

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 prédictat permettant d'effectuer un choix parmi des ensembles. Voici le code de nos prédictats, ainsi que les tests associés.

Listing 1 – "Prédicat choixCouleur"

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

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.

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

Question 1.3 Voici le code du prédictat isBetween :

Listing 2 – "Prédicat isBetween"

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

Question 1.4 Voici le code du prédictat 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
9 ,NbMaxR) ,
10 isBetween( NbCondensateur ,
11 NbMinC,NbMaxC) ,
12 NbResistance >=NbCondensateur .
13
14 /* Tests
15 [eclipse 79]: commande( NbResistance ,NbCondensateur ) .
16
17 NbResistance = 9000
18 NbCondensateur = 9000
19 Yes ( 7.73s cpu, solution 1, maybe more ) ? ;
20
21 NbResistance = 9001
22 NbCondensateur = 9000
23 Yes ( 7.74s cpu, solution 2, maybe more ) ? ;
24
25 */

```

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

Question 1.5 Le temps de réponse n'est pas négligeable puisqu'il est de 7.73 secondes.

En utilisant la trace de l'exécution, on constate que l'on est ici dans un contexte de "Generate and Test", qui est dans ce cas beaucoup plus coûteux en temps.

Il génère les solutions qui sont dans les intervalles données (il y en a beaucoup !), puis il teste celles qui remplissent la condition `>=`.

Comme on peut le constater dans le dessin de l'arbre de recherche Prolog 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.

Question 1.6 On réitère l'essai mais en mettant le prédictat `>=` 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

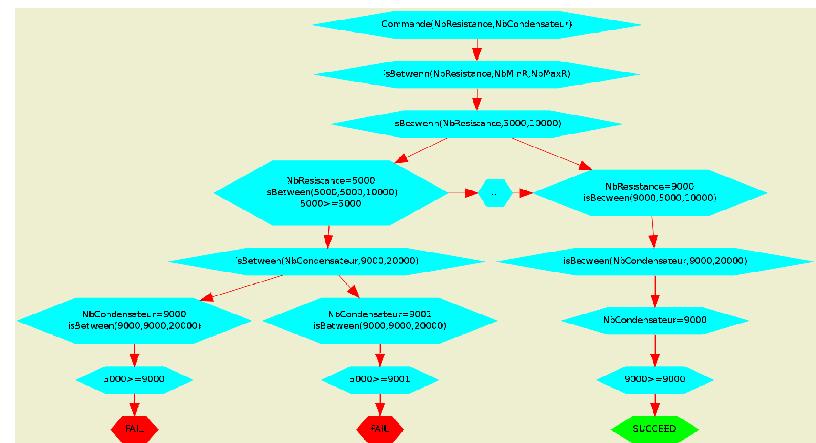


FIGURE 1 – Arbre Prolog 1

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.

1.3 Le solveur ic à la rescousse

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

Listing 4 – "prédictat 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

```

```

15 Yes (0.00s cpu)
16 */

```

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

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

Listing 5 – "prédictat Commande"

```

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

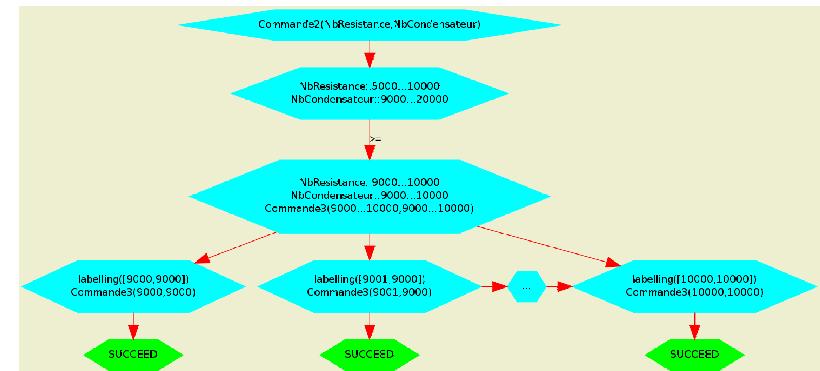


FIGURE 2 – Arbre Prolog 2

2 Zoologie

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

Listing 6 – "Prédicat chapie"

```

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

```

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

Listing 7 – "Prédicat chapie - test"

```

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 puis 1,1,6,2, etc.

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 prédicts ont ici exactement la même attitude pour ce cas de figure.

Listing 9 – "Prédicat vabs - vabs2"

```

1 vabs1(Val ,AbsVal) :- Val#=AbsVal ,
2                               AbsVal#>=0,
3                               labeling([ Val ,AbsVal ]).
4 vabs1(Val ,AbsVal) :- Val*(-1)#=AbsVal ,
5                               AbsVal#>0,
6                               labeling([ Val ,AbsVal ]).
7
8 vabs2(0,0).
9 vabs2(Val ,AbsVal) :- Val#=AbsVal
10
11                               or
12                               Val*(-1)#=AbsVal ,
13                               AbsVal#>0,
14                               labeling([ Val ,AbsVal ]).
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

```

```

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

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

Question 1.13 Définition du prédicat *faitListe/4*:

Listing 10 – "Prédicat faitListe"

```

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

```

```

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

```

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

Question 1.14 Définition du prédictat *suite/1* :

```

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

```

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

Listing 12 – "Réquête"

```

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

```

```

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

```

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.

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

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
15 choixCouleur(CouleurBateau, CouleurVoiture) :- couleurVoiture(CouleurVoiture
   ),
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)
   , nbMaxC(NbMaxC), nbMinC(NbMinC),
46                           isBetween(NbResistance, NbMinR
   , NbMaxR),

```

```

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

```

```

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)
103     .nbMaxC(NbMaxC), nbMinC(NbMinC),
104         NbResistance #:: NbMinR..
105             NbMaxR,
106             NbCondensateur #:: NbMinC..
107                 NbMaxC,
108                 NbResistance #>=
109                     NbCondensateur,
110                         labeling([NbResistance,
111                             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 %Zoologie
129 %Question 1.9
130
131 nbChatsMax(1000).
132 nbPiesMax(1000).
133
134 chapie(Chats, Pies, Pattes, Tetes) :- nbChatsMax(NbChatsMax),
135 nbPiesMax(NbPiesMax),
136 Chats #:: 0 .. NbChatsMax,
137 Pies #:: 0 .. NbPiesMax,
138 Pattes #= Chats*4+Pies*2,

```

140
141
142 /*
143 [eclipse 52]: chapie(2, Pies, Pattes, 5).
144
145 Pies = 3
146 Pattes = 14
147 Yes (0.00s cpu)
148 */
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 vabs1(Val, AbsVal) :- Val*(-1)=#AbsVal,
180     AbsVal#>0,  

181     labeling([Val, AbsVal]).  

