

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	2
1.1	Des contraintes sur les arbres	2
1.2	Prolog ne comprend pas les Maths (mais il a une bonne calculatrice)	2
1.3	Le solveur ic à la rescousse	4
2	Zoologie	6
3	Le "OU" en contraintes	7
4	Code Complet, avec tests	11

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 ),
14 couleurBateau(CouleurBateau),
15 CouleurVoiture=CouleurBateau.
16
17 /* Tests
18 [eclipse 78]: choixCouleur(CouleurBateau, CouleurVoiture).
19
20 CouleurBateau = vert(clair)
21 CouleurVoiture = vert(clair)
22 Yes (0.00s cpu, solution 1, maybe more) ? ;
23
24 CouleurBateau = blanc
25 CouleurVoiture = blanc
26 Yes (0.00s cpu, solution 2)
27 */
```

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 ? Quel est le rôle de l'unification ?

1.2 Prolog ne comprend pas les Maths (mais il a une bonne 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) :- Var=(Min, Max),
3 is(M, Min+1),
4 isBetween(Var, M, Max).
```

Tests ??

2

1

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)
7 , nbMaxC(NbMaxC), nbMinC(NbMinC),
8 isBetween(NbResistance, NbMinR, NbMaxR),
9 isBetween(NbCondensateur, NbMinC, NbMaxC),
10 NbResistance >= NbCondensateur.
11
12 /* Tests
13 [eclipse 79]: commande(NbResistance, NbCondensateur).
14
15 NbResistance = 9000
16 NbCondensateur = 9000
17 Yes (7.73s cpu, solution 1, maybe more) ? ;
18
19 NbResistance = 9001
20 NbCondensateur = 9000
21 Yes (7.74s cpu, solution 2, maybe more) ? ;
22
23 NbResistance = 9001
24 NbCondensateur = 9001
25 Yes (7.74s cpu, solution 3, maybe more) ? ;
26 */
```

Bien

Estimation du nombre de solutions ?

Le test de commande permet de valider le fonctionnement de isBetween2.

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

3

Pas très rigoureux dans la forme...

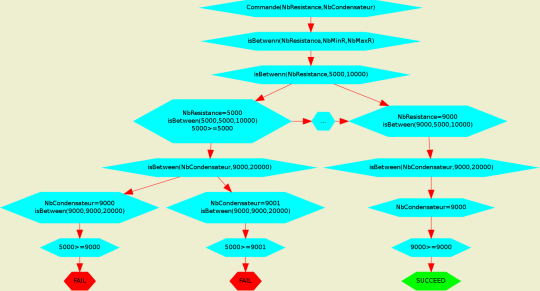


FIGURE 1 – Arbre Prolog 1

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

4

```

15 Yes (0.00s cpu)
16 */

```

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"
1 commande3(NbResistance , NbCondensateur) :-      nbMaxR(NbMaxR) , nbMinR(NbMinR)
   , nbMaxC(NbMaxC) , nbMinC(NbMinC) ,
2
   NbResistance #:: NbMinR..
   NbMaxR,
3
   NbCondensateur #:: NbMinC..
   NbMaxC,
4
   NbResistance #>=
   NbCondensateur ,
   labeling([NbResistance ,
5
   NbCondensateur]).
6
7 /* Tests
8 [eclipse 91]: commande3(NbResistance , NbCondensateur).
9
10 NbResistance = 9000
11 NbCondensateur = 9000
12 Yes (0.00s cpu , solution 1, maybe more) ? ;
13
14 NbResistance = 9001
15 NbCondensateur = 9000
16 Yes (0.00s cpu , solution 2, maybe more) ? ;
17
18 NbResistance = 9001
19 NbCondensateur = 9001
20 Yes (0.00s cpu , solution 3, maybe more) ? ;
21
22 NbResistance = 9002
23 NbCondensateur = 9000
24 Yes (0.00s cpu , solution 4, maybe more) ?
25
26 */

```

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

5

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

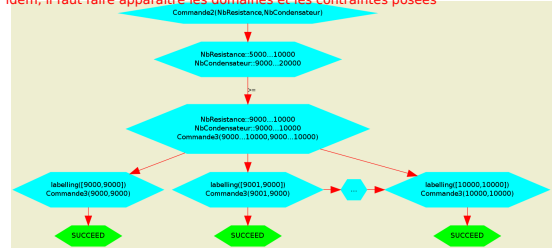


FIGURE 2 – Arbre Prolog 2

2 Zoologie

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

```

Listing 6 – "Prédicat chapie"
1 nbChatsMax(1000) .
2 nbPiesMax(1000) .
3
4 chapie(Chats , Pies , Pattes , Tetes) :-      nbChatsMax(NbChatsMax) ,
   nbPiesMax(NbPiesMax) ,
5
   Chats #:: 0 .. NbChatsMax ,
6
   Pies #:: 0 .. NbPiesMax ,
7
   Pattes #= Chats+4+Pies*2 ,
8
   Tetes #= Chats+Pies ,
9
   labeling([Chats , Pies]).
10

```

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

```

Listing 7 – "Prédicat chapie - test"
1 [eclipse 52]: chapie(2,Pies , Pattes , 5) .
2
3 Pies = 3
4 Pattes = 14
5 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 ???

6

```

Listing 8 – "Prédicat chapie - test"
1 [eclipse 4]: chapie(Chats , Pies , Pattes , Tetes) , Pattes#Tetes+3.
2
3 Chats = 0
4 Pies = 0
5 Pattes = 0
6 Tetes = 0
7 Yes (0.00s cpu , solution 1, maybe more) ? ;
8
9 Chats = 1
10 Pies = 1
11 Pattes = 6
12 Tetes = 2
13 Yes (0.00s cpu , solution 2, maybe more) ? ;
14
15 Chats = 2
16 Pies = 2
17 Pattes = 12
18 Tetes = 4
19 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"
1 vabs1(Val , AbsVal) :- Val#AbsVal ,
   AbsVal#>0 ,
2
   labeling([Val , AbsVal]).
3
4 vabs1(Val , AbsVal) :- Val*(-1)#AbsVal ,
   AbsVal#>0 ,
5
   labeling([Val , AbsVal]).
6
7
8 vabs2(0,0) .
9 vabs2(Val , AbsVal) :- Val#AbsVal
   or
10
   Val*(-1)#AbsVal ,
11
   AbsVal#>0 ,
12
   labeling([Val , AbsVal]) .
13
14
15 /* Tests
16 [eclipse 5]: vabs1(-2,Y) .
17
18 Y = 2
19 Yes (0.00s cpu)
20 [eclipse 6]: vabs2(-2,Y) .
21
22 Y = 2
23 Yes (0.00s cpu)
24
25

```

7

```

26 [eclipse 7]: vabs1(X,3) .
27
28 X = 3
29 Yes (0.00s cpu , solution 1, maybe more) ? ;
30
31 X = -3
32 Yes (0.00s cpu , solution 2)
33 [eclipse 8]: vabs2(X,3) .
34
35 X = X[-1.0Inf .. 1.0Inf]
36
37
38 Delayed goals:
39 #=(X[-1.0Inf .. 1.0Inf] , 3 , _180{[0 , 1]})
40 #=(-(X[-1.0Inf .. 1.0Inf]) , 3 , _302{[0 , 1]})
41 -(_302{[0 , 1]}) - _180{[0 , 1]} #=< -1
42 Yes (0.00s cpu)
43 */

```

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

Question 1.12 Nous avons mis les solutions dans une liste. 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 la 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"
1 faitListe(VarListe , Taille , Min , Max) :- dim(VarListe , [Taille]) ,
2
   (for(1 , 1 , Taille) , param(VarListe , Min , Max) do
3
   (Elem #:: Min..Max ,
4
   indomain(Elem) ,
5
   VarListe[1] #= Elem
6
   )
7
   ) .
8
9 /* Test
10 [eclipse 45]: faitListe(L,2,1,2) .
11
12 L = [[1 , 1]
13 Yes (0.00s cpu , solution 1, maybe more) ? ;
14
15 L = [[1 , 2]
16 Yes (0.00s cpu , solution 2, maybe more) ? ;
17
18 L = [[2 , 1]
19 Yes (0.00s cpu , solution 3, maybe more) ? ;
20
21 L = [[2 , 2]
22 Yes (0.00s cpu , solution 4)

```

8

```

23
24 Autre test
25 [eclipse 46]: faitListe([](1,6,3),T,2,8).
26
27 No (0.00s cpu)
28
29 */

```

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"

```

1 suite(ListVar) :- dim(ListVar,[Taille]),
2                     (for(1,1,Taille,2),param(ListVar) do
3                       (Temp#=:ListVar[1+1],
4                        vabs1(Temp,VabsDeuxElem),
5                        ListVar[1+2]#=:VabsDeuxElem
6                          - ListVar[1]
7                        )
8                     ).
9 /* Test
10 [eclipse 47]: suite([](3,-5,2)).
11
12 Yes (0.00s cpu)
13
14 Autre test
15 [eclipse 48]: suite([](2,2,3)).
16
17 No (0.00s cpu)
18 */

```

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 – "Requête"

```

1 % dim(L,[10]),L[1]#=1,L[2]#=2,suite(L).
2 % dim(L,[10]),L[1]#=3,L[2]#=7,suite(L).
3 /*
4   dim(L,[10]),
5   Elem1#::1..10,Elem2#::1..10,
6   indomain(Elem1),indomain(Elem2),
7   L[1]#=Elem1,L[2]#=Elem2,
8   suite(L).
9 */
10 /*
11
12 dim(L,[10]),
13 Elem1#::1..10,Elem2#::1..10,
14 indomain(Elem1),indomain(Elem2),

```

Utilisez plutôt labeling !!!

Il fallait utiliser faitListe

9

```

15 L[1]#=Elem1,L[2]#=Elem2,
16 L[1]#\=L[10],
17 suite(L).
18 dim(L,[10]),
19 Elem1#::1..100,Elem2#::1..100,
20 indomain(Elem1),indomain(Elem2),
21 L[1]#=Elem1,L[2]#=Elem2,
22 L[1]#\=L[10],
23 suite(L).
24 No (0.94s cpu)
25
26 No (93.06s cpu)
27
28 */

```

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 parcours 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

10

4 Code Complet, avec tests

Listing 13 – "test"

```

1 :- lib(ic).
2
3 %Question 1.1
4
5 couleurVoiture(rouge).
6 couleurVoiture(vert(clair)).
7 couleurVoiture(gris).
8 couleurVoiture(blanc).
9
10 couleurBateau(vert(_)).
11 couleurBateau(noir).
12 couleurBateau(blanc).
13
14 choixCouleur(CouleurBateau,CouleurVoiture) :- couleurVoiture(CouleurVoiture
15   ),
16   couleurBateau(CouleurBateau),
17   CouleurVoiture=CouleurBateau.
18
19 /* Tests
20 [eclipse 78]: choixCouleur(CouleurBateau,CouleurVoiture).
21
22 CouleurBateau = vert(clair)
23 CouleurVoiture = vert(clair)
24 Yes (0.00s cpu, solution 1, maybe more) ? ;
25
26 CouleurBateau = blanc
27 CouleurVoiture = blanc
28 Yes (0.00s cpu, solution 2)
29 */
30
31 %Question 1.3
32
33 isBetween(Var,Min,Max) :- Var=Min.
34 isBetween(Var,Min,Max) :- \=(Min,Max),
35   is(M,Min+1),
36   isBetween(Var,M,Max).
37
38 %Question 1.4
39
40 nbMaxR(10000).
41 nbMinR(5000).
42 nbMinC(9000).
43 nbMaxC(20000).
44
45 commande(NbResistance,NbCondensateur) :- nbMaxR(NbMaxR),nbMinR(NbMinR)
46   ,nbMaxC(NbMaxC),nbMinC(NbMinC),
47   isBetween(NbResistance,NbMinR,NbMaxR),

```

11

```

47   isBetween(NbCondensateur,
48   NbMinC,NbMaxC),
49   NbResistance>=NbCondensateur.
50
51 /* Tests
52 [eclipse 79]: commande(NbResistance,NbCondensateur).
53
54 NbResistance = 9000
55 NbCondensateur = 9000
56 Yes (7.73s cpu, solution 1, maybe more) ? ;
57
58 NbResistance = 9001
59 NbCondensateur = 9000
60 Yes (7.74s cpu, solution 2, maybe more) ? ;
61
62 NbResistance = 9001
63 NbCondensateur = 9001
64 Yes (7.74s cpu, solution 3, maybe more) ?
65 */
66
67 %Question 1.6
68
69 commandeBis(NbResistance,NbCondensateur) :- nbMaxR(NbMaxR),nbMinR(NbMinR)
70   ,nbMaxC(NbMaxC),nbMinC(NbMinC),
71   NbResistance>=NbCondensateur,
72   isBetween(NbResistance,NbMinR,NbMaxR),
73   isBetween(NbCondensateur,NbMinC,NbMaxC).
74
75 /* Test
76 [eclipse 82]: commandeBis(NbResistance,NbCondensateur).
77 instantiation fault in NbResistance >= NbCondensateur
78 Abort
79 */
80
81 %Question 1.7
82
83 commande2(NbResistance,NbCondensateur) :- nbMaxR(NbMaxR),nbMinR(NbMinR)
84   ,nbMaxC(NbMaxC),nbMinC(NbMinC),
85   NbResistance#::NbMinR..NbMaxR,
86   NbCondensateur#::NbMinC..NbMaxC,
87   NbResistance#>=NbCondensateur.
88
89 /* Tests
90 [eclipse 92]: commande2(NbResistance,NbCondensateur).
91
92 NbResistance = NbResistance{9000 .. 10000}

