8, 9, 7,
     10, 4, 11, 5), [](8, 10, 9, 6, 11, 3, 2), [](9, 8, 12, 13, 7, 2, 6),
     [](10, 9, 8, 11, 13, 12, 1), [](12, 13, 11, 9, 10, 8, 1), [](11, 10, 13,
     3, 8, 9, 4), [](11, 12, 10, 7, 3, 13, 9), [](13, 11, 10, 9, 6, 1, 5),
     [](13, 8, 11, 12, 4, 10, 3), [](10, 11, 9, 8, 12, 2, 3), [](9, 11, 6, 12,
     10, 5, 13), [](9, 7, 10, 11, 12, 8, 2), [](7, 8, 9, 10, 3, 4, 1), [](7,
     6, 5, 9, 11, 2, 8), [](5, 7, 1, 8, 3, 10, 13), [](4, 1, 6, 5, 3, 2, 12),
     [](3, 2, 4, 1, 7, 11, 12), [](3, 1, 2, 6, 9, 10, 11), [](2, 4, 3, 1, 9,
     8, 6), [](2, 3, 1, 6, 7, 4, 5), [](1, 3, 2, 5, 4, 7, 6))
Yes (6.98s cpu, solution 1, maybe more) ? ;
```



```
T = [](](1, 2, 3, 4, 5, 6, 7), [](2, 1, 4, 3, 6, 5, 8), [](3, 4, 1, 2, 8, 7,
     9), [](4, 3, 5, 1, 2, 9, 10), [](5, 6, 2, 7, 1, 3, 4), [](6, 5, 7, 2, 1,
     4, 11), [](6, 7, 8, 5, 2, 1, 3), [](7, 5, 6, 8, 9, 1, 2), [](8, 9, 7,
     10, 4, 11, 5), [](8, 10, 9, 6, 11, 3, 2), [](9, 8, 12, 13, 7, 2, 6),
     [](10, 9, 8, 11, 13, 12, 1), [](12, 13, 11, 9, 10, 8, 1), [](11, 10, 13,
     3, 8, 9, 4), [](11, 12, 10, 7, 3, 13, 9), [](13, 11, 10, 9, 6, 1, 5),
     [](13, 8, 11, 12, 4, 10, 3), [](10, 11, 9, 8, 12, 2, 3), [](9, 11, 6, 12,
     10, 5, 13), [](9, 7, 10, 11, 12, 8, 2), [](7, 8, 9, 10, 3, 4, 1), [](7,
     6, 5, 9, 11, 2, 8), [](5, 7, 1, 8, 3, 10, 13), [](4, 1, 6, 5, 3, 2, 12),
     [](3, 2, 4, 1, 7, 11, 12), [](3, 1, 2, 6, 9, 10, 11), [](2, 4, 3, 1, 9,
     8, 6), [](2, 3, 1, 6, 7, 4, 5), [](1, 3, 2, 5, 4, 7, 6))
Yes (7.31s cpu, solution 2, maybe more) ?
*/
```

1 Code Complet, avec l'ensemble des tests

Listing 9 – "TP4"

```
:-lib(ic).

% Q4.1
getData(TailleEquipes,NbEquipes,CapaBateaux,NbBateaux,NbConf):-
        TailleEquipes = [](5,5,2,1),
        NbEquipes = 4,
        CapaBateaux = [](7,6,5),
        NbBateaux = 3,
        NbConf = 3.

/* Tests
[eclipse 7]: getData(TailleEq,NbEq,CapaBat,NbBat,NbConf).

TailleEq = [](5, 5, 2, 1)
NbEq = 4
CapaBat = [](7, 6, 5)
NbBat = 3
NbConf = 3
Yes (0.00s cpu)
*/

% Q4.2
defineVars(T,NbEquipes,NbConf,NbBateaux):-
        dim(T,[NbEquipes,NbConf]),
        ( for(Ind1,1,NbEquipes),param(T,NbBateaux,NbConf)
        do
                ( for(Ind2,1,NbConf),param(T,Ind1,NbBateaux)
                do
                        T[Ind1,Ind2] #:: 1..NbBateaux
                )
        ).

/* Tests
[eclipse 8]: getData(TailleEq,NbEq,CapaBat,NbBat,NbConf),defineVars(T,NbEq,
    NbConf,NbBat).

TailleEq = [](5, 5, 2, 1)
NbEq = 4
CapaBat = [](7, 6, 5)
NbBat = 3
NbConf = 3
T = [](](_419{1 .. 3}, _488{1 .. 3}, _557{1 .. 3}), [](_628{1 .. 3}, _697{1
    .. 3}, _766{1 .. 3}), [](_837{1 .. 3}, _906{1 .. 3}, _975{1 .. 3}), [](
    _1046{1 .. 3}, _1115{1 .. 3}, _1184{1 .. 3}))
Yes (0.00s cpu)
*/

% Q4.3
getVarList(T,L):-
```



```
        dim(T,[NbEquipes,NbConf]),
        ( for(Indice1,1,NbConf),fromto([],In,Out,L),param(T,NbEquipes)
        do
                ( for(Indice2,1,NbEquipes),fromto([],In2,Out2,L2),param(T,
                    Indice1)
                do
                        Var is T[Indice2,Indice1],
                        append(In2,[Var],Out2)
                ),
                append(In,L2,Out)
        ).

/* Tests
[eclipse 9]: getData(TailleEq,NbEq,CapaBat,NbBat,NbConf),defineVars(T,NbEq,
    NbConf,NbBat),getVarList(T,L).

TailleEq = [](5, 5, 2, 1)
NbEq = 4
CapaBat = [](7, 6, 5)
NbBat = 3
NbConf = 3
T = [](](_484{1 .. 3}, _553{1 .. 3}, _622{1 .. 3}), [](_693{1 .. 3}, _762{1
    .. 3}, _831{1 .. 3}), [](_902{1 .. 3}, _971{1 .. 3}, _1040{1 .. 3}), [](
    _1111{1 .. 3}, _1180{1 .. 3}, _1249{1 .. 3}))
L = [_484{1 .. 3}, _693{1 .. 3}, _902{1 .. 3}, _1111{1 .. 3}, _553{1 .. 3},
    _762{1 .. 3}, _971{1 .. 3}, _1180{1 .. 3}, _622{1 .. 3}, _831{1 .. 3},
    _1040{1 .. 3}, _1249{1 .. 3}]
Yes (0.00s cpu)
*/

% Q4.4

solve1(T) :-
        getData(_TailleEquipes,NbEquipes,_CapaBateaux,NbBateaux,NbConf),
        defineVars(T,NbEquipes,NbConf,NbBateaux),
        getVarList(T,L),
        labeling(L).

/* Tests
[eclipse 10]: solve1(T).

T = [](](1, 1, 1), [](1, 1, 1), [](1, 1, 1), [](1, 1, 1))
Yes (0.00s cpu, solution 1, maybe more) ? ;

T = [](](1, 1, 1), [](1, 1, 1), [](1, 1, 1), [](1, 1, 2))
Yes (0.00s cpu, solution 2, maybe more) ? ;

T = [](](1, 1, 1), [](1, 1, 1), [](1, 1, 1), [](1, 1, 3))
Yes (0.00s cpu, solution 3, maybe more) ?
*/

% Q4.5
pasMemeBateaux(T,NbEquipes,NbConf):-
```



```
        dim(T,[NbEquipes,NbConf]),
        ( for(Indice1,1,NbConf),param(T,NbConf)
        do
                ( for(Indice2,1,NbConf), fromto([],In,Out,L), param(T,Indice1
                    )
                do
                        Bat is T[Indice1,Indice2],
                        append(In,[Bat],Out)
                ),
                alldifferent(L)
        ).

solve2(T) :-
        getData(_TailleEquipes,NbEquipes,_CapaBateaux,NbBateaux,NbConf),
        defineVars(T,NbEquipes,NbConf,NbBateaux),
        pasMemeBateaux(T,NbEquipes,NbConf),
        getVarList(T,L),
        labeling(L).