182
183 vabs2(0, 0).
184 vabs2(Val, AbsVal) :- Val#=AbsVal
185     or
186         Val*(-1)=#AbsVal,
187             AbsVal#>0,
188                 labeling([Val, AbsVal]).  

189
190 /* Tests
191 [eclipse 5]: vabs1(-2, Y).
192
```

```

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 ]),
223 (for(I,1,Taille), param(VarListe, Min, Max) do
224 (Elem #:: Min..Max,
225 indomain(Elem),
226 VarListe[I] #= Elel
227 )
228 ).
```

229

230 /* Test

231 [eclipse 45]: faitListe(L,2,1,2).

232

233 L = [J(1, 1)

234 Yes (0.00s cpu, solution 1, maybe more) ? ;

235

236 L = J(1, 2)

237 Yes (0.00s cpu, solution 2, maybe more) ? ;

238

239 L = J(2, 1)

240 Yes (0.00s cpu, solution 3, maybe more) ? ;

241

242 L = J(2, 2)

243 Yes (0.00s cpu, solution 4)

244

245 Autre test

```

246 [eclipse 46]: faitListe([J(I,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 ]),
255 (for(I,1,Taille-2), param(ListVar) do
256 (Temp #= ListVar[I+1],
257 vabs1(Temp, VabsDeuxElem),
258 ListVar[I+2] #= VabsDeuxElem
259
260
261 /* Test
262 [eclipse 47]: suite([J(3,-5,2)).
263
264 Yes (0.00s cpu)
265
266 Autre test
267 [eclipse 48]: suite([J(2,2,3)).
268
269 No (0.00s cpu)
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 dim(L,[10]),
277 Elel1#::1..10,Elel2#::1..10,
278 indomain(Elel1),indomain(Elel2),
279 L[1]#=Elel1,L[2]#=Elel2,
280 suite(L).
281 */
282 /*
283 dim(L,[10]),
284 Elel1#::1..10,Elel2#::1..10,
285 indomain(Elel1),indomain(Elel2),
286 L[1]#=Elel1,L[2]#=Elel2,
287 L[1]#\=L[10],
288 suite(L).
289 dim(L,[10]),
290 Elel1#::1..100,Elel2#::1..100,
291 indomain(Elel1),indomain(Elel2),
292 L[1]#=Elel1,L[2]#=Elel2,
293 L[1]#\=L[10],
294 suite(L).
295
296 No (0.94s cpu)
297
```

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

Rapport Travaux Pratiques :

Programmation par Contraintes

- TP 2 :

Contraintes Logiques

Nicolas Desfeux
Aurélien Texier

1^{er} mars 2011

Dans ce T.P., nous allons utiliser les contraintes et les domaines finis pour résoudre un problème. Nous allons tenter, à partir des informations (contraintes) qui nous sont fournies, de déduire d'autres informations.

Pour cela, nous allons définir plusieurs prédictats qui nous permettront au final d'obtenir les informations que nous cherchons.

Question 1.1 Nous allons définir les différents domaines dont nous allons avoir besoin au cours de notre problème.

Listing 1 – "Domaines"

```

1 :- local domain(pays(anglais, espagnol, ukrainien, norvegien, japonais)).
2 :- local domain(couleur(rouge, verte, blanche, jaune, bleue)).
3 :- local domain(boisson(cafe, the, lait, jusdOrange, eau)).
4 :- local domain(voiture(bmw, toyota, ford, honda, datsun)).
5 :- local domain(animal(chien, serpents, renard, cheval, zebre)).
```

Question 1.2 Nous avons également besoin de définir des prédictats qui permettent de contraindre le domaine des variables qui composent une maison.

Listing 2 – "Domaines maisons"

```

1 domaines_maison(m(Pays, Couleur, Boisson, Voiture, Animal, Numero)) :-
2     Pays &:: pays,
3     Couleur &:: couleur,
4     Boisson &:: boisson,
5     Voiture &:: voiture,
6     Animal &:: animal,
7     Numero #:: 1..5.
```

Question 1.3 Ce prédictat permet de définir la liste des maisons, tout en posant les différentes contraintes que l'on a précédemment définies. C'est également grâce à ce prédictat que l'on pose les numéros des maisons.

Listing 3 – "Prédicat rue(?Rue)"