```

12

```

92 NbCondensateur = NbCondensateur[9000 .. 10000]
93
94
95 Delayed goals:
96   NbCondensateur[9000 .. 10000] - NbResistance[9000 .. 10000] #=< 0
97   Yes (0.00s cpu)
98   */
99
100 %Question 1.8
101
102 commande3(NbResistance, NbCondensateur) :-
    nbMaxR(NbMaxR), nbMinR(NbMinR)
    , nbMaxC(NbMaxC), nbMinC(NbMinC),
103     NbResistance #:: NbMinR..
    NbMaxR,
104     NbCondensateur #:: NbMinC..
    NbMaxC,
105     NbResistance #>=
    NbCondensateur,
106     labeling([NbResistance,
    NbCondensateur]).
107
108 /* Tests
109 [eclipse 91]: commande3(NbResistance, NbCondensateur).
110
111 NbResistance = 9000
112 NbCondensateur = 9000
113 Yes (0.00s cpu, solution 1, maybe more) ? ;
114
115 NbResistance = 9001
116 NbCondensateur = 9000
117 Yes (0.00s cpu, solution 2, maybe more) ? ;
118
119 NbResistance = 9001
120 NbCondensateur = 9001
121 Yes (0.00s cpu, solution 3, maybe more) ? ;
122
123 NbResistance = 9002
124 NbCondensateur = 9000
125 Yes (0.00s cpu, solution 4, maybe more) ?
126
127 */
128
129 %Zoologie
130 %Question 1.9
131
132 nbChatsMax(1000).
133 nbPiesMax(1000).
134
135 chapie(Chats, Pies, Pattes, Tetes) :-
    nbChatsMax(NbChatsMax),
    nbPiesMax(NbPiesMax),
136     Chats #:: 0 .. NbChatsMax,
137     Pies #:: 0 .. NbPiesMax,
138     Pattes #= Chats*4+Pies*2,
139

```

13

```

140 Tetes #= Chats+Pies,
141 labeling([Chats, Pies]).
142
143 /*
144 [eclipse 52]: chapie(2, Pies, Pattes, 5).
145
146 Pies = 3
147 Pattes = 14
148 Yes (0.00s cpu)
149
150 % Question 1.10
151
152 /*
153 [eclipse 4]: chapie(Chats, Pies, Pattes, Tetes), Pattes#>Tetes*3.
154
155 Chats = 0
156 Pies = 0
157 Pattes = 0
158 Tetes = 0
159 Yes (0.00s cpu, solution 1, maybe more) ? ;
160
161 Chats = 1
162 Pies = 1
163 Pattes = 6
164 Tetes = 2
165 Yes (0.00s cpu, solution 2, maybe more) ? ;
166
167 Chats = 2
168 Pies = 2
169 Pattes = 12
170 Tetes = 4
171 Yes (0.00s cpu, solution 3, maybe more) ?
172
173 /*
174 % Question 1.11
175
176 vabs1(Val, AbsVal) :- Val#>=AbsVal,
177     AbsVal#>=0,
178     labeling([Val, AbsVal]).
179
180 vabs1(Val, AbsVal) :- Val#<(-1)*AbsVal,
181     AbsVal#>0,
182     labeling([Val, AbsVal]).
183
184 vabs2(0, 0).
185 vabs2(Val, AbsVal) :- Val#>=AbsVal
186
187
188
189
190 /* Tests
191 [eclipse 5]: vabs1(-2, Y).
192

```

14

```

193 Y = 2
194 Yes (0.00s cpu)
195 [eclipse 6]: vabs2(-2, Y).
196
197 Y = 2
198 Yes (0.00s cpu)
199
200
201 [eclipse 7]: vabs1(X, 3).
202
203 X = 3
204 Yes (0.00s cpu, solution 1, maybe more) ? ;
205
206 X = -3
207 Yes (0.00s cpu, solution 2)
208 [eclipse 8]: vabs2(X, 3).
209
210 X = X[-1.0Inf .. 1.0Inf]
211
212
213 Delayed goals:
214   #=(X[-1.0Inf .. 1.0Inf], 3, _180{[0, 1]})
215   #=(-(X[-1.0Inf .. 1.0Inf]), 3, _302{[0, 1]})
216   -(_302{[0, 1]}) - _180{[0, 1]} #=< -1
217   Yes (0.00s cpu)
218   */
219
220 % Question 1.13
221
222 faitListe(VarListe, Taille, Min, Max) :-
    dim(VarListe, [Taille]),
    (for(1, 1, Taille), param(VarListe, Min, Max) do
    (Elem #:: Min..Max,
    indomain(Elem),
    VarListe[1] #= Elem
    )
    ).
223
224
225 /* Test
226 [eclipse 45]: faitListe(L, 2, 1, 2).
227
228
229 L = [[1, 1]
230
231 Yes (0.00s cpu, solution 1, maybe more) ? ;
232
233 L = [[1, 2]
234
235 Yes (0.00s cpu, solution 2, maybe more) ? ;
236
237 L = [[2, 1]
238
239 Yes (0.00s cpu, solution 3, maybe more) ? ;
240
241 L = [[2, 2]
242
243 Yes (0.00s cpu, solution 4)
244
245 Autre test

```

15

```

246 [eclipse 46]: faitListe([[1, 6, 3], T, 2, 8].
247
248 No (0.00s cpu)
249
250
251
252 % Question 1.14
253
254 suite(ListVar) :-
    dim(ListVar, [Taille]),
    (for(1, 1, Taille-2), param(ListVar) do
    (Temp #= ListVar[1+1],
    vabs1(Temp, VabsDeuxElem),
    ListVar[1+2] #= VabsDeuxElem
    - ListVar[1]
    )
    ).
255
256
257 /* Test
258 [eclipse 47]: suite([[3, -5, 2]).
259
260
261 Yes (0.00s cpu)
262
263
264 Autre test
265 [eclipse 48]: suite([[2, 2, 3]).
266
267
268 No (0.00s cpu)
269
270
271
272 % Question 1.15
273 % dim(L, [10]), L[1]#>=1, L[2]#>=2, suite(L).
274 % dim(L, [10]), L[1]#>=3, L[2]#>=7, suite(L).
275
276 /*
277 dim(L, [10]),
278     Elem1#::1..10, Elem2#::1..10,
279     indomain(Elem1), indomain(Elem2),
280     L[1]#>=Elem1, L[2]#>=Elem2,
281     suite(L).
282
283
284
285 dim(L, [10]),
286     Elem1#::1..10, Elem2#::1..10,
287     indomain(Elem1), indomain(Elem2),
288     L[1]#>=Elem1, L[2]#>=Elem2,
289     L[1]#>=L[10],
290     suite(L).
291
292 dim(L, [10]),
293     Elem1#::1..100, Elem2#::1..100,
294     indomain(Elem1), indomain(Elem2),
295     L[1]#>=Elem1, L[2]#>=Elem2,
296     L[1]#>=L[10],
297     suite(L).
298
299 No (0.94s cpu)
300

```

16

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

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'ordonnement 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 =
2                        [(tache(3,[1],m1,_),
3                         tache(8,[1],m1,_),
4                         tache(8,[4,5],m1,_),
5                         tache(6,[1],m2,_),
6                         tache(3,[1],m2,_),
7                         tache(4,[1,7],m1,_),
8                         tache(8,[3,5],m1,_),
9                         tache(6,[4],m2,_),
10                        tache(6,[6,7],m2,_),
11                        tache(6,[9,12],m2,_),
12                        tache(3,[1],m2,_),
13                        tache(6,[7,8],m2,_))].
14 /* Test
15 [eclipse 3]: taches(T).
16 T = [tache(3, [1], m1, _169), tache(8, [1], m1, _176), tache(8, [4, 5], m1,
17        _183), tache(6, [1], m2, _194), tache(3, [1], m2, _201), tache(4, [1, 7],
18        m1, _210), tache(8, [3, 5], m1, _221), tache(6, [4], m2, _232), tache(6,
```

OK

1

```
3                        do
4                        tache(Duree,_Nom,
5                          _Machine,_Debut)
6                          is Taches[Index]
7                          =
8                          Debut + Duree #=< Fin
9                          ,
10                         Debut #>= 0
11                         ).
12 /*
13 [eclipse 5]: taches(T),domaines(T,10).
14 T = [tache(3, [1], m1, _421[0 .. 7]), tache(8, [1], m1, _644[0 .. 2]), tache(8,
15        [4, 5], m1, _867[0 .. 2]), tache(6, [1], m2, _1090[0 .. 4]), tache(3,
16        [1], m2, _1313[0 .. 7]), tache(4, [1, 7], m1, _1536[0 .. 6]), tache(8,
17        [3, 5], m1, _1759[0 .. 2]), tache(6, [4], m2, _1982[0 .. 4]), tache(6,
18        [6, 7], m2, _2205[0 .. 4]), tache(6, [9, 12], m2, _2428[0 .. 4]), tache
19        (3, [1], m2, _2651[0 .. 7]), tache(6, [7, 8], m2, _2874[0 .. 4])]
20
21 There are 12 delayed goals. Do you want to see them? (y\N)
22 Yes (0.00s cpu)
23 */
```

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(Index, 1, Dim), fromto([], In, Out, List),
3                                       param(Taches)
4                                       do
5                                       Xi is Taches[Index],
6                                       Xi = tache(_,_,_,Debut),
7                                       Out=[Debut|In]
8                                       ),
9                                     ListFin=[Fin|List].
10
11 /*
12 [eclipse 6]: taches(T), getVarList(T, Fin, L).
13
14 T = [(tache(3, [1], m1, _238), tache(8, [1], m1, _243), tache(8, [4, 5], m1,
15        _248), tache(6, [1], m2, _257), tache(3, [1], m2, _262), tache(4, [1, 7],
16        m1, _269), tache(8, [3, 5], m1, _278), tache(6, [4], m2, _287), tache(6,
17        [6, 7], m2, _294), tache(6, [9, 12], m2, _303), tache(3, [1], m2, _312),
18        tache(6, [7, 8], m2, _319))
19
20 Fin = Fin
21 L = [Fin, _319, _312, _303, _294, _287, _278, _269, _262, _257, _248, _243,
22        _238]
23 Yes (0.00s cpu)
24
```

Bien

3

```
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(Index, 1, Dim), param(Taches)
3                          do
4                          Elem is Taches[Index]
5                          ,
6                          writeln(Elem)
7                          ).
8
9 /*
10 [eclipse 4]: taches(T),affiche(T).
11 tache(3, [1], m1, _235)
12 tache(8, [1], m1, _242)
13 tache(8, [4, 5], m1, _249)
14 tache(6, [1], 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, [1], m1, _235), tache(8, [1], m1, _242), tache(8, [4, 5], m1,
25        _249), tache(6, [1], m2, _260), tache(3, [1], m2, _267), tache(4, [1, 7],
26        m1, _276), tache(8, [3, 5], m1, _287), tache(6, [4], m2, _298), tache(6,
27        [6, 7], m2, _307), tache(6, [9, 12], m2, _318), tache(3, [1], m2, _329),
28        tache(6, [7, 8], m2, _338)]
29 Yes (0.00s cpu)
30
31 */
```

OK

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(Index, 1, Dim), param(Taches, Fin)
```

2

```
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 /*
8 [eclipse 4]: solve1(Taches, Fin).
9 lists.eco loaded in 0.00 seconds
10 tache(3, [1], m1, 0)
11 tache(8, [1], m1, 0)
12 tache(8, [4, 5], m1, 0)
13 tache(6, [1], m2, 0)
14 tache(3, [1], m2, 0)
15 tache(4, [1, 7], m1, 0)
16 tache(8, [3, 5], m1, 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, [1], m1, 0), tache(8, [1], m1, 0), tache(8, [4, 5], m1, 0),
23        tache(6, [1], m2, 0), tache(3, [1], m2, 0), tache(4, [1, 7], m1, 0),
24        tache(8, [3, 5], m1, 0), tache(6, [4], m2, 0), tache(6, [6, 7], m2, 0),
25        tache(6, [9, 12], m2, 0), tache(3, [1], m2, 0), tache(6, [7, 8], m2, 0))
26
27 Fin = 8
28 Yes (0.00s cpu, solution 1, maybe more) ? :
29 tache(3, [1], m1, 1)
30 tache(8, [1], m1, 0)
31 tache(8, [4, 5], m1, 0)
32 tache(6, [1], m2, 0)
33 tache(3, [1], m2, 0)
34 tache(4, [1, 7], m1, 0)
35 tache(8, [3, 5], m1, 0)
36 tache(6, [6, 7], m2, 0)
37 tache(6, [9, 12], m2, 0)
38 tache(3, [1], m2, 0)
39 tache(6, [7, 8], m2, 0)
40
```

4