/* Tests
[eclipse 11]: solve2(T).

T = [](](1, 2, 3), [](1, 2, 3), [](1, 2, 3), [](1, 2, 3))
Yes (0.00s cpu, solution 1, maybe more) ? ;

T = [](](1, 2, 3), [](1, 2, 3), [](1, 2, 3), [](1, 3, 2))
Yes (0.00s cpu, solution 2, maybe more) ? ;

T = [](](1, 2, 3), [](1, 2, 3), [](1, 3, 2), [](1, 2, 3))
Yes (0.00s cpu, solution 3, maybe more) ?
*/

% Q4.6
pasMemePartenaires(T,NbEquipes,NbConf):-
        dim(T,[NbEquipes,NbConf]),
        ( for(Equipe1,1,NbEquipes),param(T,NbConf,NbEquipes)
        do
                Indice is Equipe1+1,
                ( for(Equipe2,Indice,NbEquipes), param(T,Equipe1,NbConf)
                do

                        ( for(Conf,1,NbConf),param(T,Equipe1,Equipe2),fromto
                            (0,In,Out,Tot)
                        do
                                Bateau1 is T[Equipe1,Conf],
                                Bateau2 is T[Equipe2,Conf],
                                #=(Bateau1,Bateau2,Ans),
                                Out #= In + Ans
                        ),
                        Tot #=< 1
                )
        ).

```



```
solve3(T) :-
        getData(_TailleEquipes,NbEquipes,_CapaBateaux,NbBateaux,NbConf),
        defineVars(T,NbEquipes,NbConf,NbBateaux),
        pasMemeBateaux(T,NbEquipes,NbConf),
        pasMemePartenaires(T,NbEquipes,NbConf),
        getVarList(T,L),
        labeling(L).

/* Tests
[eclipse 12]: solve3(T).

T = [](](1, 2, 3), [](1, 3, 2), [](2, 1, 3), [](2, 3, 1))
Yes (0.00s cpu, solution 1, maybe more) ? ;

T = [](](1, 2, 3), [](1, 3, 2), [](2, 3, 1), [](2, 1, 3))
Yes (0.00s cpu, solution 2, maybe more) ? ;

T = [](](1, 3, 2), [](1, 2, 3), [](2, 1, 3), [](2, 3, 1))
Yes (0.00s cpu, solution 3, maybe more) ?
*/

% Q4.7
capaBateaux(T,TailleEquipes,NbEquipes,CapaBateaux,NbBateaux,NbConf):-
        dim(T,[NbEquipes,NbConf]),
        ( for(Bateau,1,NbBateaux), param(T,NbEquipes,NbConf,CapaBateaux,
            TailleEquipes)
        do
                ( for(Conf,1,NbConf), param(T,NbEquipes,Bateau,CapaBateaux,
                    TailleEquipes)
                do
                        ( for(Equipe,1,NbEquipes),param(T,Bateau,Conf,
                            TailleEquipes),fromto(0,In,Out,Total)
                        do
                                Bateau1 is T[Equipe,Conf],
                                #=(Bateau,Bateau1,Cond),
                                Inc #= TailleEquipes[Equipe] * Cond,
                                Out #= In + Inc
                        ),
                        Capacite is CapaBateaux[Bateau],
                        Total #=< Capacite
                )
        ).

solve4(T) :-
        getData(TailleEquipes,NbEquipes,CapaBateaux,NbBateaux,NbConf),
        defineVars(T,NbEquipes,NbConf,NbBateaux),
        pasMemeBateaux(T,NbEquipes,NbConf),
        pasMemePartenaires(T,NbEquipes,NbConf),
        capaBateaux(T,TailleEquipes,NbEquipes,CapaBateaux,NbBateaux,NbConf),
        getVarList(T,L),
        labeling(L).

/* Tests
```

```
[eclipse 13]: solve4(T).

T = [](1, 2, 3), [](2, 3, 1), [](3, 1, 2), [](3, 2, 1))
Yes (0.01s cpu, solution 1, maybe more) ? ;

T = [](1, 3, 2), [](2, 1, 3), [](3, 2, 1), [](3, 1, 2))
Yes (0.01s cpu, solution 2, maybe more) ? ;

T = [](1, 2, 3), [](3, 1, 2), [](2, 3, 1), [](1, 3, 2))
Yes (0.01s cpu, solution 3, maybe more) ?
*/

% Q4.8

getData2(TailleEquipes,NbEquipes,CapaBateaux,NbBateaux,NbConf):-
        TailleEquipes =
            [](7,6,5,5,5,4,4,4,4,4,4,4,4,3,3,2,2,2,2,2,2,2,2,2,2,2,2,2),
        NbEquipes = 29,
        CapaBateaux = [](10,10,9,8,8,8,8,8,8,7,6,4,4),
        NbBateaux = 13,
        NbConf = 7.

solve5(T):-
        getData2(TailleEq,NbEq,CapaBat,NbBat,NbConf),
        defineVars(T,NbEq,NbConf,NbBat),
        pasMemeBateaux(T,NbEq,NbConf),
        pasMemePartenaires(T,NbEq,NbConf),
        capaBateaux(T,TailleEq,NbEq,CapaBat,NbBat,NbConf),
        getVarList(T,L),
        labeling(L).

/* Tests
[eclipse 14]: solve5(T).

T = [](](1, 2, 3, 4, 5, 6, 7), [](2, 1, 4, 3, 6, 5, 8), [](3, 4, 1, 2, 7, 8,
     5), [](4, 3, 1, 5, 2, 7, 6), [](5, 6, 2, 1, 3, 4, 9), [](2, 3, 5, 1, 4,
    9, 10), [](3, 1, 2, 6, 4, 10, 11), [](6, 5, 7, 2, 1, 3, 4), [](6, 7, 5,
    8, 2, 1, 3), [](7, 5, 6, 8, 3, 2, 1), [](7, 8, 9, 6, 1, 11, 2), [](8, 7,
    6, 9, 10, 12, 2), [](8, 9, 7, 10, 11, 1, 12), [](1, 4, 8, 3, 9, 7, 10),
    [](4, 2, 8, 10, 7, 3, 9), [](5, 8, 3, 11, 6, 9, 1), [](9, 6, 4, 5, 8, 11,
     13), [](9, 8, 10, 12, 13, 2, 11), [](9, 10, 8, 11, 12, 13, 2), [](9, 11,
     12, 13, 1, 10, 3), [](10, 9, 11, 7, 12, 3, 13), [](10, 11, 9, 12, 8, 1,
    4), [](10, 12, 13, 11, 9, 2, 3), [](11, 9, 10, 13, 8, 4, 5), [](11, 10,
    12, 9, 13, 5, 1), [](11, 12, 9, 7, 10, 13, 8), [](12, 10, 13, 7, 11, 9,
    4), [](12, 13, 11, 9, 8, 2, 6), [](13, 11, 10, 7, 9, 8, 1))
Yes (55.23s cpu, solution 1, maybe more) ?
*/