```

1 rue(Rue) :- length(Rue,5),
2     foreach(Elem,Rue), for(1, 1, 5), fromto([],InP,OutP,
3         FinP), fromto([],InC,OutC,FinC), fromto([],InB,
4         OutB,FinB), fromto([],InV,OutV,FinV), fromto([],InA,
5         OutA,FinA)
6     do
7         domaines_maison(Elem),
8         ELEM = m(P,C,B,V,A,I),
9         OutP=[P|InP],
10        OutC=[C|InC],
11        OutB=[B|InB],
12        OutV=[V|InV],
13        OutA=[A|InA]
14    ),
15    ic_symbolic : alldifferent(FinP),
16    ic_symbolic : alldifferent(FinC),
17    ic_symbolic : alldifferent(FinB),
18    ic_symbolic : alldifferent(FinV),
19    ic_symbolic : alldifferent(FinA).
20
21 /* Tests
22 [ eclipse 42]: rue(R).
23
24 R = [m(_296{[anglais, espagnol, ukrainien, norvegien, japonais]}, _400{[rouge,
25     , verte, blanche, jaune, bleue]}, _504{[cafe, the, lait, jusdOrange, eau
26     ]}, _608{[bmw, toyota, ford, honda, datsun]}, _712{[chien, serpents,
27     renard, cheval, zebre]}, 1), m(_894{[anglais, espagnol, ukrainien,
28     norvegien, japonais]}, _998{[rouge, verte, blanche, jaune, bleue]}, _1102
29     {[cafe, the, lait, jusdOrange, eau]}, _1206{[bmw, toyota, ford, honda,
30     datsun]}, _1310{[chien, serpents, renard, cheval, zebre]}, 2), m(_1492{[
31     anglais, espagnol, ukrainien, norvegien, japonais]}, _1596{[rouge, verte,
32     blanche, jaune, bleue]}, _1700{[cafe, the, lait, jusdOrange, eau]},
33     _1804{[bmw, toyota, ford, honda, datsun]}, _1908{[chien, serpents, renard
34     , cheval, zebre]}, 3), m(_2090{[anglais, espagnol, ukrainien, norvegien,
35     japonais]}, _2194{[rouge, verte, blanche, jaune, bleue]}, _2298{[cafe,
36     the, lait, jusdOrange, eau]}, _2402{[bmw, toyota, ford, honda, datsun]},
37     _2506{[chien, serpents, renard, cheval, zebre]}, 4), m(_2688{[anglais,
38     espagnol, ukrainien, norvegien, japonais]}, _2792{[rouge, verte, blanche,
39     jaune, bleue]}, _2896{[cafe, the, lait, jusdOrange, eau]}, _3000{[bmw,
40     toyota, ford, honda, datsun]}, _3104{[chien, serpents, renard, cheval,
41     zebre]}, 5)]
42
43 There are 30 delayed goals. Do you want to see them? (y/n)
44 Yes (0.10s cpu)
45 */
```

Question 1.4 Ce prédictat va permettre un affichage clair des maisons.

Listing 4 – "Prédicat `ecrit_maisons(?Rue)"`

```

1  ecrit_maisons(Rue) :- (foreach(Elem,Rue)
2                                do
3                                   writeln(Elem)
4                                ).  

5  % Permet de tester l'affichage
6  % Rue test = [m(anglais,rouge,cafe,bmw,chien,1),m(norvegien,bleue,lait,honda,
7  % zebre,2)]  

8  /* Test
9  [eclipse 48]: Rue test = [m(anglais,rouge,cafe,bmw,chien,1),m(norvegien,bleue,
10 % lait,honda,zebre,2)],ecrit_maisons(Rue test).
11 m(anglais,rouge,cafe,bmw,chien,1)
12 m(norvegien,bleue,lait,honda,zebre,2)
13 Rue test = [m(anglais,rouge,cafe,bmw,chien,1),m(norvegien,bleue,lait,
14 % honda,zebre,2)]
15 Yes (0.00s cpu)
16 */

```

Question 1.5 On définit ici un prédicat qui permet de récupérer la liste des variables du problème.

On utilise ensuite un prédicat de labeling, qui va permettre de labeliser les variables par rapport aux différents domaines que l'on a défini.

Listing 5 – "Prédicat `getVarList(+List)"`

```

1  getVarList([],[]).
2  getVarList([m(P,C,B,V,A,_I)|Rest],[P,C,B,V,A | Liste]) :- getVarList(Rest,
3                                Liste).
4  labeling_symbolic(Liste) :- (foreach(Elem,Liste)
5                                do
6                                   ic_symbolic:
7                                       indomain(
8                                         Elemt
9                                ).  

9  /* Test
10 rue(R),getVarList(R,L),labeling_symbolic(L).
11
12 R = [m(anglais,rouge,cafe,bmw,chien,1),m(espagnol,verte,the,toyota,
13 % serpents,2),m(ukrainien,blanche,lait,ford,renard,3),m(norvegien,
14 % jaune,jusdOrange,honda,cheval,4),m(japonais,bleue,eau,datsun,
15 % zebre,5)]
16 L = [anglais,rouge,cafe,bmw,chien,espagnol,verte,the,toyota,serpents
17 % ,ukrainien,blanche,lait,ford,renard,norvegien,jaune,jusdOrange,
18 % honda,...]
19 Yes (0.00s cpu, solution 1, maybe more) ? ;
20
21 R = [m(anglais,rouge,cafe,bmw,chien,1),m(espagnol,verte,the,toyota,
22 % serpents,2),m(ukrainien,blanche,lait,ford,renard,3),m(norvegien,
23 % jaune,jusdOrange,honda,cheval,4),m(japonais,bleue,eau,datsun,
24 % zebre,5)]

```

```

17 R = [m(anglais,rouge,cafe,bmw,chien,1),m(espagnol,verte,the,toyota,
18 % serpents,2),m(ukrainien,blanche,lait,ford,renard,3),m(norvegien,
19 % jaune,jusdOrange,honda,cheval,4),m(japonais,bleue,eau,datsun,
20 % zebre,5)]
21 L = [anglais,rouge,cafe,bmw,chien,espagnol,verte,the,toyota,serpents
22 % ,ukrainien,blanche,lait,ford,renard,norvegien,jaune,jusdOrange,
23 % honda,...]
24 Yes (0.00s cpu, solution 2, maybe more) ? ;
25
26 R = [m(anglais,rouge,cafe,bmw,chien,1),m(espagnol,verte,the,toyota,
27 % serpents,2),m(ukrainien,blanche,lait,ford,renard,3),m(norvegien,
28 % jaune,jusdOrange,datsun,cheval,4),m(japonais,bleue,eau,honda,
29 % zebre,5)]
30 L = [anglais,rouge,cafe,bmw,chien,espagnol,verte,the,toyota,serpents
31 % ,ukrainien,blanche,lait,ford,renard,norvegien,jaune,jusdOrange,
32 % datsun,...]
33
34 */

```

Question 1.6 Le prédicat `resoudre` va permettre de trouver une solution respectant les contraintes de domaines, mais pas les contraintes du problèmes.

Listing 6 – "Prédicat `resoudre(?Rue)"`