```

38 Taches = [(tache(3, [], ml, 1), tache(8, [], ml, 0), tache(8, [4, 5], ml, 0)
, tache(6, [], m2, 0), tache(3, [1], m2, 0), tache(4, [1, 7], ml, 0),
tache(8, [3, 5], ml, 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))
39 Fin = 8
40 Yes (0.01 s cpu, solution 2, maybe more) ? ;
41 tache(3, [], ml, 2)
42 tache(8, [1, ml, 0)
43 tache(8, [4, 5], ml, 0)
44 tache(6, [1, m2, 0)
45 tache(3, [1], m2, 0)
46 tache(4, [1, 7], ml, 0)
47 tache(8, [3, 5], ml, 0)
48 tache(6, [4], m2, 0)
49 tache(6, [6, 7], m2, 0)
50 tache(6, [9, 12], m2, 0)
51 tache(3, [1], m2, 0)
52 tache(6, [7, 8], m2, 0)
53
54 Taches = [(tache(3, [], ml, 2), tache(8, [], ml, 0), tache(8, [4, 5], ml, 0)
, tache(6, [1, m2, 0), tache(3, [1], m2, 0), tache(4, [1, 7], ml, 0),
tache(8, [3, 5], ml, 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))
55 Fin = 8
56 Yes (0.01 s cpu, solution 3, maybe more) ?
57 */

```

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 puis- qu'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 unfie Fin à 38).

Listing 6 – "precedences"

```

1 precedences(Taches) :- dim(Taches,[Dim]),
2 (for(Indice,1,Dim),param(Taches)
3 do
4 Elem is Taches[Indice],
5 Elem = tache(_D,Noms,_M,Debut),
6 (foreach(1,Noms),param(Debut,Taches)
7 do
8 tache(Duree2,_N,_M,Debut2) is Taches[
9 I],
10 Debut #>= Debut2+Duree2
11 ).

```

Bien

5

```

12 solve(Taches,Fin) :- taches(Taches),
13 domaines(Taches,Fin),
14 precedences(Taches),
15 getVarList(Taches,Fin,Liste),
16 labeling(Liste),
17 affiche(Taches).
18
19
20 /*
21 [eclipse 44]: solve(T,Fin).
22 tache(3, [], ml, 0)
23 tache(8, [1, ml, 0)
24 tache(8, [4, 5], ml, 6)
25 tache(6, [1, m2, 0)
26 tache(3, [1], m2, 3)
27 tache(4, [1, 7], ml, 22)
28 tache(8, [3, 5], ml, 14)
29 tache(6, [4], m2, 6)
30 tache(6, [6, 7], m2, 26)
31 tache(6, [9, 12], m2, 32)
32 tache(3, [1], m2, 3)
33 tache(6, [7, 8], m2, 22)
34
35 T = [(tache(3, [], ml, 0), tache(8, [], ml, 0), tache(8, [4, 5], ml, 6),
tache(6, [1, m2, 0), tache(3, [1], m2, 3), tache(4, [1, 7], ml, 22),
tache(8, [3, 5], ml, 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))
36 Fin = 38
37
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"

```

1 conflits(Taches) :- dim(Taches,[Dim]),
2 (for(Indice,1,Dim),param(Taches,Dim)
3 do
4 Elem is Taches[Indice],
5 I2 is Indice+1,
6 /* Il faut assurer que les indices soient
différents, sinon on va se retrouver a
comparer deux fois la meme taches*/
7 (for(1,I2,Dim),param(Taches,Elem)
8 do
9 Elem2 is Taches[I1],

```

6

```

10 machinesDifférentes(Elem,Elem2)
11 )
12 ).
13
14 machinesDifférentes(tache(_,_M1,_),tache(_,_M2,_)):- \=(M1,M2),!.
15 machinesDifférentes(tache(Duree,_M,Debut),tache(Duree2,_M,Debut2)) :- ((
Debut #>= Debut2+Duree2) or (Debut+Duree #=< Debut2)).
16
17
18 solve2(Taches,Fin) :- taches(Taches),
19 domaines(Taches,Fin),
20 precedences(Taches),
21 conflits(Taches),
22 getVarList(Taches,Fin,Liste),
23 labeling(Liste),
24 affiche(Taches).
25 /*
26 [eclipse 10]: solve2(Taches,Fin).
27 tache(3, [], ml, 0)
28 tache(8, [1, ml, 29)
29 tache(8, [4, 5], ml, 9)
30 tache(6, [1, m2, 0)
31 tache(3, [1], m2, 6)
32 tache(4, [1, 7], ml, 25)
33 tache(8, [3, 5], ml, 17)
34 tache(6, [4], m2, 12)
35 tache(6, [6, 7], m2, 31)
36 tache(6, [9, 12], m2, 37)
37 tache(3, [1], m2, 9)
38 tache(6, [7, 8], m2, 25)
39
40 Taches = [(tache(3, [], ml, 0), tache(8, [], ml, 29), tache(8, [4, 5], ml,
9), tache(6, [1, m2, 0), tache(3, [1], m2, 6), tache(4, [1, 7], ml, 25),
tache(8, [3, 5], ml, 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))
41 Fin = 43
42 Yes (0.01 s cpu, solution 1, maybe more) ? ;
43 tache(3, [1, ml, 1)
44 tache(8, [1, ml, 29)
45 tache(8, [4, 5], ml, 9)
46 tache(6, [1, m2, 0)
47 tache(3, [1], m2, 6)
48 tache(4, [1, 7], ml, 25)
49 tache(8, [3, 5], ml, 17)
50 tache(6, [4], m2, 12)
51 tache(6, [6, 7], m2, 31)
52 tache(6, [9, 12], m2, 37)
53 tache(3, [1], m2, 9)
54 tache(6, [7, 8], m2, 25)
55
56 Taches = [(tache(3, [1, ml, 1), tache(8, [1, ml, 29), tache(8, [4, 5], ml,
9), tache(6, [1, m2, 0), tache(3, [1], m2, 6), tache(4, [1, 7], ml, 25),

```

TB

```

tache(8, [3, 5], ml, 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))
57 Fin = 43
58 Yes (0.01 s cpu, solution 2, maybe more) ? ;
59 tache(3, [1, ml, 2)
60 tache(8, [1, ml, 29)
61 tache(8, [4, 5], ml, 9)
62 tache(6, [1, m2, 0)
63 tache(3, [1], m2, 6)
64 tache(4, [1, 7], ml, 25)
65 tache(8, [3, 5], ml, 17)
66 tache(6, [4], m2, 12)
67 tache(6, [6, 7], m2, 31)
68 tache(6, [9, 12], m2, 37)
69 tache(3, [1], m2, 9)
70 tache(6, [7, 8], m2, 25)
71
72 Taches = [(tache(3, [1, ml, 2), tache(8, [1, ml, 29), tache(8, [4, 5], ml,
9), tache(6, [1, m2, 0), tache(3, [1], m2, 6), tache(4, [1, 7], ml, 25),
tache(8, [3, 5], ml, 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))
73 Fin = 43
74 Yes (0.01 s cpu, solution 3, maybe more) ?
75 */
76
77 /*
78 [eclipse 11]: solve2(Taches,42).
79
80 No (0.00 s cpu)
81 */

```

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 incrément, 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.