% Amelioration du labeling pour gagner du temps

getLast([A],A,[]).
getLast([A|R],B,[A|L]):- getLast(R,B,L).
```



```
debutfin([A|R],A,L,B) :- getLast(R,B,L).
debutfin([A,B,C],A,[A,C,B],C).

melangListe([],[]).
melangListe([A,B,C],[A,C,B]):-!.
melangListe(L,[A,B|L2]) :- debutfin(L,A,L1,B),melangListe(L1,L2).

getVarList2(T,L):-
        dim(T,[NbEquipes,NbConf]),
        ( for(Indice1,1,NbConf),fromto([],In,Out,L),param(T,NbEquipes)
        do
                ( for(Indice2,1,NbEquipes),fromto([],In2,Out2,L2),param(T,
                    Indice1)
                do
                        Var is T[Indice2,Indice1],
                        append(In2,[Var],Out2)
                ),
                melangListe(L2,L3),
                append(In,L3,Out)
        ).

solve6(T):-
        getData2(TailleEq,NbEq,CapaBat,NbBat,NbConf),
        defineVars(T,NbEq,NbConf,NbBat),
        pasMemeBateaux(T,NbEq,NbConf),
        pasMemePartenaires(T,NbEq,NbConf),
        capaBateaux(T,TailleEq,NbEq,CapaBat,NbBat,NbConf),
        getVarList2(T,L),
        labeling(L).

/* Tests
[eclipse 15]: solve6(T).

T = [](](1, 2, 3, 4, 5, 6, 7), [](2, 1, 4, 3, 6, 5, 8), [](3, 4, 1, 2, 8, 7,
    9), [](4, 3, 5, 1, 2, 9, 10), [](5, 6, 2, 7, 1, 3, 4), [](6, 5, 7, 2, 1,
    4, 3), [](6, 7, 8, 5, 2, 1, 11), [](7, 5, 6, 8, 9, 1, 2), [](8, 9, 7,
    10, 4, 11, 5), [](8, 10, 9, 6, 11, 3, 2), [](9, 8, 12, 13, 7, 2, 6),
    [](10, 9, 8, 11, 13, 12, 1), [](12, 13, 11, 9, 10, 8, 1), [](11, 10, 13,
    3, 8, 9, 4), [](11, 12, 10, 7, 3, 13, 9), [](13, 11, 10, 9, 6, 1, 5),
    [](13, 8, 11, 12, 4, 10, 3), [](10, 11, 9, 8, 12, 2, 3), [](9, 11, 6, 12,
     10, 5, 13), [](9, 7, 10, 11, 12, 8, 2), [](7, 8, 9, 10, 3, 4, 1), [](7,
    6, 5, 9, 11, 2, 8), [](5, 7, 1, 8, 3, 10, 13), [](4, 1, 6, 5, 3, 2, 12),
    [](3, 2, 4, 1, 7, 11, 12), [](3, 1, 2, 6, 9, 10, 11), [](2, 4, 3, 1, 9,
    8, 6), [](2, 3, 1, 6, 7, 4, 5), [](1, 3, 2, 5, 4, 7, 6))
Yes (6.98s cpu, solution 1, maybe more) ? ;

T = [](](1, 2, 3, 4, 5, 6, 7), [](2, 1, 4, 3, 6, 5, 8), [](3, 4, 1, 2, 8, 7,
    9), [](4, 3, 5, 1, 2, 9, 10), [](5, 6, 2, 7, 1, 3, 4), [](6, 5, 7, 2, 1,
    4, 11), [](6, 7, 8, 5, 2, 1, 3), [](7, 5, 6, 8, 9, 1, 2), [](8, 9, 7,
    10, 4, 11, 5), [](8, 10, 9, 6, 11, 3, 2), [](9, 8, 12, 13, 7, 2, 6),
    [](10, 9, 8, 11, 13, 12, 1), [](12, 13, 11, 9, 10, 8, 1), [](11, 10, 13,
    3, 8, 9, 4), [](11, 12, 10, 7, 3, 13, 9), [](13, 11, 10, 9, 6, 1, 5),
    [](13, 8, 11, 12, 4, 10, 3), [](10, 11, 9, 8, 12, 2, 3), [](9, 11, 6, 12,
     10, 5, 13), [](9, 7, 10, 11, 12, 8, 2), [](7, 8, 9, 10, 3, 4, 1), [](7,
    6, 5, 9, 11, 2, 8), [](5, 7, 1, 8, 3, 10, 13), [](4, 1, 6, 5, 3, 2, 12),
    [](3, 2, 4, 1, 7, 11, 12), [](3, 1, 2, 6, 9, 10, 11), [](2, 4, 3, 1, 9,
    8, 6), [](2, 3, 1, 6, 7, 4, 5), [](1, 3, 2, 5, 4, 7, 6))
Yes (7.31s cpu, solution 2, maybe more) ?
*/
```

Bon travail, votre code aurait toutefois été plus élégant en définissant des prédicats génériques de manipulation de vecteurs (produit scalaire, multiplication) et en utilisant ces prédicats pour poser les contraintes.
NB : l'exercice intermédiaire sur le branch & bound est à reprendre.

Rapport Travaux Pratiques : Programmation par Contraintes - TP 5 : **Contraindre puis chercher**

Nicolas Desfeux
Aurélien Texier

4 avril 2011

Dans ce TP, nous allons chercher à résoudre un problème de gestion de production. Pour cela, nous allons utiliser la programmation par contraintes.

1 Le problème

Le problème que nous allons résoudre concerne une unité de production pour différentes gammes de téléphones mobiles.
L'objectif est de trouver quels fabrications il faut lancer pour obtenir un bénéfice maximum.
Voici les données qui nous sont fournies :
Pour chaque type gamme, nous avons :

- Le nombre de techniciens de la gamme,
- La quantité de téléphone que l'on peut produire par jour,
- Le bénéfice obtenu.

Bonne intro

2 Modéliser et contraindre

Question 5.1 On définit ici différents prédicats qui créent l'ensembles des données et des variables du problème.

Listing 1 – "Définition des 3 vecteurs de valeurs et du vecteur de variables"

```
1  getData(NbTechniciens,NbSortes,Techniciens,Quantite,Benefice) :-
2          NbTechniciens = 22,
3          NbSortes = 9,
4          Techniciens = [](5,7,2,6,9,3,7,5,3),
5          Quantite = [](140,130,60,95,70,85,100,30,45),
6          Benefice = [](4,5,8,5,5,6,4,7,10,11).
7
8  defineVars(Fabriquer,NbSortes):-
```



```
9          dim(Fabriquer,[NbSortes]),
10         (for(I,1,NbSortes),param(Fabriquer)
11         do
12                 Fabriquer[I] #:: 0..1
13         ).
14
15 getVarList(Fabriquer,NbSortes,L) :-
16         (for(Ind,1,NbSortes),fromto([],In,Out,L),param(Fabriquer)
17                 do
18                         Var #= Fabriquer[Ind],
19                         append(In,[Var],Out)
20         ).
```

OK

OK

Question 5.2 Le code suivant permet de définir les prédicats permettant d'exprimer :

- Le nombre toal d'ouvriers nécéssaire,
- Le vecteur de bénéfice total par sorte de téléphones,
- Le profit total.

Listing 2 – "Différents prédicats utiles à la résolution du problème"