```

1  resoudre(R) :- rue(R),getVarList(R,L),labeling_symbolic(L),ecrit_maisons(R).
2
3  /* Test
4  [eclipse 3]: resoudre(R).
5  m(anglais,rouge,cafe,bmw,chien,1)
6  m(espagnol,verte,the,toyota,serpents,2)
7  m(ukrainien,blanche,lait,ford,renard,3)
8  m(norvegien,jaune,jusdOrange,honda,cheval,4)
9  m(japonais,bleue,eau,datsun,zebre,5)
10
11 R = [m(anglais,rouge,cafe,bmw,chien,1),m(espagnol,verte,the,toyota,
12 % serpents,2),m(ukrainien,blanche,lait,ford,renard,3),m(norvegien,
13 % jaune,jusdOrange,honda,cheval,4),m(japonais,bleue,eau,datsun,
14 % zebre,5)]
15 Yes (0.10s cpu, solution 1, maybe more) ? ;
16 m(anglais,rouge,cafe,bmw,chien,1)
17 m(espagnol,verte,the,toyota,serpents,2)
18 m(ukrainien,blanche,lait,ford,renard,3)
19 m(norvegien,jaune,jusdOrange,honda,zebre,4)
20 m(japonais,bleue,eau,datsun,cheval,5)
21
22 R = [m(anglais,rouge,cafe,bmw,chien,1),m(espagnol,verte,the,toyota,
23 % serpents,2),m(ukrainien,blanche,lait,ford,renard,3),m(norvegien,
24 % jaune,jusdOrange,honda,zebre,4),m(japonais,bleue,eau,datsun,
25 % cheval,5)]
26 Yes (0.10s cpu, solution 2, maybe more) ? ;
27 m(anglais,rouge,cafe,bmw,chien,1)
28 m(espagnol,verte,the,toyota,serpents,2)
29 m(ukrainien,blanche,lait,ford,renard,3)

```

```

24 m(norvegien, jaune, jasdOrange, datsun, cheval, 4)
25 m(japonais, bleue, eau, honda, zebre, 5)
26
27 R = [m(anglais, rouge, cafe, bmw, chien, 1), m(espagnol, verte, the, toyota,
    serpents, 2), m(ukrainien, blanche, lait, ford, renard, 3), m(norvegien,
    jaune, jasdOrange, datsun, cheval, 4), m(japonais, bleue, eau, honda,
    zebre, 5)]
28 Yes (0.10s cpu, solution 3, maybe more) ? ;
29 m(anglais, rouge, cafe, bmw, chien, 1)
30 m(espagnol, verte, the, toyota, serpents, 2)
31 m(ukrainien, blanche, lait, ford, renard, 3)
32 m(norvegien, jaune, jasdOrange, datsun, zebre, 4)
33 m(japonais, bleue, eau, honda, cheval, 5)
34
35 R = [m(anglais, rouge, cafe, bmw, chien, 1), m(espagnol, verte, the, toyota,
    serpents, 2), m(ukrainien, blanche, lait, ford, renard, 3), m(norvegien,
    jaune, jasdOrange, datsun, zebre, 4), m(japonais, bleue, eau, honda,
    cheval, 5)]
36 */

```

```

28
29 [ eclipse 34]: resoudre2(R).
30 m(norvegien, jaune, eau, toyota, renard, 1)
31 m(ukrainien, bleue, the, ford, cheval, 2)
32 m(anglais, rouge, lait, bmw, serpents, 3)
33 m(espagnol, blanche, jasdOrange, honda, chien, 4)
34 m(japonais, verte, cafe, datsun, zebre, 5)
35
36 R = [m(norvegien, jaune, eau, toyota, renard, 1), m(ukrainien, bleue, the,
    ford, cheval, 2), m(anglais, rouge, lait, bmw, serpents, 3), m(espagnol,
    blanche, jasdOrange, honda, chien, 4), m(japonais, verte, cafe, datsun,
    zebre, 5)]
37 Yes (0.01s cpu, solution 1, maybe more) ? ;
38
39 No (0.01s cpu)
40
41 */

```

Question 1.7 Maintenant, nous définissons les différentes contraintes, et un prédictat pouvant les utiliser.

Listing 7 – "Définition des contraintes et prédictat de résolution"

```

1 contraintes(R) :-      (foreach(m(P,C,B,V,A,I),R),
2                                do
3                                (P &= anglais) #= (C &= rouge),
4                                (P &= espagnol) #= (A &= chien),
5                                (B &= cafe) #= (C &= verte),
6                                (P &= ukrainien) #= (B &= the),
7                                (A &= serpents) #= (V &= bmw),
8                                (C &= jaune) #= (V &= toyota),
9                                (B &= lait) #= (I #= 3),
10                               (P &= norvegien) #= (I #= 1),
11                               (B &= jasdOrange) #= (V &= honda),
12                               (P &= japonais) #= (V &= datsun)
13                                ).
14
15 contraintes2(R) :-      (foreach(m(P1,C1,_B1,V1,_A1,I1),R),param(R)
16                                do
17                                (foreach(m(_P2,C2,_B2,_V2,A2,I2),R),param(I1,C1,V1,P1)
18                                do
19                                ((C1 &= verte) and (C2 &= blanche)) => (I1 #= I2+1),
20                                ((V1 &= ford) and (A2 &= renard)) => ((I2 #= I1+1) or (I2 #= I1-1)),
21                                ((V1 &= toyota) and (A2 &= cheval)) => ((I2 #= I1+1) or (I2 #= I1-1)),
22                                ((P1 &= norvegien) and (C2 &= bleue)) => ((I2 #= I1+1) or (I2 #= I1-1))
23                                )
24                                .
25
26 resoudre2(R) :- rue(R),getVarList(R,L),contraintes(R),contraintes2(R),
27     labeling_symbolic(L),ecrit_maisons(R).
27 /* Test

```

Question 1.8 On obtient donc la réponse aux questions posées par le problème :

Le japonais possède un zèbre et le norvégien boit de l'eau

1 Code Complet, avec l'ensemble des tests

Listing 8 – "TP2"