7

8

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,[1],ml,_),
5                                     tache(8,[1],ml,_),
6                                     tache(8,[4,5],ml,_),
7                                     tache(6,[1],m2,_),
8                                     tache(3,[1],m2,_),
9                                     tache(4,[1,7],ml,_),
10                                    tache(8,[3,5],ml,_),
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, [1], ml, _169), tache(8, [1], ml, _176), tache(8, [4, 5], ml,
    _183), tache(6, [1], m2, _194), tache(3, [1], m2, _201), tache(4, [1, 7],
    ml, _210), tache(8, [3, 5], ml, _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                                ],
31                            writeln(Elem)
32                        ).
33
34 /*
35 [eclipse 4]: taches(T), affiche(T).
36 tache(3, [1], ml, _235)
37 tache(8, [1], ml, _242)
38 tache(8, [4, 5], ml, _249)
39 tache(6, [1], m2, _260)
40 tache(3, [1], m2, _267)
41 tache(4, [1, 7], ml, _276)
42 tache(8, [3, 5], ml, _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, [1], ml, _235), tache(8, [1], ml, _242), tache(8, [4, 5], ml,
    _249), tache(6, [1], m2, _260), tache(3, [1], m2, _267), tache(4, [1, 7],
    ml, _276), tache(8, [3, 5], ml, _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,
59                                  _Machine,Debut)
60                              is Taches[Indice
61                                  ],
62                              Debut + Duree #=< Fin
63                              ,
64                              Debut #>= 0
65                          ).
66
67 /*
68 [eclipse 5]: taches(T),domaines(T,10).
69
70 T = [tache(3, [1], ml, _421[0 .. 7]), tache(8, [1], ml, _644[0 .. 2]), tache(8,
    [4, 5], ml, _867[0 .. 2]), tache(6, [1], m2, _1090[0 .. 4]), tache(3,
    [1], m2, _1313[0 .. 7]), tache(4, [1, 7], ml, _1536[0 .. 6]), tache(8,
    [3, 5], ml, _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])]
71
72
73 There are 12 delayed goals. Do you want to see them? (y\N)
74 Yes (0.00s cpu)
75
76 */
77
78 getVarList(Taches,Fin,ListFin):- dim(Taches,[Dim]),
79                                  (for(Indice,1,Dim),fromto([],In,Out,List),
80                                    param(Taches)
81                                    do
82                                      Xi is Taches[Indice],
83                                      Xi = tache(_,_,_),Debut),
84                                      Out=[Debut|In]
85                                  ),
86                                  ListFin=[Fin|List].
87
88 /*
89
90 Fin = 38
91
92 */
93
94 conflits(Taches) :- dim(Taches,[Dim]),
95                    (for(Indice,1,Dim),param(Taches,Dim)
96                      do
97                        Elem is Taches[Indice],
98                        I2 is Indice+1,
99                        /*Il faut assurer que les indices soient
100                        differents , sinon on va se retrouver a
101                        comparer deux fois la meme taches*/
102                        (for(1,I2,Dim),param(Taches,Elem)
103                          do
104                            Elem2 is Taches[I1],
105                            machinesDifferentes(Elem,Elem2)
106                        )
107                    ).
108
109 machinesDifferentes(tache(_,_,_M1,_),tache(_,_,_M2,_)):- \=(M1,M2),!.
110 machinesDifferentes(tache(Duree,_M,Debut),tache(Duree2,_M,Debut2)) :- ((
111     Debut #>= Debut2+Duree2) or (Debut+Duree #=< Debut2)).
112
113 solve2(Taches,Fin) :-      taches(Taches),
114                          domaines(Taches,Fin),
115                          precedences(Taches),
116                          getVarList(Taches,Fin,Liste),
117                          labeling(Liste),
118                          affiche(Taches).
119
120 precedences(Taches) :-      dim(Taches,[Dim]),
121                          (for(Indice,1,Dim),param(Taches)
122                            do
123                              Elem is Taches[Indice],
124                              Elem = tache(_D,Noms,_M,Debut),
125                              (foreach(1,Noms),param(Debut,Taches)
126                                do
127                                  tache(Duree2,_N,_M,Debut2) is Taches[
128                                      ],
129                                  Debut #>= Debut2+Duree2
130                                )
131                            ).
132
133 /*
134 [eclipse 44]: taches(T), solve(T, Fin).
135 tache(3, [1], ml, 0)
136 tache(8, [1], ml, 0)
137 tache(8, [4, 5], ml, 6)
138 tache(6, [1], m2, 0)
139 tache(3, [1], m2, 3)
140 tache(4, [1, 7], ml, 22)
141 tache(8, [3, 5], ml, 14)
142 tache(6, [4], m2, 6)
143 tache(6, [6, 7], m2, 26)
144 tache(6, [9, 12], m2, 32)
145 tache(3, [1], m2, 3)
146 tache(6, [7, 8], m2, 22)
147
148 T = [(tache(3, [1], ml, 0), tache(8, [1], ml, 0), tache(8, [4, 5], ml, 6),
    tache(6, [1], m2, 0), tache(3, [1], m2, 3), tache(4, [1, 7], ml, 22),
    tache(8, [3, 5], ml, 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))
149
150
151 Fin = 43
152 Yes (0.01s cpu, solution 1, maybe more) ? :
153 tache(3, [1], ml, 1)
154 tache(8, [1], ml, 29)
155 tache(8, [4, 5], ml, 9)
156 tache(6, [1], m2, 0)
157 tache(3, [1], m2, 6)
158 tache(4, [1, 7], ml, 25)
159 tache(8, [3, 5], ml, 17)
160 tache(6, [4], m2, 12)
161 tache(6, [6, 7], m2, 31)
162 tache(6, [9, 12], m2, 37)
163 tache(3, [1], m2, 9)
164 tache(6, [7, 8], m2, 25)
165
166 Taches = [(tache(3, [1], ml, 0), tache(8, [1], ml, 29), tache(8, [4, 5], ml,
    9), tache(6, [1], m2, 0), tache(3, [1], m2, 6), tache(4, [1, 7], ml, 25),
    tache(8, [3, 5], ml, 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))
167
168 Fin = 43
169 Yes (0.01s cpu, solution 1, maybe more) ? :
170 tache(3, [1], ml, 1)
171 tache(8, [1], ml, 29)
172 tache(8, [4, 5], ml, 9)
173 tache(6, [1], m2, 0)
174 tache(3, [1], m2, 6)
175 tache(4, [1, 7], ml, 25)
176 tache(8, [3, 5], ml, 17)
177 tache(6, [4], m2, 12)
178 tache(6, [6, 7], m2, 31)
179 tache(6, [9, 12], m2, 37)
180 tache(3, [1], m2, 9)
181 tache(6, [7, 8], m2, 25)
182
183 Taches = [(tache(3, [1], ml, 0), tache(8, [1], ml, 29), tache(8, [4, 5], ml,
    9), tache(6, [1], m2, 0), tache(3, [1], m2, 6), tache(4, [1, 7], ml, 25),
    tache(8, [3, 5], ml, 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))
184
185 Fin = 43
186 Yes (0.01s cpu, solution 1, maybe more) ? :
187 tache(3, [1], ml, 1)
188 tache(8, [1], ml, 29)
189 tache(8, [4, 5], ml, 9)
190 tache(6, [1], m2, 0)
191 tache(3, [1], m2, 6)
192 tache(4, [1, 7], ml, 25)
193 tache(8, [3, 5], ml, 17)
194 tache(6, [4], m2, 12)
195 tache(6, [6, 7], m2, 31)
196 tache(6, [9, 12], m2, 37)
197 tache(3, [1], m2, 9)
198 tache(6, [7, 8], m2, 25)
199
200 Taches = [(tache(3, [1], ml, 0), tache(8, [1], ml, 29), tache(8, [4, 5], ml,
    9), tache(6, [1], m2, 0), tache(3, [1], m2, 6), tache(4, [1, 7], ml, 25),
    tache(8, [3, 5], ml, 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))
201
202 Fin = 43
203 Yes (0.01s cpu, solution 1, maybe more) ? :
204 tache(3, [1], ml, 1)
205 tache(8, [1], ml, 29)
206 tache(8, [4, 5], ml, 9)
207 tache(6, [1], m2, 0)
208 tache(3, [1], m2, 6)
209 tache(4, [1, 7], ml, 25)
210 tache(8, [3, 5], ml, 17)
211 tache(6, [4], m2, 12)
212 tache(6, [6, 7], m2, 31)
213 tache(6, [9, 12], m2, 37)
214 tache(3, [1], m2, 9)
215 tache(6, [7, 8], m2, 25)
216
217 Taches = [(tache(3, [1], ml, 0), tache(8, [1], ml, 29), tache(8, [4, 5], ml,
    9), tache(6, [1], m2, 0), tache(3, [1], m2, 6), tache(4, [1, 7], ml, 25),
    tache(8, [3, 5], ml, 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))
218
219 Fin = 43
220 Yes (0.01s cpu, solution 1, maybe more) ? :
221 tache(3, [1], ml, 1)
222 tache(8, [1], ml, 29)
223 tache(8, [4, 5], ml, 9)
224 tache(6, [1], m2, 0)
225 tache(3, [1], m2, 6)
226 tache(4, [1, 7], ml, 25)
227 tache(8, [3, 5], ml, 17)
228 tache(6, [4], m2, 12)
229 tache(6, [6, 7], m2, 31)
230 tache(6, [9, 12], m2, 37)
231 tache(3, [1], m2, 9)
232 tache(6, [7, 8], m2, 25)
233
234 Taches = [(tache(3, [1], ml, 0), tache(8, [1], ml, 29), tache(8, [4, 5], ml,
    9), tache(6, [1], m2, 0), tache(3, [1], m2, 6), tache(4, [1, 7], ml, 25),
    tache(8, [3, 5], ml, 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))
235
236 Fin = 43
237 Yes (0.01s cpu, solution 1, maybe more) ? :
238 tache(3, [1], ml, 1)
239 tache(8, [1], ml, 29)
240 tache(8, [4, 5], ml, 9)
241 tache(6, [1], m2, 0)
242 tache(3, [1], m2, 6)
243 tache(4, [1, 7], ml, 25)
244 tache(8, [3, 5], ml, 17)
245 tache(6, [4], m2, 12)
246 tache(6, [6, 7], m2, 31)
247 tache(6, [9, 12], m2, 37)
248 tache(3, [1], m2, 9)
249 tache(6, [7, 8], m2, 25)
250
251 Taches = [(tache(3, [1], ml, 0), tache(8, [1], ml, 29), tache(8, [4, 5], ml,
    9), tache(6, [1], m2, 0), tache(3, [1], m2, 6), tache(4, [1, 7], ml, 25),
    tache(8, [3, 5], ml, 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))
252
253 Fin = 43
254 Yes (0.01s cpu, solution 1, maybe more) ? :
255 tache(3, [1], ml, 1)
256 tache(8, [1], ml, 29)
257 tache(8, [4, 5], ml, 9)
258 tache(6, [1], m2, 0)
259 tache(3, [1], m2, 6)
260 tache(4, [1, 7], ml, 25)
261 tache(8, [3, 5], ml, 17)
262 tache(6, [4], m2, 12)
263 tache(6, [6, 7], m2, 31)
264 tache(6, [9, 12], m2, 37)
265 tache(3, [1], m2, 9)
266 tache(6, [7, 8], m2, 25)
267
268 Taches = [(tache(3, [1], ml, 0), tache(8, [1], ml, 29), tache(8, [4, 5], ml,
    9), tache(6, [1], m2, 0), tache(3, [1], m2, 6), tache(4, [1, 7], ml, 25),
    tache(8, [3, 5], ml, 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))
269
270 Fin = 43
271 Yes (0.01s cpu, solution 1, maybe more) ? :
272 tache(3, [1], ml, 1)
273 tache(8, [1], ml, 29)
274 tache(8, [4, 5], ml, 9)
275 tache(6, [1], m2, 0)
276 tache(3, [1], m2, 6)
277 tache(4, [1, 7], ml, 25)
278 tache(8, [3, 5], ml, 17)
279 tache(6, [4], m2, 12)
280 tache(6, [6, 7], m2, 31)
281 tache(6, [9, 12], m2, 37)
282 tache(3, [1], m2, 9)
283 tache(6, [7, 8], m2, 25)
284
285 Taches = [(tache(3, [1], ml, 0), tache(8, [1], ml, 29), tache(8, [4, 5], ml,
    9), tache(6, [1], m2, 0), tache(3, [1], m2, 6), tache(4, [1, 7], ml, 25),
    tache(8, [3, 5], ml, 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))
286
287 Fin = 43
288 Yes (0.01s cpu, solution 1, maybe more) ? :
289 tache(3, [1], ml, 1)
290 tache(8, [1], ml, 29)
291 tache(8, [4, 5], ml, 9)
292 tache(6, [1], m2, 0)
293 tache(3, [1], m2, 6)
294 tache(4, [1, 7], ml, 25)
295 tache(8, [3, 5], ml, 17)
296 tache(6, [4], m2, 12)
297 tache(6, [6, 7], m2, 31)
298 tache(6, [9, 12], m2, 37)
299 tache(3, [1], m2, 9)
300 tache(6, [7, 8], m2, 25)
301
302 Taches = [(tache(3, [1], ml, 0), tache(8, [1], ml, 29), tache(8, [4, 5], ml,
    9), tache(6, [1], m2, 0), tache(3, [1], m2, 6), tache(4, [1, 7], ml, 25),
    tache(8, [3, 5], ml, 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))
303
304 Fin = 43
305 Yes (0.01s cpu, solution 1, maybe more) ? :
306 tache(3, [1], ml, 1)
307 tache(8, [1], ml, 29)
308 tache(8, [4, 5], ml, 9)
309 tache(6, [1], m2, 0)
310 tache(3, [1], m2, 6)
311 tache(4, [1, 7], ml, 25)
312 tache(8, [3, 5], ml, 17)
313 tache(6, [4], m2, 12)
314 tache(6, [6, 7], m2, 31)
315 tache(6, [9, 12], m2, 37)
316 tache(3, [1], m2, 9)
317 tache(6, [7, 8], m2, 25)
318
319 Taches = [(tache(3, [1], ml, 0), tache(8, [1], ml, 29), tache(8, [4, 5], ml,
    9), tache(6, [1], m2, 0), tache(3, [1], m2, 6), tache(4, [1, 7], ml, 25),
    tache(8, [3, 5], ml, 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))
320
321 Fin = 43
322 Yes (0.01s cpu, solution 1, maybe more) ? :
323 tache(3, [1], ml, 1)
324 tache(8, [1], ml, 29)
325 tache(8, [4, 5], ml, 9)
326 tache(6, [1], m2, 0)
327 tache(3, [1], m2, 6)
328 tache(4, [1, 7], ml, 25)
329 tache(8, [3, 5], ml, 17)
330 tache(6, [4], m2, 12)
331 tache(6, [6, 7], m2, 31)
332 tache(6, [9, 12], m2, 37)
333 tache(3, [1], m2, 9)
334 tache(6, [7, 8], m2, 25)
335
336 Taches = [(tache(3, [1], ml, 0), tache(8, [1], ml, 29), tache(8, [4, 5], ml,
    9), tache(6, [1], m2, 0), tache(3, [1], m2, 6), tache(4, [1, 7], ml, 25),
    tache(8, [3, 5], ml, 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))
337
338 Fin = 43
339 Yes (0.01s cpu, solution 1, maybe more) ? :
340 tache(3, [1], ml, 1)
341 tache(8, [1], ml, 29)
342 tache(8, [4, 5], ml, 9)
343 tache(6, [1], m2, 0)
344 tache(3, [1], m2, 6)
345 tache(4, [1, 7], ml, 25)
346 tache(8, [3, 5], ml, 17)
347 tache(6, [4], m2, 12)
348 tache(6, [6, 7], m2, 31)
349 tache(6, [9, 12], m2, 37)
350 tache(3, [1], m2, 9)
351 tache(6, [7, 8], m2, 25)
352
353 Taches = [(tache(3, [1], ml, 0), tache(8, [1], ml, 29), tache(8, [4, 5], ml,
    9), tache(6, [1], m2, 0), tache(3, [1], m2, 6), tache(4, [1, 7], ml, 25),
    tache(8, [3, 5], ml, 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))
354
355 Fin = 43
356 Yes (0.01s cpu, solution 1, maybe more) ? :
357 tache(3, [1], ml, 1)
358 tache(8, [1], ml, 29)
359 tache(8, [4, 5], ml, 9)
360 tache(6, [1], m2, 0)
361 tache(3, [1], m2, 6)
362 tache(4, [1, 7], ml, 25)
363 tache(8, [3, 5], ml, 17)
364 tache(6, [4], m2, 12)
365 tache(6, [6, 7], m2, 31)
366 tache(6, [9, 12], m2, 37)
367 tache(3, [1], m2, 9)
368 tache(6, [7, 8], m2, 25)
369
370 Taches = [(tache(3, [1], ml, 0), tache(8, [1], ml, 29), tache(8, [4, 5], ml,
    9), tache(6, [1], m2, 0), tache(3, [1], m2, 6), tache(4, [1, 7], ml, 25),
    tache(8, [3, 5], ml, 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))
371
372 Fin = 43
373 Yes (0.01s cpu, solution 1, maybe more) ? :
374 tache(3, [1], ml, 1)
375 tache(8, [1], ml, 29)
376 tache(8, [4, 5], ml, 9)
377 tache(6, [1], m2, 0)
378 tache(3, [1], m2, 6)
379 tache(4, [1, 7], ml, 25)
380 tache(8, [3, 5], ml, 17)
381 tache(6, [4], m2, 12)
382 tache(6, [6, 7], m2, 31)
383 tache(6, [9, 12], m2, 37)
384 tache(3, [1], m2, 9)
385 tache(6, [7, 8], m2, 25)
386
387 Taches = [(tache(3, [1], ml, 0), tache(8, [1], ml, 29), tache(8, [4, 5], ml,
    9), tache(6, [1], m2, 0), tache(3, [1], m2, 6), tache(4, [1, 7], ml, 25),
    tache(8, [3, 5], ml, 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))
388
389 Fin = 43
390 Yes (0.01s cpu, solution 1, maybe more) ? :
391 tache(3, [1], ml, 1)
392 tache(8, [1], ml, 29)
393 tache(8, [4, 5], ml, 9)
394 tache(6, [1], m2, 0)
395 tache(3, [1], m2, 6)
396 tache(4, [1, 7], ml, 25)
397 tache(8, [3, 5], ml, 17)
398 tache(6, [4], m2, 12)
399 tache(6, [6, 7], m2, 31)
400 tache(6, [9, 12], m2, 37)
401 tache(3, [1], m2, 9)
402 tache(6, [7, 8], m2, 25)
403
404 Taches = [(tache(3, [1], ml, 0), tache(8, [1], ml, 29), tache(8, [4, 5], ml,
    9), tache(6, [1], m2, 0), tache(3, [1], m2, 6), tache(4, [1, 7], ml, 25),
    tache(8, [3, 5], ml, 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))
405
406 Fin = 43
407 Yes (0.01s cpu, solution 1, maybe more) ? :
408 tache(3, [1], ml, 1)
409 tache(8, [1], ml, 29)
410 tache(8, [4, 5], ml, 9)
411 tache(6, [1], m2, 0)
412 tache(3, [1], m2, 6)
413 tache(4, [1, 7], ml, 25)
414 tache(8, [3, 5], ml, 17)
415 tache(6, [4], m2, 12)
416 tache(6, [6, 7], m2, 31)
417 tache(6, [9, 12], m2, 37)
418 tache(3, [1], m2, 9)
419 tache(6, [7, 8], m2, 25)
420
421 Taches = [(tache(3, [1], ml, 0), tache(8, [1], ml, 29), tache(8, [4, 5], ml,
    9), tache(6, [1], m2, 0), tache(3, [1], m2, 6), tache(4, [1, 7], ml, 25),
    tache(8, [3, 5], ml, 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))
422
423 Fin = 43
424 Yes (0.01s cpu, solution 1, maybe more) ? :
425 tache(3, [1], ml, 1)
426 tache(8, [1], ml, 29)
427 tache(8, [4, 5], ml, 9)
428 tache(6, [1], m2, 0)
429 tache(3, [1], m2, 6)
430 tache(4, [1, 7], ml, 25)
431 tache(8, [3, 5], ml, 17)
432 tache(6, [4], m2, 12)
433 tache(6, [6, 7], m2, 31)
434 tache(6, [9, 12], m2, 37)
435 tache(3, [1], m2, 9)
436 tache(6, [7, 8], m2, 25)
437
438 Taches = [(tache(3, [1], ml, 0), tache(8, [1], ml, 29), tache(8, [4, 5], ml,
    9), tache(6, [1], m2, 0), tache(3, [1], m2, 6), tache(4, [1, 7], ml, 25),
    tache(8, [3, 5], ml, 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))
439
440 Fin = 43
441 Yes (0.01s cpu, solution 1, maybe more) ? :
442 tache(3, [1], ml, 1)
443 tache(8, [1], ml, 29)
444 tache(8, [4, 5], ml, 9)
445 tache(6, [1], m2, 0)
446 tache(3, [1], m2, 6)
447 tache(4, [1, 7], ml, 25)
448 tache(8, [3, 5], ml, 17)
449 tache(6, [4], m2, 12)
450 tache(6, [6, 7], m2, 31)
451 tache(6, [9, 12], m2, 37)
452 tache(3, [1], m2, 9)
453 tache(6, [7, 8], m2, 25)
454
455 Taches = [(tache(3, [1], ml, 0), tache(8, [1], ml, 29), tache(8, [4, 5], ml,
    9), tache(6, [1], m2, 0), tache(3, [1], m2, 6), tache(4, [1, 7], ml, 25),
    tache(8, [3, 5], ml, 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))
456
457 Fin = 43
458 Yes (0.01s cpu, solution 1, maybe more) ? :
459 tache(3, [1], ml, 1)
460 tache(8, [1], ml, 29)
461 tache(8, [4, 5], ml, 9)
462 tache(6, [1], m2, 0)
463 tache(3, [1], m2, 6)
464 tache(4, [1, 7], ml, 25)
465 tache(8, [3, 5], ml, 17)
466 tache(6, [4], m2, 12)
467 tache(6, [6, 7], m2, 31)
468 tache(6, [9, 12], m2, 37)
469 tache(3, [1], m2, 9)
470 tache(6, [7, 8], m2, 25)
471
472 Taches = [(tache(3, [1], ml, 0), tache(8, [1], ml, 29), tache(8, [4, 5], ml,
    9), tache(6, [1], m2, 0), tache(3, [1], m2, 6), tache(4, [1, 7], ml, 25),
    tache(8, [3, 5], ml, 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))
473
474 Fin = 43
475 Yes (0.01s cpu, solution 1, maybe more) ? :
476 tache(3, [1], ml, 1)
477 tache(8, [1], ml, 29)
478 tache(8, [4, 5], ml, 9)
479 tache(6, [1], m2, 0)
480 tache(3, [1], m2, 6)
481 tache(4, [1, 7], ml, 25)
482 tache(8, [3, 5], ml, 17)
483 tache(6, [4], m2, 12)
484 tache(6, [6, 7], m2, 31)
485 tache(6, [9, 12], m2, 37)
486 tache(3, [1], m2, 9)
487 tache(6, [7, 8], m2, 25)
488
489 Taches = [(tache(3, [1], ml, 0), tache(8, [1], ml, 29), tache(8, [4, 5], ml,
    9), tache(6, [1], m2, 0), tache(3, [1], m2, 6), tache(4, [1, 7], ml, 25),
    tache(8, [3, 5], ml, 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,
```