```
1  nbOuvriersNecessaires(Fabriquer,Techniciens,NbSortes,NbOuvriersTotal) :-
2          (for(J,1,NbSortes),fromto(0,In,Out,NbOuvriersTotal),param(Fabriquer,
               Techniciens)
3                  do
4                          Var is Fabriquer[J],
5                          NbTech is Techniciens[J],
6                          Out #= In + NbTech * Var
7          ).
8
9  vectBeneficeTotal(Fabriquer,Benefice,Quantite,NbSortes,VectBenefTotal) :-
10         (for(K,1,NbSortes),fromto([],In,Out,VectBenefTotal),param(Fabriquer,
               Benefice,Quantite)
11                 do
12                         Var is Fabriquer[K],
13                         Benef is Benefice[K],
14                         Quan is Quantite[K],
15                         Elem #= Var * Benef * Quan,
16                         append(In,[Elem],Out)
17         ).
18
19 profitTotal(Fabriquer,Benefice,Quantite,NbSortes,ProfitTotal) :-
20         vectBeneficeTotal(Fabriquer,Benefice,Quantite,NbSortes,VectBenefTotal
               ),
21         ProfitTotal #= sum(VectBenefTotal).
```

OK

OK

OK

Question 5.3 On a défini le prédicat pose_contraintes comme une succesion de prédicats. Contrairement aux T.P. précédents, nous n'avons pas créé de prédicat solve, qui aurait appelé (entre autre), pose_contrainte.

Listing 3 – "Prédicat **pose_contraintes**"

```
1  pose_contraintes(Fabriquer,NbTechniciensTotal,Profit) :-
```

En fait c'est ce prédicat que vous auriez pu appelé 'solve'...


```
2          getData(NbTech,NbSor,Tech,Quan,Bene),
3          defineVars(Fabriquer,NbSor),
4          nbOuvriersNecessaires(Fabriquer,Tech,NbSor,NbTechniciensTotal),
5          NbTechniciensTotal #=< NbTech,
6          getVarList(Fabriquer,NbSor,L),
7          profitTotal(Fabriquer,Bene,Quan,NbSor,Profit),
8          labeling(L).
```

OK

3 Optimiser

3.1 Branch and bound dans ECLiPSe

Question 5.4 Il faut toujours faire un labeling dans le but, car il faut que les variables aient ete instanciees.

plus que ça : il faut faire un labeling sur TOUTES les variables du problème pour que la recherche de l'optimal se fasse uniquement sur des solutions

Listing 4 – Prédicat **test** avec minimize

```
1  test(X,Y,Z,W) :-
2          [X,Y,Z,W] #:: [0..10],
3          X #= Z+Y+2*W,
4          X#\= Z+Y+W.
5
6  /* Test
7
8  [eclipse 53]: minimize(test(X,Y,Z,W),X).
9  bb_min: search did not instantiate cost variable
10 Aborting execution ...
11 Abort
12
13 Avec le labeling sur [X,Y,Z,W], cela fonctionne, Prolog trouve les reponses
14
15 [eclipse 57]: test(X,Y,Z,W),labeling([X,Y,Z,W]).
16
17 X = 2                                    Il vous était demandé de tester avec labeling([X]) puis
18 Y = 0                                    de comparer avec labeling([X,Y,Z,W])...
19 Z = 0
20 W = 1
21 Yes (0.00s cpu, solution 1, maybe more) ? ;
22
23 X = 3
24 Y = 0
25 Z = 1
26 W = 1
27 Yes (0.00s cpu, solution 2, maybe more) ? ;
28 */
```

3.2 Application à notre problème

Question 5.5 Pour trouver le profit maximum, on crée une variable que l'on cherche à minimaliser, et l'on pose comme contrainte qu'elle soit égale à l'opposé du profit. On obtient ainsi le profit maximum.

Listing 5 – "Prédicat **pose_contrainte** avec maximisation du profit"

```
1  /* Test
2
3  [eclipse 71]: P2#=P*(-1), minimize(pose_contraintes(F,N,P),P2).
4  Found a solution with cost 0
5  Found a solution with cost -495
6  Found a solution with cost -795
7  Found a solution with cost -1195
8  Found a solution with cost -1495
9  Found a solution with cost -1535
10 Found a solution with cost -1835
11 Found a solution with cost -1955
12 Found a solution with cost -1970
13 Found a solution with cost -2010
14 Found a solution with cost -2015
15 Found a solution with cost -2315
16 Found a solution with cost -2490
17 Found a solution with cost -2665
18 Found no solution with cost -1.0Inf .. -2666
19
20 P2 = -2665
21 P = 2665
22 F = [](0, 1, 1, 0, 0, 1, 1, 0, 1)
23 N = 22
24 Yes (0.00s cpu)
25 */
```

Bien

Nous avons choisi d'appeler le minimize lors de l'appel de la fonction principale.

Question 5.6 Pour cette question, il nous est demandé de changer la politique de l'entreprise. On choisit de créer un seuil pour le profit, et minimiser le nombre d'employés.

Listing 6 – "Prédicat **pose_contrainte2** avec seuil pour le profit et minimisation du nombre d'ouvriers"

```
1  pose_contraintes2(Fabriquer,NbTechniciensTotal,Profit) :-
2          getData(NbTech,NbSor,Tech,Quan,Bene),
3          defineVars(Fabriquer,NbSor),
4          nbOuvriersNecessaires(Fabriquer,Tech,NbSor,NbTechniciensTotal),
5          NbTechniciensTotal #=< NbTech,
6          getVarList(Fabriquer,NbSor,L),
7          profitTotal(Fabriquer,Bene,Quan,NbSor,Profit),
8          Profit #>=1000,
9          labeling(L).
10
11 /* Tests
12 [eclipse 74]: minimize(pose_contraintes2(F,N,P),N).
13 Found a solution with cost 10
14 Found a solution with cost 9
15 Found a solution with cost 8
16 Found a solution with cost 7
17 Found no solution with cost -1.0Inf .. 6
18
```

OK

```
F = [](1, 0, 1, 0, 0, 0, 0, 0, 0)
N = 7
P = 1040
Yes (0.00s cpu)
*/
```

Pour obtenir ce résultat, nous avons juste rajouter une contrainte pour le seuil du profit, et une recherche du minimum pour le nombre d'ouvrier.

4 Code Complet, avec l'ensemble des tests

Listing 7 – "TP5"

```
%TP5

:- lib(ic).
:- lib(branch_and_bound).

getData(NbTechniciens,NbSortes,Techniciens,Quantite,Benefice) :-
        NbTechniciens = 22,
        NbSortes = 9,
        Techniciens = [](5,7,2,6,9,3,7,5,3),
        Quantite = [](140,130,60,95,70,85,100,30,45),
        Benefice = [](4,5,8,5,6,4,7,10,11).

defineVars(Fabriquer,NbSortes):-
        dim(Fabriquer,[NbSortes]),
        (for(I,1,NbSortes),param(Fabriquer)
        do
                Fabriquer[I] #:: 0..1
        ).

getVarList(Fabriquer,NbSortes,L) :-
        (for(Ind,1,NbSortes),fromto([],In,Out,L),param(Fabriquer)
                do
                        Var #= Fabriquer[Ind],
                        append(In,[Var],Out)
        ).