```

1 :-lib(ic).
2 :-lib(ic_symbolic).
3
4 % Question 2.1
5
6 :- local domain(pays(anglais,espagnol,ukrainien,norvegien,japonais)).
7 :- local domain(couleur(rouge,verte,blanche,jaune,bleue)).
8 :- local domain(boisson(cafe,the,lait,jusdOrange,eau)).
9 :- local domain(voiture(bmw,toyota,ford,honda,datsun)).
10 :- local domain(animal(chien,serpents,renard,cheval,zebre)).
11
12 domaines_maison(m(Pays,Couleur,Boisson,Voiture,Animal,Numero)) :-  

13     Pays &:: pays,  

14     Couleur &:: couleur,  

15     Boisson &:: boisson,  

16     Voiture &:: voiture,  

17     Animal &:: animal,  

18     Numero #:: 1..5.
19
20 rue(Rue) :- length(Rue,5),  

21     (foreach(Elem,Rue), for(I, 1, 5), fromto([],InP,OutP,  

22         FinP), fromto([],InC,OutC,FinC), fromto([],InB,  

23         OutB,FinB), fromto([],InV,OutV,FinV), fromto([],  

24         InA,OutA,FinA)  

25         do  

26             domaines_maison(Elem),  

27             Elemt = m(P,C,B,V,A,I),  

28             OutP=[P|InP],  

29             OutC=[C|InC],  

30             OutB=[B|InB],  

31             OutV=[V|InV],  

32             OutA=[A|InA]  

33             ),  

34             ic_symbolic: alldifferent(FinP),  

35             ic_symbolic: alldifferent(FinC),  

36             ic_symbolic: alldifferent(FinB),  

37             ic_symbolic: alldifferent(FinV),  

38             ic_symbolic: alldifferent(FinA).  

39
40 /*Tests
41 [eclipse 42]: rue(R).
42
43 R = [m(_296{[anglais , espagnol , ukrainien , norvegien , japonais ]}, _400{[rouge , verte , blanche , jaune , bleue ]}, _504{[cafe , the , lait , jusdOrange , eau ]}, _608{[bmw , toyota , ford , honda , datsun ]}, _712{[chien , serpents , renard , cheval , zebre ]}, 1), m(_894{[anglais , espagnol , ukrainien , norvegien , japonais ]}, _998{[rouge , verte , blanche , jaune , bleue ]}, _1102{[cafe , the , lait , jusdOrange , eau ]}, _1206{[bmw , toyota , ford , honda , datsun ]}, _1310{[chien , serpents , renard , cheval , zebre ]}, 2), m(_1492{[anglais , espagnol , ukrainien , norvegien , japonais ]}, _1596{[rouge , verte , blanche , jaune , bleue ]}, _1700{[cafe , the , lait , jusdOrange , eau ]}, _1804{[bmw , toyota , ford , honda , datsun ]}, _1908{[chien , serpents , renard , cheval , zebre ]}, 3), m(_2090{[anglais , espagnol , ukrainien , norvegien , japonais ]}, _2194{[rouge , verte , blanche , jaune , bleue ]}, _2298{[cafe , the , lait , jusdOrange , eau ]}, _2402{[bmw , toyota , ford , honda , datsun ]}, _2506{[chien , serpents , renard , cheval , zebre ]}, 4), m(_2688{[anglais , espagnol , ukrainien , norvegien , japonais ]}, _2792{[rouge , verte , blanche , jaune , bleue ]}, _2896{[cafe , the , lait , jusdOrange , eau ]}, _3000{[bmw , toyota , ford , honda , datsun ]}, _3104{[chien , serpents , renard , cheval , zebre ]}, 5)]
```

```

41
42 There are 30 delayed goals. Do you want to see them? (y/n)
43 Yes (0.10s cpu)
44 */
45
46 ecrit_maisons(Rue) :- (foreach(Elem,Rue)
47                                     do
48                                         writeln(Elem)
49                                     ).
50 % Permet de tester l'affichage
51 % Ruetest = [m(anglais ,rouge ,cafe ,bmw ,chien ,1) ,m(norvegien ,bleue ,lait ,honda ,
52 % zebre ,2)]
53 /* Test
54 [eclipse 48]: Ruetest = [m(anglais ,rouge ,cafe ,bmw ,chien ,1) ,m(norvegien ,bleue ,
55 % lait ,honda ,zebre ,2)],ecrit_maisons(Ruetest).
56 m(anglais ,rouge ,cafe ,bmw ,chien ,1)
57 m(norvegien ,bleue ,lait ,honda ,zebre ,2)
58 Ruetest = [m(anglais ,rouge ,cafe ,bmw ,chien ,1) ,m(norvegien ,bleue ,lait ,
59 % honda ,zebre ,2)]
60 Yes (0.00s cpu)
61 */
62
63 getVarList([],[]).
64 getVarList([m(P,C,B,V,A,_I)|Rest],[P,C,B,V,A | Liste]) :- getVarList(Rest ,
65 Liste).
66 labeling_symbolic(Liste) :- (foreach(Elem,Liste)
67                                     do
68                                         ic_symbolic:
69                                         indomain(
70                                         Elemt
71                                         ).
72
73 rue(R),getVarList(R,L),labeling_symbolic(L).
74
75 R = [m(anglais ,rouge ,cafe ,bmw ,chien ,1) ,m(espagnol ,verte ,toyota ,
76 % serpents ,2) ,m(ukrainien ,blanche ,lait ,ford ,renard ,3) ,m(norvegien ,
```