```

173  tache(8, [4, 5], m1, 9)
174  tache(6, [], m2, 0)
175  tache(3, [1], m2, 6)
176  tache(4, [1, 7], m1, 25)
177  tache(8, [3, 5], m1, 17)
178  tache(6, [4], m2, 12)
179  tache(6, [6, 7], m2, 31)
180  tache(6, [9, 12], m2, 37)
181  tache(3, [1], m2, 9)
182  tache(6, [7, 8], m2, 25)
183
184  Taches = [(tache(3, [], m1, 1), tache(8, [], m1, 29), tache(8, [4, 5], m1,
9), tache(6, [1], 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))
185  Fin = 43
186  Yes (0.01s cpu, solution 2, maybe more) ? ;
187  tache(3, [], m1, 2)
188  tache(8, [], m1, 29)
189  tache(8, [4, 5], m1, 9)
190  tache(6, [], m2, 0)
191  tache(3, [1], m2, 6)
192  tache(4, [1, 7], m1, 25)
193  tache(8, [3, 5], m1, 17)
194  tache(6, [4], m2, 12)
195  tache(6, [6, 7], m2, 31)
196  tache(6, [9, 12], m2, 37)
197  tache(3, [1], m2, 9)
198  tache(6, [7, 8], m2, 25)
199
200  Taches = [(tache(3, [], m1, 2), tache(8, [], m1, 29), tache(8, [4, 5], m1,
9), tache(6, [1], 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))
201  Fin = 43
202  Yes (0.01s cpu, solution 3, maybe more) ?
203  */
204
205  /*
206  [eclipse 11]: solve2(Taches,42).
207
208  No (0.00s cpu)
209  */

```

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 régates, planning qui respecte certaines contraintes.

Question 4.1 Nous définissons ici un prédicat `getData(?TailleEquipes, ?NbEquipes, ?CapaBateaux, ?NbBateaux, :` 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 contraind le domaine des variables.

Listing 2 – "defineVars"

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

1

```
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,
13   NbConf, NbBat).
14
15 TailleEq = [(5, 5, 2, 1)
16 NbEq = 4
17 CapaBat = [(7, 6, 5)
18 NbBat = 3
19 T = [( [(419[1..3], 488[1..3], 557[1..3]), [(628[1..3], 697[1
20   ..3], 766[1..3]), [(837[1..3], 906[1..3], 975[1..3]), [(
21   1046[1..3], 1115[1..3], 1184[1..3])]
```

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

Bien

2

```
21 T = [( [(484[1..3], 553[1..3], 622[1..3]), [(693[1..3], 762[1
22   ..3], 831[1..3]), [(902[1..3], 971[1..3], 1040[1..3]), [(
23   1111[1..3], 1180[1..3], 1249[1..3])]
```

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 – "solve"

```
1 solve(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]: solve(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     )
7     do
8       Bat is T[Indice1, Indice2],
9       append(In, [Bat], Out)
10    ),
11    alldifferent(L)
12  ).
```

Bien

3

```
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
10    (0, In, Out, Tot)
11    do
12      Bateau1 is T[Equipe1, Conf],
13      Bateau2 is T[Equipe2, Conf],
14      #=(Bateau1, Bateau2, Ans),
15      Out #= In + Ans
16    ),
17    Tot #=< 1
18  ),
19  ).
20
21 solve3(T) :-
22   getData(_TailleEquipes, _NbEquipes, _CapaBateaux, _NbBateaux, _NbConf),
23   defineVars(T, NbEquipes, NbConf, NbBateaux),
24   pasMemeBateaux(T, NbEquipes, NbConf),
25   pasMemePartenaires(T, NbEquipes, NbConf),
26   getVarList(T, L),
27   labeling(L).
```

Bien

4


```

31 T =  $\{[(\{[(1, 2, 3)], [(2, 3, 1)], [(3, 1, 2)], [(3, 2, 1)]\})]$ 
32 Yes (0.01s cpu, solution 1, maybe more) ? ;
33
34 T =  $\{[(\{[(1, 3, 2)], [(2, 1, 3)], [(3, 2, 1)], [(1, 3, 2)]\})]$ 
35 Yes (0.01s cpu, solution 2, maybe more) ? ;
36
37 T =  $\{[(\{[(1, 2, 3)], [(3, 1, 2)], [(2, 3, 1)], [(1, 1, 3, 2)]\})]$ 
38 Yes (0.01s cpu, solution 3, maybe more) ?
39 */

```

Question 4.8 On passe ici à un problème de taille réelle. On dispose dorénavant de 13 voliers et de 29 équipes qui doivent effectuer la régaté comportant 7 confrontations. Le temps d'exécution étant relativement long, il nous a été proposé d'améliorer le labeling. Pour cela, nous avons mélangé la liste des variables 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 labelling"
1 getData2(TailleEquipes ,NbEquipes ,CapaBateaux ,NbBateaux ,NbConf):-
2     TailleEquipes =
3         [[(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) ],
4             NbEquipes = 29,
5             CapaBateaux = [(10,10,9,8,8,8,8,8,8,7,6,4,4) ],
6             NbBateaux = 13,
7             NbConf = 7.
8
9 solve5(T):-
10    getData2(TailleEq,NbEq,CapaBat,NbBat,NbConf),
11    defineVars(T,NbEq,NbConf,NbBat),
12    pasMemeBateaux(T,NbEq,NbConf),
13    pasMemePartenaires(T,NbEq,NbConf),
14    capabateaux(T,TailleEq,NbEq,CapaBat,NbBat,NbConf),
15    getVarList(T,L),
16    labeling(L).
17
18 /* Tests
19 [eclipse 14]: solve5(T).
20
21 T = [[[(1, 2, 3, 4, 5, 6, 7)], [(2, 1, 4, 3, 6, 5, 8)], [(3, 4, 1, 2, 7, 8,
22      5)], [(4, 3, 1, 5, 2, 7, 6)], [(5, 6, 2, 1, 3, 4, 9)], [(2, 3, 5, 1, 4,
23      9, 10)], [(3, 2, 6, 4, 10, 11)], [(6, 5, 7, 2, 1, 3, 4)], [(6, 7, 5,
24      8, 2, 1, 3)], [(7, 5, 6, 8, 3, 9)], [(8, 9, 6, 1, 11, 2)], [(8, 7,
25      6, 9, 10, 12, 1)], [(8, 9, 10, 11, 1, 12)], [(1, 4, 8, 3, 9, 7, 10)],
26      [(4, 2, 8, 10, 7, 3, 9)], [(5, 8, 3, 11, 6, 9, 1)], [(9, 6, 4, 5, 8, 11,
27      13)], [(9, 8, 10, 12, 13, 2, 11)], [(9, 10, 11, 12, 13, 2)], [(9, 11,
28      12, 13, 1, 10, 3)], [(10, 9, 11, 7, 12, 3, 13)], [(10, 11, 9, 12, 8, 1,
29      4)], [(10, 12, 13, 11, 9, 2, 3)], [(11, 9, 10, 13, 8, 4, 5)], [(11, 10,
30      12, 9, 13, 5, 1)], [(11, 12, 9, 7, 10, 13, 8)], [(12, 10, 7, 11, 9,
31      13)], [(12, 13, 11, 9, 8, 2, 6)], [(13, 11, 10, 7, 9, 8, 1)])]
32
33 Yes (.5523s cpu, solution 1, maybe more) ?
34 */

```

```

54 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))
65 Yes (7.31s cpu, solution 2, maybe more) ?
66 */