nbOuvriersNecessaires(Fabriquer,Techniciens,NbSortes,NbOuvriersTotal) :-
        (for(J,1,NbSortes),fromto(0,In,Out,NbOuvriersTotal),param(Fabriquer,
            Techniciens)
                do
                        Var is Fabriquer[J],
                        NbTech is Techniciens[J],
                        Out #= In + NbTech * Var
        ).

vectBeneficeTotal(Fabriquer,Benefice,Quantite,NbSortes,VectBenefTotal) :-
        (for(K,1,NbSortes),fromto([],In,Out,VectBenefTotal),param(Fabriquer,
            Benefice,Quantite)
                do
                        Var is Fabriquer[K],
                        Benef is Benefice[K],
                        Quan is Quantite[K],
                        Elem #= Var * Benef * Quan,
                        append(In,[Elem],Out)
        ).

profitTotal(Fabriquer,Benefice,Quantite,NbSortes,ProfitTotal) :-
        vectBeneficeTotal(Fabriquer,Benefice,Quantite,NbSortes,VectBenefTotal
            ),
```



```
        ProfitTotal #= sum(VectBenefTotal).

pose_contraintes(Fabriquer,NbTechniciensTotal,Profit) :-
        getData(NbTech,NbSor,Tech,Quan,Bene),
        defineVars(Fabriquer,NbSor),
        nbOuvriersNecessaires(Fabriquer,Tech,NbSor,NbTechniciensTotal),
        NbTechniciensTotal #=< NbTech,
        getVarList(Fabriquer,NbSor,L),
        profitTotal(Fabriquer,Bene,Quan,NbSor,Profit),
        labeling(L).

test(X,Y,Z,W) :-
        [X,Y,Z,W] #:: [0..10],
        X #= Z+Y+2*W,
        X#\= Z+Y+W.

/* Question 5.4
Il faut toujours faire un labeling dans le but,
car il faut que les variables aient ete instanciees.

[eclipse 53]: minimize(test(X,Y,Z,W),X).
bb_min: search did not instantiate cost variable
Aborting execution ...
Abort

Avec le labeling sur [X,Y,Z,W], cela fonctionne, Prolog trouve les reponses

[eclipse 57]: test(X,Y,Z,W),labeling([X,Y,Z,W]).

X = 2
Y = 0
Z = 0
W = 1
Yes (0.00s cpu, solution 1, maybe more) ? ;

X = 3
Y = 0
Z = 1
W = 1
Yes (0.00s cpu, solution 2, maybe more) ? ;
*/

/* Test

[eclipse 71]: P2#=P*(-1), minimize(pose_contraintes(F,N,P),P2).
Found a solution with cost 0
Found a solution with cost -495
Found a solution with cost -795
Found a solution with cost -1195
Found a solution with cost -1495
Found a solution with cost -1535
Found a solution with cost -1835
Found a solution with cost -1955
```



```
Found a solution with cost -1970
Found a solution with cost -2010
Found a solution with cost -2015
Found a solution with cost -2315
Found a solution with cost -2490
Found a solution with cost -2665
Found no solution with cost -1.01Inf .. -2666

P2 = -2665
P = 2665
F = [](0, 1, 1, 0, 0, 1, 1, 0, 1)
N = 22
Yes (0.00s cpu)
*/

% Question 5.6

pose_contraintes2(Fabriquer,NbTechniciensTotal,Profit) :-
        getData(NbTech,NbSor,Tech,Quan,Bene),
        defineVars(Fabriquer,NbSor),
        nbOuvriersNecessaires(Fabriquer,Tech,NbSor,NbTechniciensTotal),
        NbTechniciensTotal #=< NbTech,
        getVarList(Fabriquer,NbSor,L),
        profitTotal(Fabriquer,Bene,Quan,NbSor,Profit),
        Profit #>=1000,
        labeling(L).

/* Tests
[eclipse 74]: minimize(pose_contraintes2(F,N,P),N).
Found a solution with cost 10
Found a solution with cost 9
Found a solution with cost 8
Found a solution with cost 7
Found no solution with cost -1.01Inf .. 6

F = [](1, 0, 1, 0, 0, 0, 0, 0, 0)
N = 7
P = 1040
Yes (0.00s cpu)
*/
```

Rapport Travaux Pratiques :
Programmation par Contraintes
- TP 6 :
Sur une balançoire

Nicolas Desfeux
Aurélien Texier

13 avril 2011

Dans ce TP, nous allons chercher à appliquer ce que nous avons appris à un problème de la vie réelle (aussi vraisemblable soit-il !). Le problème modélise une balançoire à seize places (huit de chaque côté), qu'une famille de dix personnes veut utiliser. Il y a plusieurs règles à respecter pour cette balançoire :

- La balançoire doit être équilibrée (calcul du moments des forces) ;
- La maman et le papa souhaitent encadrer leurs enfants pour les surveiller ;
- Dan et Max doivent être assis chacun devant un parent.
- Il doit y avoir 5 personnes de chaque côté.

Table des matières

1 Trouver une solution au problème

Question 6.1 Nous avons procéder comme pour les problèmes précédents pour résoudre celui ci. Vous retrouverez donc les étapes suivantes : définition des variables, définitions des domaines, pose des contraintes.

Listing 1 – "Solution 1"

```
%TP6

:- lib(ic).
:- lib(branch_and_bound).
:- lib(ic_global).

getData(Poids,NbPersonnes,NbPlaces) :-
        NbPersonnes = 10,
        NbPlaces = 16,
        Poids = [](24,39,85,60,165,6,32,123,7,14).

defineVars(Places,NbPlaces,NbPersonnes):-
        dim(Places,[NbPersonnes]),
        Borne #= NbPlaces/2,
        (for(I,1,NbPersonnes),param(Places,Borne)
                do
                        Places[I] #:: -Borne..Borne,
                        Places[I] #\= 0
        ).

getVarList(Places,NbPersonnes,L) :-
        (for(Ind,1,NbPersonnes),fromto([],In,Out,L),param(Places)
                do
                        Var #= Places[Ind],
                        append([Var],In,Out)
        ).

balancoireEquilibree(Poids,NbPersonnes,Places) :-
        (for(J,1,NbPersonnes),fromto(0,In,Out,SommeMoments),param(Places,
            Poids)
                do
                        Distance #= Places[J],
                        Poids1 #= Poids[J],
                        Out #= In + Distance * Poids1
        ),
        SommeMoments #= 0.

pasMemesPlaces(Places) :-
        ic:alldifferent(Places).

cinqChaqueCote(Places,NbPersonnes) :-
        NbPersGauche #= NbPersonnes/2,
        (for(K,1,NbPersonnes),fromto(0,In,Out,NbPersonnesAGauche),param(
            Places)
                do
                        Var #= Places[K],
```



```
                        (Var #< 0) => (Out #= In + 1),
                        (Var #> 0) => (Out #= In)
        ),
        NbPersGauche #= NbPersonnesAGauche.