```

jaune , jasdOrange , honda , cheval , 4) , m(japonais , bleue , eau , datsun ,
zebre , 5)]
76 L = [anglais , rouge , cafe , bmw , chien , espagnol , verte , the , toyota , serpents
, ukrainien , blanche , lait , ford , renard , norvegien , jaune , jasdOrange ,
honda , ...]
77 Yes (0.00s cpu, solution 1, maybe more) ? ;
78
79 R = [m(anglais , rouge , cafe , bmw , chien , 1) , m(espagnol , verte , the , toyota ,
serpents , 2) , m(ukrainien , blanche , lait , ford , renard , 3) , m(norvegien ,
jaune , jasdOrange , honda , zebre , 4) , m(japonais , bleue , eau , datsun ,
cheval , 5)]
80 L = [anglais , rouge , cafe , bmw , chien , espagnol , verte , the , toyota , serpents
, ukrainien , blanche , lait , ford , renard , norvegien , jaune , jasdOrange ,
honda , ...]
81 Yes (0.00s cpu, solution 2, maybe more) ? ;
82
83 R = [m(anglais , rouge , cafe , bmw , chien , 1) , m(espagnol , verte , the , toyota ,
serpents , 2) , m(ukrainien , blanche , lait , ford , renard , 3) , m(norvegien ,
jaune , jasdOrange , datsun , 4) , m(japonais , bleue , eau , honda ,
zebre , 5)]
84 L = [anglais , rouge , cafe , bmw , chien , espagnol , verte , the , toyota , serpents
, ukrainien , blanche , lait , ford , renard , norvegien , jaune , jasdOrange ,
datsun , ...]
85 */
86 */
87
88 contraintes(R) :-      (foreach(m(P,C,B,V,A,I),R)
89           do
90             (P &= anglais) #= (C &= rouge),
91             (P &= espagnol) #= (A &= chien),
92             (B &= cafe) #= (C &= verte),
93             (P &= ukrainien) #= (B &= the),
94             (A &= serpents) #= (V &= bmw),
95             (C &= jaune) #= (V &= toyota),
96             (B &= lait) #= (I #= 3),
97             (P &= norvegien) #= (I #= 1),
98             (B &= jasdOrange) #= (V &= honda),
99             (P &= japonais) #= (V &= datsun)
100            ).
101
102 contraintes2(R) :-      (foreach(m(P1,C1,_B1,V1,_A1,I1),R),param(R)
103           do
104             (foreach(m(_P2,C2,_B2
105               ,_V2,A2,I2),R),
106               param(I1,C1,V1,P1
107               )
108               do
109                 ((C1 &= verte) and (C2 &= blanche)) => (I1 #= I2+1),
110                 ((V1 &= ford) and (A2 &= renard)) => ((I2 #= I1+1) or (I2 #= I1
111                   -1)),
112                 ((V1 &= toyota) and (A2 &= cheval)) => ((I2 #= I1+1) or (I2 #=
113                   I1-1)),
114
115 resoudre(R) :- rue(R),contraintes(R),getVarList(R,L),labeling_symbolic(L),
116   ecrit_maisons(R).
117
118 resoudre2(R) :- rue(R),getVarList(R,L),contraintes2(R),
119   labeling_symbolic(L),ecrit_maisons(R).
```

Rapport Travaux Pratiques :

Programmation par Contraintes

- TP 2 :

Contraintes Logiques

Nicolas Desfeux
Aurélien Texier

1^{er} mars 2011

Dans ce T.P., nous allons utiliser les contraintes et les domaines finis pour résoudre un problème. Nous allons tenter, à partir des informations (contraintes) qui nous sont fournies, de déduire d'autres informations.

Pour cela, nous allons définir plusieurs prédictats qui nous permettront au final d'obtenir les informations que nous cherchons.

Question 1.1 Nous allons définir les différents domaines dont nous allons avoir besoin au cours de notre problème.

Listing 1 – "Domaines"

```

1 :- local domain(pays(anglais, espagnol, ukrainien, norvegien, japonais)).
2 :- local domain(couleur(rouge, verte, blanche, jaune, bleue)).
3 :- local domain(boisson(cafe, the, lait, jusdOrange, eau)).
4 :- local domain(voiture(bmw, toyota, ford, honda, datsun)).
5 :- local domain(animal(chien, serpents, renard, cheval, zebre)).
```

Question 1.2 Nous avons également besoin de définir des prédictats qui permettent de contraindre le domaine des variables qui composent une maison.

Listing 2 – "Domaines maisons"

```

1 domaines_maison(m(Pays, Couleur, Boisson, Voiture, Animal, Numero)) :-
2     Pays &:: pays,
3     Couleur &:: couleur,
4     Boisson &:: boisson,
5     Voiture &:: voiture,
6     Animal &:: animal,
7     Numero #:: 1..5.
```

Question 1.3 Ce prédictat permet de définir la liste des maisons, tout en posant les différentes contraintes que l'on a précédemment définies. C'est également grâce à ce prédictat que l'on pose les numéros des maisons.

Listing 3 – "Prédicat rue(?Rue)"

```

1 rue(Rue) :- length(Rue,5),
2     foreach(Elem,Rue), for(1, 1, 5), fromto([],InP,OutP,
3         FinP), fromto([],InC,OutC,FinC), fromto([],InB,
4         OutB,FinB), fromto([],InV,OutV,FinV), fromto([],InA,
5         OutA,FinA)
6     do
7         domaines_maison(Elem),
8         ELEM = m(P,C,B,V,A,I),
9         OutP=[P|InP],
10        OutC=[C|InC],
11        OutB=[B|InB],
12        OutV=[V|InV],
13        OutA=[A|InA]
14    ),
15    ic_symbolic : alldifferent(FinP),
16    ic_symbolic : alldifferent(FinC),
17    ic_symbolic : alldifferent(FinB),
18    ic_symbolic : alldifferent(FinV),
19    ic_symbolic : alldifferent(FinA).
20
21 /* Tests
22 [ eclipse 42]: rue(R).
23
24 R = [m(_296{[anglais, espagnol, ukrainien, norvegien, japonais]}, _400{[rouge,
25     , verte, blanche, jaune, bleue]}, _504{[cafe, the, lait, jusdOrange, eau
26     ]}, _608{[bmw, toyota, ford, honda, datsun]}, _712{[chien, serpents,
27     renard, cheval, zebre]}, 1), m(_894{[anglais, espagnol, ukrainien,
28     norvegien, japonais]}, _998{[rouge, verte, blanche, jaune, bleue]}, _1102
29     {[cafe, the, lait, jusdOrange, eau]}, _1206{[bmw, toyota, ford, honda,
30     datsun]}, _1310{[chien, serpents, renard, cheval, zebre]}, 2), m(_1492{[
31     anglais, espagnol, ukrainien, norvegien, japonais]}, _1596{[rouge, verte,
32     blanche, jaune, bleue]}, _1700{[cafe, the, lait, jusdOrange, eau]},
33     _1804{[bmw, toyota, ford, honda, datsun]}, _1908{[chien, serpents, renard
34     , cheval, zebre]}, 3), m(_2090{[anglais, espagnol, ukrainien, norvegien,
35     japonais]}, _2194{[rouge, verte, blanche, jaune, bleue]}, _2298{[cafe,
36     the, lait, jusdOrange, eau]}, _2402{[bmw, toyota, ford, honda, datsun]},
37     _2506{[chien, serpents, renard, cheval, zebre]}, 4), m(_2688{[anglais,
38     espagnol, ukrainien, norvegien, japonais]}, _2792{[rouge, verte, blanche,
39     jaune, bleue]}, _2896{[cafe, the, lait, jusdOrange, eau]}, _3000{[bmw,
40     toyota, ford, honda, datsun]}, _3104{[chien, serpents, renard, cheval,
41     zebre]}, 5)]
42
43 There are 30 delayed goals. Do you want to see them? (y/n)
44 Yes (0.10s cpu)
45 */
```

Question 1.4 Ce prédictat va permettre un affichage clair des maisons.

Listing 4 – "Prédicat `ecrit_maisons(?Rue)"`