```

1 Code Complet, avec l'ensemble des tests

Listing 9 – "TP4"

```

1 :-lib(ic).
2
3 % Q4.1
4 getData(TailleEquipes ,NbEquipes ,CapaBateaux ,NbBateaux ,NbConf):-
5     TailleEquipes = [(5,5,2,1) ,
6         NbEquipes = 4 ,
7         CapaBateaux = [(7,6,5) ,
8             NbBateaux = 3 ,
9             NbConf = 3.
10
11 /* Tests
12 [eclipse 7]: getData(TailleEq ,NbEq ,CapaBat ,NbBat ,NbConf) .
13
14 TailleEq = [(5, 5, 2, 1)
15 NbEq = 4
16 CapaBat = [(7, 6, 5)
17 NbBat = 3
18 NbConf = 3
19 Yes (0.00s cpu)
20 */
21
22 % Q4.2
23 defineVars(T,NbEquipes ,NbConf ,NbBateaux):-
24     dim(T,[NbEquipes ,NbConf]) ,
25     ( for(Ind1,1,NbEquipes) ,param(T,NbBateaux ,NbConf)
26         do
27             ( for(Ind2,1,NbConf) ,param(T,Ind1 ,NbBateaux)
28                 do
29                     T[Ind1,Ind2] #:: 1..NbBateaux
30                 )
31             )
32
33 /* Tests
34 [eclipse 8]: getData(TailleEq ,NbEq ,CapaBat ,NbBat ,NbConf) ,defineVars(T,NbEq ,
35     NbConf,NbBat) .
36
37 TailleEq = [(5, 5, 2, 1)
38 NbEq = 4
39 CapaBat = [(7, 6, 5)
40 NbBat = 3
41 NbConf = 3
42 T = [([(1,419[1 .. 3],_488[1 .. 3],_557[1 .. 3]),[(1,628[1 .. 3],_697[1
43     .. 3],_766[1 .. 3]),[(1,837[1 .. 3],_906[1 .. 3],_975[1 .. 3]),[(1,
44     _1046[1 .. 3],_1115[1 .. 3],_1184[1 .. 3])])
45 Yes (0.00s cpu)
46 */
47
48 % Q4.3
49 getVarList(T,L):-

```

```

47     dim(T,[NbEquipes ,NbConf]) ,
48     ( for(Indice1,1,NbConf) ,fromto([],In ,Out ,L) ,param(T,NbEquipes)
49         do
50             ( for(Indice2 ,1,NbEquipes) ,fromto([],In2 ,Out2 ,L2) ,param(T,Indice1)
51                 do
52                     Var is T[Indice2 ,Indice1] ,
53                     append(In2 ,[Var] ,Out2)
54                 ),
55                 append(In ,L2 ,Out)
56             )
57
58 /* Tests
59 [eclipse 9]: getData(TailleEq ,NbEq ,CapaBat ,NbBat ,NbConf) ,defineVars(T,NbEq ,
60     NbConf,NbBat) ,getVarList(T,L) .
61
62 TailleEq = [(5, 5, 2, 1)
63 NbEq = 4
64 CapaBat = [(7, 6, 5)
65 NbBat = 3
66 NbConf = 3
67 T = [([(1,484[1 .. 3],_553[1 .. 3],_622[1 .. 3]),[(1,693[1 .. 3],_762[1
68     .. 3],_831[1 .. 3]),[(1,902[1 .. 3],_971[1 .. 3],_1040[1 .. 3]),[(1,
69     _1111[1 .. 3],_1180[1 .. 3],_1249[1 .. 3])])
70 L = [(1,484[1 .. 3],_693[1 .. 3],_902[1 .. 3],_1111[1 .. 3],_553[1 .. 3],
71     _762[1 .. 3],_971[1 .. 3],_1180[1 .. 3],_622[1 .. 3],_831[1 .. 3],
72     _1040[1 .. 3],_1249[1 .. 3])])
73 Yes (0.00s cpu)
74 */
75
76 % Q4.4
77 solve1(T) :-
78     getData(_TailleEquipes ,NbEquipes ,_CapaBateaux ,NbBateaux ,NbConf) ,
79     defineVars(T,NbEquipes ,NbConf ,NbBateaux) ,
80     getVarList(T,L) ,
81     labeling(L) .
82
83 /* Tests
84 [eclipse 10]: solve1(T) .
85
86 T = [([(1, 1, 1), [(1, 1, 1), [(1, 1, 1), [(1, 1, 1)
87 Yes (0.00s cpu, solution 1, maybe more) ? ;
88
89 T = [([(1, 1, 1), [(1, 1, 1), [(1, 1, 1), [(1, 1, 2)
90 Yes (0.00s cpu, solution 2, maybe more) ? ;
91
92 T = [([(1, 1, 1), [(1, 1, 1), [(1, 1, 1), [(1, 1, 3)
93 Yes (0.00s cpu, solution 3, maybe more) ? ;
94
95 % Q4.5
96 pasMemeBateaux(T,NbEquipes ,NbConf):-

```

```

94     dim(T,[NbEquipes ,NbConf]) ,
95     ( for(Indice1,1,NbEquipes) ,param(T,NbConf)
96         do
97             ( for(Indice2 ,1,NbConf) ,fromto([],In ,Out ,L) ,param(T,Indice1)
98                 do
99                     Bat is T[Indice1,Indice2] ,
100                     append(In ,[Bat] ,Out)
101                 ),
102                 alldifferent(L)
103             )
104
105 solve2(T) :-
106     getData(_TailleEquipes ,NbEquipes ,_CapaBateaux ,NbBateaux ,NbConf) ,
107     defineVars(T,NbEquipes ,NbConf ,NbBateaux) ,
108     pasMemeBateaux(T,NbEquipes ,NbConf) ,
109     getVarList(T,L) ,
110     labeling(L) .
111
112 /* Tests
113 [eclipse 11]: solve2(T) .
114
115 T = [([(1, 2, 3), [(1, 2, 3), [(1, 2, 3), [(1, 2, 3)
116 Yes (0.00s cpu, solution 1, maybe more) ? ;
117
118 T = [([(1, 1, 2, 3), [(1, 2, 3), [(1, 2, 3), [(1, 3, 2)
119 Yes (0.00s cpu, solution 2, maybe more) ? ;
120
121 T = [([(1, 1, 2, 3), [(1, 2, 3), [(1, 3, 2), [(1, 2, 3)
122 Yes (0.00s cpu, solution 3, maybe more) ? ;
123
124 % Q4.6
125 pasMemePartenaires(T,NbEquipes ,NbConf):-
126     dim(T,[NbEquipes ,NbConf]) ,
127     ( for(Equipe1,1,NbEquipes) ,param(T,NbConf ,NbEquipes)
128         do
129             Indice is Equipe1 +1 ,
130             ( for(Equipe2,Indice ,NbEquipes) ,param(T,Equipe1 ,NbConf)
131                 do
132                     ( for(Conf,1,NbConf) ,param(T,Equipe1 ,Equipe2) ,fromto
133                         (0,In ,Out ,Tot)
134                             do
135                                 Bateau1 is T[Equipe1 ,Conf] ,
136                                 Bateau2 is T[Equipe2 ,Conf] ,
137                                 #=(Bateau1 ,Bateau2 ,Ans) ,
138                                 Out #= In + Ans
139                             )
140                         )
141                     Tot #=< 1
142                 )
143             )
144

```

```

145 solve3(T) :-
146     getData(_TailleEquipes ,NbEquipes ,_CapaBateaux ,NbBateaux ,NbConf) ,
147     defineVars(T,NbEquipes ,NbConf ,NbBateaux) ,
148     pasMemeBateaux(T,NbEquipes ,NbConf) ,
149     pasMemePartenaires(T,NbEquipes ,NbConf) ,
150     getVarList(T,L) ,
151     labeling(L) .
152
153 /* Tests
154 [eclipse 12]: solve3(T) .
155
156 T = [([(1, 2, 3), [(1, 3, 2), [(2, 1, 3), [(2, 3, 1)
157 Yes (0.00s cpu, solution 1, maybe more) ? ;
158
159 T = [([(1, 1, 2, 3), [(1, 3, 2), [(2, 3, 1), [(2, 1, 3)
160 Yes (0.00s cpu, solution 2, maybe more) ? ;
161
162 T = [([(1, 1, 3, 2), [(1, 2, 3), [(2, 1, 3), [(2, 3, 1)
163 Yes (0.00s cpu, solution 3, maybe more) ? ;
164
165 % Q4.7
166 capabateaux(T,TailleEquipes ,NbEquipes ,CapaBateaux ,NbBateaux ,NbConf):-
167     dim(T,[NbEquipes ,NbConf]) ,
168     ( for(Bateau ,1,NbBateaux) ,param(T,NbEquipes ,NbConf ,CapaBateaux ,
169         TailleEquipes)
170         do
171             ( for(Conf,1,NbConf) ,param(T,NbEquipes ,Bateau ,CapaBateaux ,
172                 TailleEquipes)
173                 do
174                     ( for(Equipe ,1,NbEquipes) ,param(T,Bateau ,Conf ,
175                         TailleEquipes) ,fromto(0,In ,Out ,Total)
176                         do
177                             Bateau1 is T[Equipe ,Conf] ,
178                             #=(Bateau ,Bateau1 ,Cond) ,
179                             Inc #= TailleEquipes[Equipe] * Cond ,
180                             Out #= In + Inc
181                         ),
182                         Capacite is CapaBateaux[Bateau] ,
183                         Total #=< Capacite
184                     )
185                 )
186
187 solve4(T) :-
188     getData(TailleEquipes ,NbEquipes ,CapaBateaux ,NbBateaux ,NbConf) ,
189     defineVars(T,NbEquipes ,NbConf ,NbBateaux) ,
190     pasMemeBateaux(T,NbEquipes ,NbConf) ,
191     pasMemePartenaires(T,NbEquipes ,NbConf) ,
192     capabateaux(T,TailleEquipes ,NbEquipes ,CapaBateaux ,NbBateaux ,NbConf) ,
193     getVarList(T,L) ,
194     labeling(L) .
195
196 /* Tests

```

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))

273 Yes (7.31s cpu, solution 2, maybe more) ?

274 */

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 = [(4,5,8,5,6,4,7,10,11).
7
8 defineVars(Fabriquer, NbSortes):-
```

1

Listing 4 – Prédicat test avec minimize

```
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 étéinstanciées.

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

```
1 test(X,Y,Z,W) :-
2   [X,Y,Z,W] #:: [0..10],
3   X #= Z+Y+2*W,
4   X#1= 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
18 Y = 0
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 minimiser, et l'on pose comme contrainte qu'elle soit égale à l'opposé du profit. On obtient ainsi le profit maximum.

3

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

OK

OK

OK

Question 5.3 On a défini le prédicat pose_contraintes comme une succession 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

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 -1.01Inf ... -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 seul pour le profit, et minimiser le nombre d'employés.

Listing 6 – "Prédicat pose_contrainte2 avec seul 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 -1.01Inf ... 6
18
```

OK

4

```

19 F = [(1, 0, 1, 0, 0, 0, 0, 0)]
20 N = 7
21 P = 1040
22 Yes (0.00s cpu)
23 */

```

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.

5

```

47 ProfitTotal #:= sum(VectBenefTotal).
48
49 pose_contraintes(Fabriqueur, NbTechniciensTotal, Profit) :-
50   getData(NbTech, NbSor, Tech, Quan, Bene),
51   defineVars(Fabriqueur, NbSor),
52   nbOuvriersNecessaires(Fabriqueur, Tech, NbSor, NbTechniciensTotal),
53   NbTechniciensTotal #=< NbTech,
54   getVarList(Fabriqueur, NbSor, L),
55   profitTotal(Fabriqueur, Bene, Quan, NbSor, Profit),
56   labeling(L).
57
58 test(X,Y,Z,W) :-
59   [X,Y,Z,W] #:: [0..10],
60   X #= Z+Y+2*W,
61   X#|= Z+Y+W.
62
63 /* Question 5.4
64 Il faut toujours faire un labeling dans le but,
65 car il faut que les variables aient été instanciées.
66
67 [eclipse 53]: minimize(test(X,Y,Z,W),X).
68 bb_min: search did not instantiate cost variable
69 Aborting execution ...
70 Abort
71
72 Avec le labeling sur [X,Y,Z,W], cela fonctionne. Prolog trouve les réponses
73
74 [eclipse 57]: test(X,Y,Z,W),labeling([X,Y,Z,W]).
75
76 X = 2
77 Y = 0
78 Z = 0
79 W = 1
80 Yes (0.00s cpu, solution 1, maybe more) ? ;
81
82 X = 3
83 Y = 0
84 Z = 1
85 W = 1
86 Yes (0.00s cpu, solution 2, maybe more) ? ;
87
88
89 /* Test
90
91 [eclipse 71]: P2#=-1, minimize(pose_contraintes(F,N,P),P2).
92 Found a solution with cost 0
93 Found a solution with cost -495
94 Found a solution with cost -795
95 Found a solution with cost -1195
96 Found a solution with cost -1495
97 Found a solution with cost -1535
98 Found a solution with cost -1835
99 Found a solution with cost -1955