parentsEncadrentEnfants(Places) :-
        PlaceMere #= Places[4],
        PlacePere #= Places[8],
        ((PlaceMere #= ic:max(Places)) and (PlacePere #= ic:min(Places)))
        or
        ((PlacePere #= ic:max(Places)) and (PlaceMere #= ic:min(Places))).
        /*
        (maxlist(Places,PlaceMere) and minlist(Places,PlacePere))
        or
        (maxlist(Places,PlacePere) and minlist(Places,PlaceMere)).
*/

danEtMaxDevantParents(Places) :-
        PlaceDan #= Places[6],
        PlaceMax #= Places[9],
        PlaceMere #= Places[4],
        PlacePere #= Places[8],
        (((PlaceMere #<0) and (PlaceDan #<0)) and ((PlaceMere #= PlaceDan -
            1) and (PlacePere #= PlaceMax + 1)))
        or
        (((PlaceMere #<0) and (PlaceMax #<0)) and ((PlaceMere #= PlaceMax -
            1) and (PlacePere #= PlaceDan + 1)))
        or
        (((PlacePere #<0) and (PlaceDan #<0)) and ((PlacePere #= PlaceDan -
            1) and (PlaceMere #= PlaceMax + 1)))
        or
        (((PlacePere #<0) and (PlaceMax #<0)) and ((PlacePere #= PlaceMax -
            1) and (PlaceMere #= PlaceDan + 1))).

solve(Places) :-
        getData(Poids,NbPers,NbPlac),
        defineVars(Places,NbPlac,NbPers),
        pasMemesPlaces(Places),
        balancoireEquilibree(Poids,NbPers,Places),
        cinqChaqueCote(Places,NbPers),
        parentsEncadrentEnfants(Places),
        danEtMaxDevantParents(Places),
        getVarList(Places,NbPers,L),
        labeling(L).

/* Tests
[eclipse 29]: solve(P).

P = [](-6, -5, -4, -8, 2, 5, 3, 6, -7, 1)
Yes (0.02s cpu, solution 1, maybe more) ? ;

P = [](-6, -5, -1, 8, 5, 7, 3, -8, -7, 1)
Yes (0.02s cpu, solution 2, maybe more) ? ;
```



```

P = [](-6, -5, 1, -8, -2, 6, 5, 7, -7, 4)
Yes (0.02s cpu, solution 3, maybe more) ?
*/
```

Question 6.2 Voici une solution possible :

```
P = [](-6, -5, -4, -8, 2, 5, 3, 6, -7, 1)
Yes (0.13s cpu, solution 1, maybe more) ? ;
```

Eclipse a besoin de 0.13s pour trouver ce premier résultat. Il y en a bien sur d'autres.

Question 6.3 Il existe une symétrie entre les solutions du problème. Une solution qui place 5 personnes à droite possède une symétrie si l'on met ces 5 personnes à gauche. Il n'est donc pas forcément utile de les faire rechercher par Eclipse !
Pour éliminer cette symétrie, nous avons choisi de créer une nouvelle contrainte

```
% Impose que la mere soit a gauche sur la balancoire
elimineSymetrie(Places) :-
        Places[4] #< 0.
```

Et nous avons juste eu besoin d'adapter le prédicat solve.

Listing 2 – "Solution 2"

```
solve2(Places) :-
        getData(Poids,NbPers,NbPlac),
        defineVars(Places,NbPlac,NbPers),
        elimineSymetrie(Places), %Gain de temps monstrueux
        pasMemesPlaces(Places),
        cinqChaqueCote(Places,NbPers),
        balancoireEquilibree(Poids,NbPers,Places),
        parentsEncadrentEnfants(Places),
        danEtMaxDevantParents(Places),
        getVarList(Places,NbPers,L),
        labeling(L).

/* Tests
[eclipse 31]: solve2(P).

P = [](-6, -5, -4, -8, 2, 5, 3, 6, -7, 1)
Yes (0.02s cpu, solution 1, maybe more) ? ;

P = [](-6, -5, 1, -8, -2, 6, 5, 7, -7, 4)
Yes (0.04s cpu, solution 2, maybe more) ? ;

P = [](-6, -5, 3, -8, -3, -7, 4, 7, 6, 5)
Yes (0.01s cpu, solution 3, maybe more) ?
*/
```

2 Trouver la meilleure solution

Question 6.4 On cherche maintenant à trouver la meilleur solution, c'est à dire un solution qui minimise les normes des moments des forces pour la balançoire. Pour cela, nous avons effectuer deux modifications dans notre code. Nous avons ajouté le calcul du moment des forces sur chque coté de la balançoire. Nous avons aussi modifier notre prédicat solve, afin qu'il permettent de rechercher le minimum des moments de forces que l'on vient de calculer.

Listing 3 – "Solution 3"

```
balancoireEquilibree2(Poids,NbPersonnes,Places,SNormeMoments) :-
        ( for(J,1,NbPersonnes),fromto(0,In,Out,SommeMoments),fromto(0,In2,
            Out2,SNormeMoments),param(Places,Poids)
                do
                        Distance #= Places[J],
                        Poids1 #= Poids[J],
                        Out #= In + Distance * Poids1,
                        (Places[J] #> 0) => (Out2 #= In2 + Distance * Poids1)
                            ,
                        (Places[J] #=< 0) => (Out2 #= In2)
        ),
        SommeMoments #= 0.

solve3(Places) :-
        getData(Poids,NbPers,NbPlac),
        defineVars(Places,NbPlac,NbPers),
        elimineSymetrie(Places), %Gain de temps monstrueux
        pasMemesPlaces(Places),
        cinqChaqueCote(Places,NbPers),
        balancoireEquilibree2(Poids,NbPers,Places,SNormeMom),
        parentsEncadrentEnfants(Places),
        danEtMaxDevantParents(Places),
        getVarList(Places,NbPers,L),
        minimize(labeling(L),SNormeMom).

/* Tests
[eclipse 25]: solve3(P).
Found a solution with cost 874
Found a solution with cost 872
Found a solution with cost 802
Found no solution with cost -1.01Inf .. 801

P = [](-3, 1, -2, -6, -1, 4, 3, 5, -5, 2)
Yes (1.66s cpu)
*/
```

Dans le prédicat balancoireEquilibree2, on effectue le calcul de la norme du moment des forces sur un coté de la balançoire. Dans notre problème, il suffit de calculer un seul coté, puisque l'on impose que la balançoire soit équilibrée !

Utilisation du prédicat search/6

Dans cette partie, nous avons utiliser le prédicat search qui va nous permettre de controler l'énumération de Eclipse (en influent sur l'ordre dans lequel on instancie les variables, et sur l'ordre dans lequel chaque valeur du domaine d'une variable est testée).

Version 1 « Pour réussir, essaie d'abord là où tu as le plus de chances d'échouer ». Ici, les premières varaibles testée sont celles qui interviennent dans le plus de contraintes.

Listing 4 – "Version 1"

```
solve4(Places) :-
        getData(Poids,NbPers,NbPlac),
        defineVars(Places,NbPlac,NbPers),
        elimineSymetrie(Places),
        pasMemesPlaces(Places),
        cinqChaqueCote(Places,NbPers),
        balancoireEquilibree2(Poids,NbPers,Places,SNormeMom),
        parentsEncadrentEnfants(Places),
        danEtMaxDevantParents(Places),
        getVarList(Places,NbPers,L),
        minimize(search(L,0,occurrence,indomain_min,complete,[]),SNormeMom).