```

1  ecrit_maisons(Rue) :- (foreach(Elem,Rue)
2                                do
3                                   writeln(Elem)
4                                ).  

5  % Permet de tester l'affichage
6  % Rue test = [m(anglais,rouge,cafe,bmw,chien,1),m(norvegien,bleue,lait,honda,
7  % zebre,2)]  

8  /* Test
9  [eclipse 48]: Rue test = [m(anglais,rouge,cafe,bmw,chien,1),m(norvegien,bleue,
10 % lait,honda,zebre,2)],ecrit_maisons(Rue test).
11 m(anglais,rouge,cafe,bmw,chien,1)
12 m(norvegien,bleue,lait,honda,zebre,2)
13 Rue test = [m(anglais,rouge,cafe,bmw,chien,1),m(norvegien,bleue,lait,
14 % honda,zebre,2)]
15 Yes (0.00s cpu)
16 */

```

Question 1.5 On définit ici un prédicat qui permet de récupérer la liste des variables du problème.

On utilise ensuite un prédicat de labeling, qui va permettre de labeliser les variables par rapport aux différents domaines que l'on a défini.

Listing 5 – "Prédicat `getVarList(+List)"`

```

1  getVarList([],[]).
2  getVarList([m(P,C,B,V,A,_I)|Rest],[P,C,B,V,A | Liste]) :- getVarList(Rest,
3                                Liste).
4  labeling_symbolic(Liste) :- (foreach(Elem,Liste)
5                                do
6                                   ic_symbolic:
7                                       indomain(
8                                         Elemt
9                                ).  

9  /* Test
10 rue(R),getVarList(R,L),labeling_symbolic(L).
11
12 R = [m(anglais,rouge,cafe,bmw,chien,1),m(espagnol,verte,the,toyota,
13 % serpents,2),m(ukrainien,blanche,lait,ford,renard,3),m(norvegien,
14 % jaune,jusdOrange,honda,cheval,4),m(japonais,bleue,eau,datsun,
15 % zebre,5)]
16 L = [anglais,rouge,cafe,bmw,chien,espagnol,verte,the,toyota,serpents
17 % ,ukrainien,blanche,lait,ford,renard,norvegien,jaune,jusdOrange,
18 % honda,...]
19 Yes (0.00s cpu, solution 1, maybe more) ? ;
20
21 R = [m(anglais,rouge,cafe,bmw,chien,1),m(espagnol,verte,the,toyota,
22 % serpents,2),m(ukrainien,blanche,lait,ford,renard,3),m(norvegien,
23 % jaune,jusdOrange,honda,cheval,4),m(japonais,bleue,eau,datsun,
24 % zebre,5)]

```

```

17 R = [m(anglais,rouge,cafe,bmw,chien,1),m(espagnol,verte,the,toyota,
18 % serpents,2),m(ukrainien,blanche,lait,ford,renard,3),m(norvegien,
19 % jaune,jusdOrange,honda,cheval,4),m(japonais,bleue,eau,datsun,
20 % zebre,5)]
21 L = [anglais,rouge,cafe,bmw,chien,espagnol,verte,the,toyota,serpents
22 % ,ukrainien,blanche,lait,ford,renard,norvegien,jaune,jusdOrange,
23 % honda,...]
24 Yes (0.00s cpu, solution 2, maybe more) ? ;
25
26 R = [m(anglais,rouge,cafe,bmw,chien,1),m(espagnol,verte,the,toyota,
27 % serpents,2),m(ukrainien,blanche,lait,ford,renard,3),m(norvegien,
28 % jaune,jusdOrange,datsun,cheval,4),m(japonais,bleue,eau,honda,
29 % zebre,5)]
30 L = [anglais,rouge,cafe,bmw,chien,espagnol,verte,the,toyota,serpents
31 % ,ukrainien,blanche,lait,ford,renard,norvegien,jaune,jusdOrange,
32 % datsun,...]
33
34 */

```

Question 1.6 Le prédicat `resoudre` va permettre de trouver une solution respectant les contraintes de domaines, mais pas les contraintes du problèmes.

Listing 6 – "Prédicat `resoudre(?Rue)"`

```

1  resoudre(R) :- rue(R),getVarList(R,L),labeling_symbolic(L),ecrit_maisons(R).
2
3  /* Test
4  [eclipse 3]: resoudre(R).
5  m(anglais,rouge,cafe,bmw,chien,1)
6  m(espagnol,verte,the,toyota,serpents,2)
7  m(ukrainien,blanche,lait,ford,renard,3)
8  m(norvegien,jaune,jusdOrange,honda,cheval,4)
9  m(japonais,bleue,eau,datsun,zebre,5)
10
11 R = [m(anglais,rouge,cafe,bmw,chien,1),m(espagnol,verte,the,toyota,
12 % serpents,2),m(ukrainien,blanche,lait,ford,renard,3),m(norvegien,
13 % jaune,jusdOrange,honda,cheval,4),m(japonais,bleue,eau,datsun,
14 % zebre,5)]
15 Yes (0.10s cpu, solution 1, maybe more) ? ;
16 m(anglais,rouge,cafe,bmw,chien,1)
17 m(espagnol,verte,the,toyota,serpents,2)
18 m(ukrainien,blanche,lait,ford,renard,3)
19 m(norvegien,jaune,jusdOrange,honda,zebre,4)
20 m(japonais,bleue,eau,datsun,cheval,5)
21
22 R = [m(anglais,rouge,cafe,bmw,chien,1),m(espagnol,verte,the,toyota,
23 % serpents,2),m(ukrainien,blanche,lait,ford,renard,3),m(norvegien,
24 % jaune,jusdOrange,honda,zebre,4),m(japonais,bleue,eau,datsun,
25 % cheval,5)]
26 Yes (0.10s cpu, solution 2, maybe more) ? ;
27 m(anglais,rouge,cafe,bmw,chien,1)
28 m(espagnol,verte,the,toyota,serpents,2)
29 m(ukrainien,blanche,lait,ford,renard,3)

```