```

7

4 Code Complet, avec l'ensemble des tests

Listing 7 – "TP5"

```

1 %TP5
2
3 :- lib(ic).
4 :- lib(branch_and_bound).
5
6 getData(NbTechniciens, NbSortes, Techniciens, Quantite, Benefice) :-
7   NbTechniciens = 22,
8   NbSortes = 9,
9   Techniciens = [(5,7,2,6,9,3,7,5,3)],
10  Quantite = [(140,130,60,95,70,85,100,30,45)],
11  Benefice = [(4,5,8,5,6,4,7,10,11)].
12
13 defineVars(Fabriqueur, NbSortes):-
14   dim(Fabriqueur,[NbSortes]),
15   ( for(I,1,NbSortes),param(Fabriqueur)
16     do
17       Fabriqueur[I] #:: 0..1
18     ).
19
20 getVarList(Fabriqueur, NbSortes, L) :-
21   ( for(Ind,1,NbSortes),fromto([],In,Out,L),param(Fabriqueur)
22     do
23       Var #= Fabriqueur[Ind],
24       append(In,[Var],Out)
25     ).
26
27 nbOuvriersNecessaires(Fabriqueur, Techniciens, NbSortes, NbOuvriersTotal) :-
28   ( for(J,1,NbSortes),fromto(0,In,Out,NbOuvriersTotal),param(Fabriqueur,
29     Techniciens)
30     do
31       Var is Fabriqueur[J],
32       NbTech is Techniciens[J],
33       Out #= In + NbTech * Var
34     ).
35 vectBeneficeTotal(Fabriqueur, Benefice, Quantite, NbSortes, VectBenefTotal) :-
36   ( for(K,1,NbSortes),fromto([],In,Out,VectBenefTotal),param(Fabriqueur,
37     Benefice, Quantite)
38     do
39       Var is Fabriqueur[K],
40       Benef is Benefice[K],
41       Quan is Quantite[K],
42       Elem #= Var * Benef * Quan,
43       append(In,[Elem],Out)
44     ).
45 profitTotal(Fabriqueur, Benefice, Quantite, NbSortes, ProfitTotal) :-
46   vectBeneficeTotal(Fabriqueur, Benefice, Quantite, NbSortes, VectBenefTotal),

```

6

```

100 Found a solution with cost -1970
101 Found a solution with cost -2010
102 Found a solution with cost -2015
103 Found a solution with cost -2315
104 Found a solution with cost -2490
105 Found a solution with cost -2665
106 Found no solution with cost -1.01Inf .. -2666
107
108 P2 = -2665
109 P = 2665
110 F = [(0, 1, 1, 0, 0, 1, 1, 0, 1)]
111 N = 22
112 Yes (0.00s cpu)
113 */
114
115 % Question 5.6
116
117 pose_contraintes2(Fabriqueur, NbTechniciensTotal, Profit) :-
118   getData(NbTech, NbSor, Tech, Quan, Bene),
119   defineVars(Fabriqueur, NbSor),
120   nbOuvriersNecessaires(Fabriqueur, Tech, NbSor, NbTechniciensTotal),
121   NbTechniciensTotal #=< NbTech,
122   getVarList(Fabriqueur, NbSor, L),
123   profitTotal(Fabriqueur, Bene, Quan, NbSor, Profit),
124   Profit #>=1000,
125   labeling(L).
126
127 /* Tests
128 [eclipse 74]: minimize(pose_contraintes2(F,N,P),N).
129 Found a solution with cost 10
130 Found a solution with cost 9
131 Found a solution with cost 8
132 Found a solution with cost 7
133 Found no solution with cost -1.01Inf .. 6
134
135 F = [(1, 0, 1, 0, 0, 0, 0, 0, 0)]
136 N = 7
137 P = 1040
138 Yes (0.00s cpu)
139 */

```

8

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	2
2	Trouver la meilleure solution	5
3	Code Complet, avec l'ensemble des tests	9

1

1 Trouver une solution au problème

Question 6.1 Nous avons procédé 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"

```
1 %TP6
2
3 :- lib(ic).
4 :- lib(branch_and_bound).
5 :- lib(ic_global).
6
7 getData(Poids, NbPersonnes, NbPlaces) :-
8     NbPersonnes = 10,
9     NbPlaces = 16,
10    Poids = [[(24,39,85,60,165,6,32,123,7,14)].
11
12 defineVars(Places, NbPlaces, NbPersonnes) :-
13     dim(Places, [NbPersonnes]),
14     Borne #= NbPlaces/2,
15     ( for(I,1,NbPersonnes), param(Places, Borne)
16     do
17         Places[I] #:: -Borne..Borne,
18         Places[I] #\= 0
19     ).
20
21 getVarList(Places, NbPersonnes, L) :-
22     ( for(Ind,1,NbPersonnes), fromto([], In, Out, L), param(Places)
23     do
24         Var #= Places[Ind],
25         append([Var], In, Out)
26     ).
27
28 balancoireEquilibree(Poids, NbPersonnes, Places) :-
29     ( for(J,1,NbPersonnes), fromto(0, In, Out, SommeMoments), param(Places,
30     Poids)
31     do
32         Distance #= Places[J],
33         Poids1 #= Poids[J],
34         Out #= In + Distance * Poids1
35     ),
36     SommeMoments #= 0.
37 pasMemesPlaces(Places) :-
38     ic:alldifferent(Places).
39
40 cinqChaqueCote(Places, NbPersonnes) :-
41     NbPersGauche #= NbPersonnes/2,
42     ( for(K,1,NbPersonnes), fromto(0, In, Out, NbPersonnesAGauche), param(
43     Places)
44     do
45         Var #= Places[K],
```

2

```
45     (Var #< 0) => (Out #= In + 1),
46     (Var #> 0) => (Out #= In)
47 ),
48 NbPersGauche #= NbPersonnesAGauche.
49
50 parentsEncadrentEnfants(Places) :-
51     PlaceMere #= Places[4],
52     PlacePere #= Places[8],
53     ((PlaceMere #= ic:max(Places)) and (PlacePere #= ic:min(Places)))
54 or
55     ((PlacePere #= ic:max(Places)) and (PlaceMere #= ic:min(Places))).
56 /*
57 (maxlist(Places, PlaceMere) and minlist(Places, PlacePere))
58 or
59 (maxlist(Places, PlacePere) and minlist(Places, PlaceMere)).
60 */
61
62 danEtMaxDevantParents(Places) :-
63     PlaceDan #= Places[6],
64     PlaceMax #= Places[9],
65     PlaceMere #= Places[4],
66     PlacePere #= Places[8],
67     (((PlaceMere #<0) and (PlaceDan #<0)) and ((PlaceMere #= PlaceDan -
68     1) and (PlacePere #= PlaceMax + 1)))
69 or
70     (((PlaceMere #<0) and (PlaceMax #<0)) and ((PlaceMere #= PlaceMax -
71     1) and (PlacePere #= PlaceDan + 1)))
72 or
73     (((PlacePere #<0) and (PlaceDan #<0)) and ((PlacePere #= PlaceDan -
74     1) and (PlaceMere #= PlaceMax + 1))).
75
76 solve(Places) :-
77     getData(Poids, NbPers, NbPlac),
78     defineVars(Places, NbPlac, NbPers),
79     pasMemesPlaces(Places),
80     balancoireEquilibree(Poids, NbPers, Places),
81     cinqChaqueCote(Places, NbPers),
82     parentsEncadrentEnfants(Places),
83     danEtMaxDevantParents(Places),
84     getVarList(Places, NbPers, L),
85     labeling(L).
86
87 /* Tests
88 [eclipse 29]: solve(P).
89
90 P = [[(-6, -5, -4, -8, 2, 5, 3, 6, -7, 1)
91 Yes (0.02s cpu, solution 1, maybe more) ? ;
92
93 P = [[(-6, -5, -1, 8, 5, 7, 3, -8, -7, 1)
94 Yes (0.02s cpu, solution 2, maybe more) ? ;
```

3

```
94
95 P = [[(-6, -5, 1, -8, -2, 6, 5, 7, -7, 4)
96 Yes (0.02s cpu, solution 3, maybe more) ?
97 */
```

Question 6.2 Voici une solution possible :

```
1 P = [[(-6, -5, -4, -8, 2, 5, 3, 6, -7, 1)
2 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

```
1 % Impose que la mere soit a gauche sur la balançoire
2 elimineSymetrie(Places) :-
3     Places[4] #< 0.
```

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

Listing 2 – "Solution 2"

```
1 solve2(Places) :-
2     getData(Poids, NbPers, NbPlac),
3     defineVars(Places, NbPlac, NbPers),
4     elimineSymetrie(Places), %Gain de temps monstrueux
5     pasMemesPlaces(Places),
6     cinqChaqueCote(Places, NbPers),
7     balancoireEquilibree(Poids, NbPers, Places),
8     parentsEncadrentEnfants(Places),
9     danEtMaxDevantParents(Places),
10    getVarList(Places, NbPers, L),
11    labeling(L).
12
13 /* Tests
14 [eclipse 31]: solve2(P).
15
16 P = [[(-6, -5, -4, -8, 2, 5, 3, 6, -7, 1)
17 Yes (0.02s cpu, solution 1, maybe more) ? ;
18
19 P = [[(-6, -5, 1, -8, -2, 6, 5, 7, -7, 4)
20 Yes (0.04s cpu, solution 2, maybe more) ? ;
21
22 P = [[(-6, -5, 3, -8, -3, -7, 4, 7, 6, 5)
23 Yes (0.01s cpu, solution 3, maybe more) ?
24 */
```

4

2 Trouver la meilleure solution

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

```
Listing 3 – "Solution 3"
1  balançoireEquilibree2 (Poids , NbPersonnes , Places , SNormeMoments) :-
2    ( for (J,1,NbPersonnes) , fromto (0 , In , Out , SommeMoments) , fromto (0 , In2 ,
3      Out2 , SNormeMoments) , param (Places , Poids)
4      do
5          Distance #= Places [ J ] ,
6          Poids1 #= Poids [ J ] ,
7          Out #= In + Distance * Poids1 ,
8          (Places [J] #> 0) => (Out2 #= In2 + Distance * Poids1)
9      ) ,
10     SommeMoments #= 0 .
11
12 solve3 (Places) :-
13     getData (Poids , NbPers , NbPlac) ,
14     defineVars (Places , NbPlac , NbPers) ,
15     elimineSymetrie (Places) , %Gain de temps monstrueux
16     pasMemesPlaces (Places) ,
17     cinqChaqueCote (Places , NbPers) ,
18     balançoireEquilibree2 (Poids , NbPers , Places , SNormeMom) ,
19     parentsEncadrentEnfants (Places) ,
20     danEtMaxDevantParents (Places) ,
21     getVarList (Places , NbPers , L) ,
22     minimize (labeling (L) , SNormeMom) .
23
24 /* Tests
25 [eclipse 25]: solve3(P).
26 Found a solution with cost 874
27 Found a solution with cost 872
28 Found a solution with cost 802
29 Found no solution with cost -1.01nf .. 801
30
31 P = [[-3, 1, -2, -6, -1, 4, 3, 5, -5, 2)
32 Yes (1.66s cpu)
33 */
```

Dans le prédicat balançoireEquilibree2, 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 !

5

Utilisation du prédicat search/6

Dans cette partie, nous avons utiliser le prédicat search qui va nous permettre de contrôler 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 variables testées sont celles qui interviennent dans le plus de contraintes.

```
Listing 4 – "Version 1"
1  solve4 (Places) :-
2      getData (Poids , NbPers , NbPlac) ,
3      defineVars (Places , NbPlac , NbPers) ,
4      elimineSymetrie (Places) ,
5      pasMemesPlaces (Places) ,
6      cinqChaqueCote (Places , NbPers) ,
7      balançoireEquilibree2 (Poids , NbPers , Places , SNormeMom) ,
8      parentsEncadrentEnfants (Places) ,
9      danEtMaxDevantParents (Places) ,
10     getVarList (Places , NbPers , L) ,
11     minimize (search (L,0 , occurrence , indomain_min , complete , []) , SNormeMom) .
12
13 /* Tests
14 [eclipse 9]: solve4(P).
15 Found a solution with cost 953
16 Found a solution with cost 852
17 Found a solution with cost 834
18 Found a solution with cost 802
19 Found no solution with cost -1.01nf .. 801
20
21 P = [[-3, 1, -2, -6, -1, 4, 3, 5, -5, 2)
22 Yes (2.50s cpu)
23 */
```

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

Version 2 Ici on vise à 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"
1  quick_sort ([],[]) .
2  quick_sort ([H|T], Sorted):-
3      pivoting (H,T,L1,L2) , quick_sort (L1 , Sorted1) , quick_sort (L2 , Sorted2) ,
4      append (Sorted1 , [H|Sorted2] , Sorted) .
5
6  pivoting (_ , _ , [],[]) .
7  pivoting (H,[X|T] , [X|L] , G):-abs(X)<abs(H) , pivoting (H,T,L,G) , ! .
8  pivoting (H,[X|T] , L , [X|G]):-abs(X)>abs(H) , pivoting (H,T,L,G) , ! .
9
10 /* Tests
```