/* Tests
[eclipse 9]: solve4(P).
Found a solution with cost 953
Found a solution with cost 852
Found a solution with cost 834
Found a solution with cost 802
Found no solution with cost -1.01Inf .. 801

P = [](-3, 1, -2, -6, -1, 4, 3, 5, -5, 2)
Yes (2.50s cpu)
*/
```

Ici, il semble que most_constrained soit plus adapte que occurrence, vu la vitesse de reponse (2.13s au lieu de 2.50s)

Version 2 Ici on vise a essayer de mettre les places les plus au centre en premier, parce que c'est dans ce cas que l'on minimise le mieux le moment total.

Listing 5 – "Version 2"

```
quick_sort([],[]).
quick_sort([H|T],Sorted):-
        pivoting(H,T,L1,L2),quick_sort(L1,Sorted1),quick_sort(L2,Sorted2),
        append(Sorted1,[H|Sorted2],Sorted).

pivoting(_H,[],[],[]).
pivoting(H,[X|T],[X|L],G):-abs(X)=<abs(H),pivoting(H,T,L,G),!.
pivoting(H,[X|T],L,[X|G]):-abs(X)>abs(H),pivoting(H,T,L,G),!.

/* Tests
[eclipse 50]: quick_sort([1,2,3,4,-1,-2,-3,-4],L).

L = [-1, 1, -2, 2, -3, 3, -4, 4]
Yes (0.00s cpu)
*/

choice(X) :-
        get_domain_as_list(X,L),
        quick_sort(L,L2),
        member(X,L2).

solve5(Places) :-
        getData(Poids,NbPers,NbPlac),
        defineVars(Places,NbPlac,NbPers),
        elimineSymetrie(Places),
        pasMemesPlaces(Places),
        cinqChaqueCote(Places,NbPers),
        balancoireEquilibree2(Poids,NbPers,Places,SNormeMom),
        parentsEncadrentEnfants(Places),
        danEtMaxDevantParents(Places),
        getVarList(Places,NbPers,L),
        minimize(search(L,0,input_order,choice,complete,[]),SNormeMom).

/* Tests
[eclipse 32]: solve5(P).
Found a solution with cost 1216
Found a solution with cost 956
Found a solution with cost 947
Found a solution with cost 917
Found a solution with cost 852
Found a solution with cost 802
Found no solution with cost -1.01Inf .. 801

P = [](-3, 1, -2, -6, -1, 4, 3, 5, -5, 2)
Yes (1.80s cpu)
*/
```

Version 3 Cette version est une combinaison des versions précédentes.

Listing 6 – "Version 3"

```
solve6(Places) :-
        getData(Poids,NbPers,NbPlac),
        defineVars(Places,NbPlac,NbPers),
        elimineSymetrie(Places),
        pasMemesPlaces(Places),
        cinqChaqueCote(Places,NbPers),
        balancoireEquilibree2(Poids,NbPers,Places,SNormeMom),
        parentsEncadrentEnfants(Places),
        danEtMaxDevantParents(Places),
        getVarList(Places,NbPers,L),
        minimize(search(L,0,occurrence,choice,complete,[]),SNormeMom).

/* Tests
[eclipse 52]: solve6(P).
Found a solution with cost 933
Found a solution with cost 898
Found a solution with cost 872
Found a solution with cost 852
Found a solution with cost 848
Found a solution with cost 834
Found a solution with cost 802
Found no solution with cost -1.01Inf .. 801

P = [](-3, 1, -2, -6, -1, 4, 3, 5, -5, 2)
Yes (2.98s cpu)
*/
```

Le résultat n'est pas satisfaisant ici.

Version 4 Cette version est basé sur les heuristiques de choix de l'ordre des variables. L'idée ici est de trier la liste des variables dans l'ordre décroissant des poids, pour que lorsque le solveur trouve un score des variables à instancier égal, il prenne les variables dans l'ordre décroissant de leur poids respectif.
Nous n'avons pas réussi à implémenter cette version.

3 Code Complet, avec l'ensemble des tests

Listing 7 – "TP6"

```
%TP6

:- lib(ic).
:- lib(branch_and_bound).
:- lib(ic_global).

getData(Poids,NbPersonnes,NbPlaces) :-
        NbPersonnes = 10,
        NbPlaces = 16,
        Poids = [](24,39,85,60,165,6,32,123,7,14).

defineVars(Places,NbPlaces,NbPersonnes):-
        dim(Places,[NbPersonnes]),
        Borne #= NbPlaces/2,
        ( for(I,1,NbPersonnes),param(Places,Borne)
                do
                        Places[I] #:: -Borne..Borne,
                        Places[I] #\= 0
        ).

getVarList(Places,NbPersonnes,L) :-
        ( for(Ind,1,NbPersonnes),fromto([],In,Out,L),param(Places)
                do
                        Var #= Places[Ind],
                        append([Var],In,Out)
        ).

balancoireEquilibree(Poids,NbPersonnes,Places) :-
        ( for(J,1,NbPersonnes),fromto(0,In,Out,SommeMoments),param(Places,
            Poids)
                do
                        Distance #= Places[J],
                        Poids1 #= Poids[J],
                        Out #= In + Distance * Poids1
        ),
        SommeMoments #= 0.

pasMemesPlaces(Places) :-
        ic:alldifferent(Places).

cinqChaqueCote(Places,NbPersonnes) :-
        NbPersGauche #= NbPersonnes/2,
        ( for(K,1,NbPersonnes),fromto(0,In,Out,NbPersonnesAGauche),param(
            Places)
                do
                        Var #= Places[K],
                        (Var #< 0) => (Out #= In + 1),
                        (Var #> 0) => (Out #= In)
        ),
```



```
        NbPersGauche #= NbPersonnesAGauche.

parentsEncadrentEnfants(Places) :-
        PlaceMere #= Places[4],
        PlacePere #= Places[8],
        ((PlaceMere #= ic:max(Places)) and (PlacePere #= ic:min(Places)))
        or
        ((PlacePere #= ic:max(Places)) and (PlaceMere #= ic:min(Places))).
        /*
        (maxlist(Places,PlaceMere) and minlist(Places,PlacePere))
        or
        (maxlist(Places,PlacePere) and minlist(Places,PlaceMere)).
*/

danEtMaxDevantParents(Places) :-
        PlaceDan #= Places[6],
        PlaceMax #= Places[9],
        PlaceMere #= Places[4],
        PlacePere #= Places[8],
        (((PlaceMere #<0) and (PlaceDan #<0)) and ((PlaceMere #= PlaceDan -
            1) and (PlacePere #= PlaceMax + 1)))
        or
        (((PlaceMere #<0) and (PlaceMax #<0)) and ((PlaceMere #= PlaceMax -
            1) and (PlacePere #= PlaceDan + 1)))
        or
        (((PlacePere #<0) and (PlaceDan #<0)) and ((PlacePere #= PlaceDan -
            1) and (PlaceMere #= PlaceMax + 1)))
        or
        (((PlacePere #<0) and (PlaceMax #<0)) and ((PlacePere #= PlaceMax -
            1) and (PlaceMere #= PlaceDan + 1))).

solve(Places) :-
        getData(Poids,NbPers,NbPlac),
        defineVars(Places,NbPlac,NbPers),
        pasMemesPlaces(Places),
        balancoireEquilibree(Poids,NbPers,Places),
        cinqChaqueCote(Places,NbPers),
        parentsEncadrentEnfants(Places),
        danEtMaxDevantParents(Places),
        getVarList(Places,NbPers,L),
        labeling(L).

/* Tests
[eclipse 29]: solve(P).