6

```
11 [eclipse 50]: quick_sort([1,2,3,4,-1,-2,-3,-4],L).
12
13 L = [-1, 1, -2, 2, -3, 3, -4, 4]
14 Yes (0.00s cpu)
15 */
16
17 choice (X) :-
18     get_domain_as_list (X,L) ,
19     quick_sort (L,L2) ,
20     member (X,L2) .
21
22 solve5 (Places) :-
23     getData (Poids , NbPers , NbPlac) ,
24     defineVars (Places , NbPlac , NbPers) ,
25     elimineSymetrie (Places) ,
26     pasMemesPlaces (Places) ,
27     cinqChaqueCote (Places , NbPers) ,
28     balançoireEquilibree2 (Poids , NbPers , Places , SNormeMom) ,
29     parentsEncadrentEnfants (Places) ,
30     danEtMaxDevantParents (Places) ,
31     getVarList (Places , NbPers , L) ,
32     minimize (search (L,0 , input_order , choice , complete , []) , SNormeMom) .
33
34 /* Tests
35 [eclipse 32]: solve5(P).
36 Found a solution with cost 1216
37 Found a solution with cost 956
38 Found a solution with cost 947
39 Found a solution with cost 917
40 Found a solution with cost 852
41 Found a solution with cost 802
42 Found no solution with cost -1.01nf .. 801
43
44 P = [[-3, 1, -2, -6, -1, 4, 3, 5, -5, 2)
45 Yes (1.80s cpu)
46 */
```

Version 3 Cette version est une combinaison des versions précédentes.

```
Listing 6 – "Version 3"
1  solve6 (Places) :-
2      getData (Poids , NbPers , NbPlac) ,
3      defineVars (Places , NbPlac , NbPers) ,
4      elimineSymetrie (Places) ,
5      pasMemesPlaces (Places) ,
6      cinqChaqueCote (Places , NbPers) ,
7      balançoireEquilibree2 (Poids , NbPers , Places , SNormeMom) ,
8      parentsEncadrentEnfants (Places) ,
9      danEtMaxDevantParents (Places) ,
10     getVarList (Places , NbPers , L) ,
11     minimize (search (L,0 , occurrence , choice , complete , []) , SNormeMom) .
12
```

7

```
13 /* Tests
14 [eclipse 52]: solve6(P).
15 Found a solution with cost 933
16 Found a solution with cost 898
17 Found a solution with cost 872
18 Found a solution with cost 852
19 Found a solution with cost 848
20 Found a solution with cost 834
21 Found a solution with cost 802
22 Found no solution with cost -1.01nf .. 801
23
24 P = [[-3, 1, -2, -6, -1, 4, 3, 5, -5, 2)
25 Yes (2.98s cpu)
26 */
```

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.

8

3 Code Complet, avec l'ensemble des tests

Listing 7 – "TP6"

```

1 %TP6
2
3 :- lib(ic).
4 :- lib(branch_and_bound).
5 :- lib(ic_global).
6
7 getData(Poids,NbPersonnes,NbPlaces):-
8     NbPersonnes = 10,
9     NbPlaces = 16,
10    Poids = [[24,39,85,60,165,6,32,123,7,14].
11
12 defineVars(Places,NbPlaces,NbPersonnes):-
13     dim(Places,[NbPersonnes]),
14     Borne #= NbPlaces/2,
15     ( for(1,1,NbPersonnes),param(Places,Borne)
16     do
17         Places[I] #:: -Borne..Borne,
18         Places[I] #\= 0
19     ).
20
21 getVarList(Places,NbPersonnes,L):-
22     ( for(Ind,1,NbPersonnes),fromto([],In,Out,L),param(Places)
23     do
24         Var #= Places[Ind],
25         append([Var],In,Out)
26     ).
27
28 balancoireEquilibree(Poids,NbPersonnes,Places):-
29     ( for(I,1,NbPersonnes),fromto(0,In,Out,SommeMoments),param(Places,
30     Poids)
31     do
32         Distance #= Places[I],
33         Poids1 #= Poids[I],
34         Out #= In + Distance * Poids1
35     ),
36     SommeMoments #= 0.
37
38 pasMemesPlaces(Places):-
39     ic:alldifferent(Places).
40
41 cinqChaqueCote(Places,NbPersonnes):-
42     NbPersGauche #= NbPersonnes/2,
43     ( for(K,1,NbPersonnes),fromto(0,In,Out,NbPersonnesAGauche),param(
44     Places)
45     do
46         Var #= Places[K],
47         (Var #< 0 => (Out #= In + 1),
48         (Var #> 0) => (Out #= In)
49     ),
50
51 NbPersGauche #= NbPersonnesAGauche.
52
53 parentsEncadrentEnfants(Places):-
54     PlaceMere #= Places[4],
55     PlacePere #= Places[8],
56     ((PlaceMere #= ic:max(Places)) and (PlacePere #= ic:min(Places)))
57 or
58     ((PlacePere #= ic:max(Places)) and (PlaceMere #= ic:min(Places))).
59
60
61
62 danEtMaxDevantParents(Places):-
63     PlaceDan #= Places[6],
64     PlaceMax #= Places[9],
65     PlaceMere #= Places[4],
66     PlacePere #= Places[8],
67     (((PlaceMere #<0) and (PlaceDan #<0)) and ((PlaceMere #= PlaceDan -
68     1) and (PlacePere #= PlaceMax + 1)))
69 or
70     (((PlaceMere #<0) and (PlaceMax #<0)) and ((PlaceMere #= PlaceMax -
71     1) and (PlacePere #= PlaceDan + 1)))
72 or
73     (((PlacePere #<0) and (PlaceDan #<0)) and ((PlacePere #= PlaceDan -
74     1) and (PlaceMere #= PlaceMax + 1)))
75
76 solve(Places):-
77     getData(Poids,NbPers,NbPlac),
78     defineVars(Places,NbPlac,NbPers),
79     pasMemesPlaces(Places),
80     balancoireEquilibree(Poids,NbPers,Places),
81     cinqChaqueCote(Places,NbPers),
82     parentsEncadrentEnfants(Places),
83     danEtMaxDevantParents(Places),
84     getVarList(Places,NbPers,L),
85     labeling(L).
86
87 /* Tests
88 [eclipse 29]: solve(P).
89
90 P = [[-6, -5, -4, -8, 2, 5, 3, 6, -7, 1)
91 Yes (0.13 s cpu, solution 1, maybe more) ? ;
92
93 P = [[-6, -5, -1, 8, 5, 7, 3, -8, -7, 1)
94 Yes (0.15 s cpu, solution 2, maybe more) ? ;
95
96 P = [[-6, -5, 1, -8, -2, 6, 5, 7, -7, 4)
97 Yes (0.02 s cpu, solution 3, maybe more) ?
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115 /* Tests
116 [eclipse 31]: solve2(P).
117
118 P = [[-6, -5, -4, -8, 2, 5, 3, 6, -7, 1)
119 Yes (0.02 s cpu, solution 1, maybe more) ? ;
120
121 P = [[-6, -5, 1, -8, -2, 6, 5, 7, -7, 4)
122 Yes (0.04 s cpu, solution 2, maybe more) ? ;
123
124 P = [[-6, -5, 3, -8, -3, -7, 4, 7, 6, 5)
125 Yes (0.01 s cpu, solution 3, maybe more) ?
126
127
128 balancoireEquilibree2(Poids,NbPersonnes,Places,SNormeMoments):-
129     ( for(I,1,NbPersonnes),fromto(0,In,Out,SommeMoments),fromto(0,In2,
130     Out2,SNormeMoments),param(Places,Poids)
131     do
132         Distance #= Places[I],
133         Poids1 #= Poids[I],
134         Out #= In + Distance * Poids1,
135         (Places[I] #> 0) => (Out2 #= In2 + Distance * Poids1)
136     ),
137     SommeMoments #= 0.
138
139 solve3(Places):-
140     getData(Poids,NbPers,NbPlac),
141     defineVars(Places,NbPlac,NbPers),
142     elimineSymetrie(Places), %Gain de temps monstrueux
143     pasMemesPlaces(Places),
144     cinqChaqueCote(Places,NbPers),
145     balancoireEquilibree2(Poids,NbPers,Places,SNormeMom),
146     parentsEncadrentEnfants(Places),
147     danEtMaxDevantParents(Places),
148     getVarList(Places,NbPers,L),
149     minimize(labeling(L),SNormeMom).
150
151 /* Tests
152 [eclipse 25]: solve3(P).
153 Found a solution with cost 874
154 Found a solution with cost 872
155 Found a solution with cost 802
156 Found no solution with cost -1.01nf .. 801
157
158 P = [[-3, 1, -2, -6, -1, 4, 3, 5, -5, 2)
159 Yes (1.66 s cpu)
160
161
162 % Version 1 de search :
163
164 solve4(Places):-
165     getData(Poids,NbPers,NbPlac),
166     defineVars(Places,NbPlac,NbPers),
167     elimineSymetrie(Places),
168     pasMemesPlaces(Places),
169     cinqChaqueCote(Places,NbPers),
170     balancoireEquilibree2(Poids,NbPers,Places,SNormeMom),
171     parentsEncadrentEnfants(Places),
172     danEtMaxDevantParents(Places),
173     getVarList(Places,NbPers,L),
174     minimize(search(L,0,occurrence,indomain_min,complete,[],SNormeMom).
175
176 /* Tests
177 [eclipse 9]: solve4(P).
178 Found a solution with cost 953
179 Found a solution with cost 852
180 Found a solution with cost 834
181 Found a solution with cost 802
182 Found no solution with cost -1.01nf .. 801
183
184 P = [[-3, 1, -2, -6, -1, 4, 3, 5, -5, 2)
185 Yes (2.50 s cpu)
186
187 % Ici, il semble que most_constrained soit plus adapte que occurrence, vu la
188 vitesse de reponse (2.13 s au lieu de 2.50 s)
189
190 % Version 2 de search :
191
192 /* Ici on vise a essayer de mettre les places les plus au centre en premier,
193 parce que c'est dans ce cas que l'on minimise le mieux le moment total. */
194 quick_sort([],[]).
195 quick_sort([_],_).
196 quick_sort([H:T],Sorted):-
197     pivoting(H,T,L1,L2),quick_sort(L1,Sorted1),quick_sort(L2,Sorted2),
198     append(Sorted1,[H],Sorted2),
199     Sorted.
200
201 pivoting(_H,[],[],[]).
202 pivoting(H,[X:T],[X:L],G):-abs(X)<=abs(H),pivoting(H,T,L,G),!.

```



```

200 pivoting(H,[X|T],L,[X|G]):-abs(X)>abs(H),pivoting(H,T,L,G),!.
201
202 /* Tests
203 [eclipse 50]: quick_sort([1,2,3,4,-1,-2,-3,-4],L).
204
205 L = [-1, 1, -2, 2, -3, 3, -4, 4]
206 Yes (0.00s cpu)
207 */
208
209 choice(X) :-
210     get_domain_as_list(X,L),
211     quick_sort(L,L2),
212     member(X,L2).
213
214 solve5(Places) :-
215     getData(Poids,NbPers,NbPlac),
216     defineVars(Places,NbPlac,NbPers),
217     elimineSymetrie(Places),
218     pasMemesPlaces(Places),
219     cinqChaqueCote(Places,NbPers),
220     balancoireEquilibree2(Poids,NbPers,Places,SNormeMom),
221     parentsEncadrentEnfants(Places),
222     danEtMaxDevantParents(Places),
223     getVarList(Places,NbPers,L),
224     minimize(search(L,0,input_order,choice,complete,[],SNormeMom)).
225
226 /* Tests
227 [eclipse 32]: solve5(P).
228 Found a solution with cost 1216
229 Found a solution with cost 956
230 Found a solution with cost 947
231 Found a solution with cost 917
232 Found a solution with cost 852
233 Found a solution with cost 802
234 Found no solution with cost -1.01nf .. 801
235
236 P = [[-3, 1, -2, -6, -1, 4, 3, 5, -5, 2]
237 Yes (1.80s cpu)
238 */
239
240 % Version 3 de search :
241
242 solve6(Places) :-
243     getData(Poids,NbPers,NbPlac),
244     defineVars(Places,NbPlac,NbPers),
245     elimineSymetrie(Places),
246     pasMemesPlaces(Places),
247     cinqChaqueCote(Places,NbPers),
248     balancoireEquilibree2(Poids,NbPers,Places,SNormeMom),
249     parentsEncadrentEnfants(Places),
250     danEtMaxDevantParents(Places),
251     getVarList(Places,NbPers,L),
252     minimize(search(L,0,occurrence,choice,complete,[],SNormeMom)).

```

13

```

253
254 /* Tests
255 [eclipse 52]: solve6(P).
256 Found a solution with cost 933
257 Found a solution with cost 898
258 Found a solution with cost 872
259 Found a solution with cost 852
260 Found a solution with cost 848
261 Found a solution with cost 834
262 Found a solution with cost 802
263 Found no solution with cost -1.01nf .. 801
264
265 P = [[-3, 1, -2, -6, -1, 4, 3, 5, -5, 2]
266 Yes (2.98s cpu)
267 */
268
269 % Le resultat n'est pas satisfaisant ici
270
271 % Version 4 :
272
273 solve7(Places) :-
274     getData(Poids,NbPers,NbPlac),
275     defineVars(Places,NbPlac,NbPers),
276     elimineSymetrie(Places),
277     pasMemesPlaces(Places),
278     cinqChaqueCote(Places,NbPers),
279     balancoireEquilibree2(Poids,NbPers,Places,SNormeMom),
280     parentsEncadrentEnfants(Places),
281     danEtMaxDevantParents(Places),
282     getVarList2(Places,NbPers,L),
283     minimize(search(L,0,input_order,indomain_split,complete,[],SNormeMom)).

```

14