P = [](-6, -5, -4, -8, 2, 5, 3, 6, -7, 1)
Yes (0.13s cpu, solution 1, maybe more) ? ;

P = [](-6, -5, -1, 8, 5, 7, 3, -8, -7, 1)
Yes (0.15s cpu, solution 2, maybe more) ? ;

P = [](-6, -5, 1, -8, -2, 6, 5, 7, -7, 4)
Yes (0.02s cpu, solution 3, maybe more) ?
```



```
*/

% Impose que la mere soit a gauche sur la balancoire
elimineSymetrie(Places) :-
        Places[4] #< 0.

solve2(Places) :-
        getData(Poids,NbPers,NbPlac),
        defineVars(Places,NbPlac,NbPers),
        elimineSymetrie(Places), %Gain de temps monstrueux
        pasMemesPlaces(Places),
        cinqChaqueCote(Places,NbPers),
        balancoireEquilibree(Poids,NbPers,Places),
        parentsEncadrentEnfants(Places),
        danEtMaxDevantParents(Places),
        getVarList(Places,NbPers,L),
        labeling(L).

/* Tests
[eclipse 31]: solve2(P).

P = [](-6, -5, -4, -8, 2, 5, 3, 6, -7, 1)
Yes (0.02s cpu, solution 1, maybe more) ? ;

P = [](-6, -5, 1, -8, -2, 6, 5, 7, -7, 4)
Yes (0.04s cpu, solution 2, maybe more) ? ;

P = [](-6, -5, 3, -8, -3, -7, 4, 7, 6, 5)
Yes (0.01s cpu, solution 3, maybe more) ?
*/

balancoireEquilibree2(Poids,NbPersonnes,Places,SNormeMoments) :-
        ( for(J,1,NbPersonnes),fromto(0,In,Out,SommeMoments),fromto(0,In2,
            Out2,SNormeMoments),param(Places,Poids)
                do
                        Distance #= Places[J],
                        Poids1 #= Poids[J],
                        Out #= In + Distance * Poids1,
                        (Places[J] #> 0) => (Out2 #= In2 + Distance * Poids1)
                            ,
                        (Places[J] #=< 0) => (Out2 #= In2)
        ),
        SommeMoments #= 0.

solve3(Places) :-
        getData(Poids,NbPers,NbPlac),
        defineVars(Places,NbPlac,NbPers),
        elimineSymetrie(Places), %Gain de temps monstrueux
        pasMemesPlaces(Places),
        cinqChaqueCote(Places,NbPers),
        balancoireEquilibree2(Poids,NbPers,Places,SNormeMom),
        parentsEncadrentEnfants(Places),
        danEtMaxDevantParents(Places),
```



```
        getVarList(Places,NbPers,L),
        minimize(labeling(L),SNormeMom).

/* Tests
[eclipse 25]: solve3(P).
Found a solution with cost 874
Found a solution with cost 872
Found a solution with cost 802
Found no solution with cost -1.01Inf .. 801

P = [](-3, 1, -2, -6, -1, 4, 3, 5, -5, 2)
Yes (1.66s cpu)
*/

% Version 1 de search :

solve4(Places) :-
        getData(Poids,NbPers,NbPlac),
        defineVars(Places,NbPlac,NbPers),
        elimineSymetrie(Places),
        pasMemesPlaces(Places),
        cinqChaqueCote(Places,NbPers),
        balancoireEquilibree2(Poids,NbPers,Places,SNormeMom),
        parentsEncadrentEnfants(Places),
        danEtMaxDevantParents(Places),
        getVarList(Places,NbPers,L),
        minimize(search(L,0,occurrence,indomain_min,complete,[]),SNormeMom).

/* Tests
[eclipse 9]: solve4(P).
Found a solution with cost 953
Found a solution with cost 852
Found a solution with cost 834
Found a solution with cost 802
Found no solution with cost -1.01Inf .. 801

P = [](-3, 1, -2, -6, -1, 4, 3, 5, -5, 2)
Yes (2.50s cpu)
*/
% Ici, il semble que most_constrained soit plus adapte que occurrence, vu la
    vitesse de reponse (2.13s au lieu de 2.50s)

% Version 2 de search :

/* Ici on vise a essayer de mettre les places les plus au centre en premier,
parce que c'est dans ce cas que l'on minimise le mieux le moment total. */
quick_sort([],[]).
quick_sort([H|T],Sorted):-
        pivoting(H,T,L1,L2),quick_sort(L1,Sorted1),quick_sort(L2,Sorted2),
        append(Sorted1,[H|Sorted2],Sorted).

pivoting(_H,[],[],[]).
pivoting(H,[X|T],[X|L],G):-abs(X)=<abs(H),pivoting(H,T,L,G),!.
```

```
pivoting(H,[X|T],L,[X|G]):-abs(X)>abs(H),pivoting(H,T,L,G),!.

/* Tests
[eclipse 50]: quick_sort([1,2,3,4,-1,-2,-3,-4],L).

L = [-1, 1, -2, 2, -3, 3, -4, 4]
Yes (0.00s cpu)
*/

choice(X) :-
        get_domain_as_list(X,L),
        quick_sort(L,L2),
        member(X,L2).

solve5(Places) :-
        getData(Poids,NbPers,NbPlac),
        defineVars(Places,NbPlac,NbPers),
        elimineSymetrie(Places),
        pasMemesPlaces(Places),
        cinqChaqueCote(Places,NbPers),
        balancoireEquilibree2(Poids,NbPers,Places,SNormeMom),
        parentsEncadrentEnfants(Places),
        danEtMaxDevantParents(Places),
        getVarList(Places,NbPers,L),
        minimize(search(L,0,input_order,choice,complete,[]),SNormeMom).

/* Tests
[eclipse 32]: solve5(P).
Found a solution with cost 1216
Found a solution with cost 956
Found a solution with cost 947
Found a solution with cost 917
Found a solution with cost 852
Found a solution with cost 802
Found no solution with cost -1.0Inf .. 801

P = [](-3, 1, -2, -6, -1, 4, 3, 5, -5, 2)
Yes (1.80s cpu)
*/

% Version 3 de search :

solve6(Places) :-
        getData(Poids,NbPers,NbPlac),
        defineVars(Places,NbPlac,NbPers),
        elimineSymetrie(Places),
        pasMemesPlaces(Places),
        cinqChaqueCote(Places,NbPers),
        balancoireEquilibree2(Poids,NbPers,Places,SNormeMom),
        parentsEncadrentEnfants(Places),
        danEtMaxDevantParents(Places),
        getVarList(Places,NbPers,L),
        minimize(search(L,0,occurrence,choice,complete,[]),SNormeMom).
```



```

/* Tests
[eclipse 52]: solve6(P).
Found a solution with cost 933
Found a solution with cost 898
Found a solution with cost 872
Found a solution with cost 852
Found a solution with cost 848
Found a solution with cost 834
Found a solution with cost 802
Found no solution with cost -1.0Inf .. 801

P = [](-3, 1, -2, -6, -1, 4, 3, 5, -5, 2)
Yes (2.98s cpu)
*/

% Le resultat n'est pas satisfaisant ici

% Version 4 :

solve7(Places) :-
        getData(Poids,NbPers,NbPlac),
        defineVars(Places,NbPlac,NbPers),
        elimineSymetrie(Places),
        pasMemesPlaces(Places),
        cinqChaqueCote(Places,NbPers),
        balancoireEquilibree2(Poids,NbPers,Places,SNormeMom),
        parentsEncadrentEnfants(Places),
        danEtMaxDevantParents(Places),
        getVarList2(Places,NbPers,L),
        minimize(search(L,0,input_order,indomain_split,complete,[]),SNormeMom
            ).
```