```

24 m(norvegien, jaune, jasdOrange, datsun, cheval, 4)
25 m(japonais, bleue, eau, honda, zebre, 5)
26
27 R = [m(anglais, rouge, cafe, bmw, chien, 1), m(espagnol, verte, the, toyota,
    serpents, 2), m(ukrainien, blanche, lait, ford, renard, 3), m(norvegien,
    jaune, jasdOrange, datsun, cheval, 4), m(japonais, bleue, eau, honda,
    zebre, 5)]
28 Yes (0.10s cpu, solution 3, maybe more) ? ;
29 m(anglais, rouge, cafe, bmw, chien, 1)
30 m(espagnol, verte, the, toyota, serpents, 2)
31 m(ukrainien, blanche, lait, ford, renard, 3)
32 m(norvegien, jaune, jasdOrange, datsun, zebre, 4)
33 m(japonais, bleue, eau, honda, cheval, 5)
34
35 R = [m(anglais, rouge, cafe, bmw, chien, 1), m(espagnol, verte, the, toyota,
    serpents, 2), m(ukrainien, blanche, lait, ford, renard, 3), m(norvegien,
    jaune, jasdOrange, datsun, zebre, 4), m(japonais, bleue, eau, honda,
    cheval, 5)]
36 */

```

```

28
29 [ eclipse 34]: resoudre2(R).
30 m(norvegien, jaune, eau, toyota, renard, 1)
31 m(ukrainien, bleue, the, ford, cheval, 2)
32 m(anglais, rouge, lait, bmw, serpents, 3)
33 m(espagnol, blanche, jasdOrange, honda, chien, 4)
34 m(japonais, verte, cafe, datsun, zebre, 5)
35
36 R = [m(norvegien, jaune, eau, toyota, renard, 1), m(ukrainien, bleue, the,
    ford, cheval, 2), m(anglais, rouge, lait, bmw, serpents, 3), m(espagnol,
    blanche, jasdOrange, honda, chien, 4), m(japonais, verte, cafe, datsun,
    zebre, 5)]
37 Yes (0.01s cpu, solution 1, maybe more) ? ;
38
39 No (0.01s cpu)
40
41 */

```

Question 1.7 Maintenant, nous définissons les différentes contraintes, et un prédictat pouvant les utiliser.

Listing 7 – "Définition des contraintes et prédictat de résolution"

```

1 contraintes(R) :-      (foreach(m(P,C,B,V,A,I),R),
2                                do
3                                (P &= anglais) #= (C &= rouge),
4                                (P &= espagnol) #= (A &= chien),
5                                (B &= cafe) #= (C &= verte),
6                                (P &= ukrainien) #= (B &= the),
7                                (A &= serpents) #= (V &= bmw),
8                                (C &= jaune) #= (V &= toyota),
9                                (B &= lait) #= (I #= 3),
10                               (P &= norvegien) #= (I #= 1),
11                               (B &= jasdOrange) #= (V &= honda),
12                               (P &= japonais) #= (V &= datsun)
13                                ).
14
15 contraintes2(R) :-      (foreach(m(P1,C1,_B1,V1,_A1,I1),R),param(R)
16                                do
17                                (foreach(m(_P2,C2,_B2,_V2,A2,I2),R),param(I1,C1,V1,P1)
18                                do
19                                ((C1 &= verte) and (C2 &= blanche)) => (I1 #= I2+1),
20                                ((V1 &= ford) and (A2 &= renard)) => ((I2 #= I1+1) or (I2 #= I1-1)),
21                                ((V1 &= toyota) and (A2 &= cheval)) => ((I2 #= I1+1) or (I2 #= I1-1)),
22                                ((P1 &= norvegien) and (C2 &= bleue)) => ((I2 #= I1+1) or (I2 #= I1-1))
23                                )
24                                .
25
26 resoudre2(R) :- rue(R),getVarList(R,L),contraintes(R),contraintes2(R),
27                               labeling_symbolic(L),ecrit_maisons(R).
27 /* Test

```

Question 1.8 On obtient donc la réponse aux questions posées par le problème :

Le japonais possède un zèbre et le norvégien boit de l'eau

1 Code Complet, avec l'ensemble des tests

Listing 8 – "TP2"

```

1 :-lib(ic).
2 :-lib(ic_symbolic).
3
4 % Question 2.1
5
6 :- local domain(pays(anglais,espagnol,ukrainien,norvegien,japonais)).
7 :- local domain(couleur(rouge,verte,blanche,jaune,bleue)).
8 :- local domain(boisson(cafe,the,lait,jusdOrange,eau)).
9 :- local domain(voiture(bmw,toyota,ford,honda,datsun)).
10 :- local domain(animal(chien,serpents,renard,cheval,zebre)).
11
12 domaines_maison(m(Pays,Couleur,Boisson,Voiture,Animal,Numero)) :-  

13     Pays &:: pays,  

14     Couleur &:: couleur,  

15     Boisson &:: boisson,  

16     Voiture &:: voiture,  

17     Animal &:: animal,  

18     Numero #:: 1..5.
19
20 rue(Rue) :- length(Rue,5),  

21     (foreach(Elem,Rue), for(I, 1, 5), fromto([],InP,OutP,  

22         FinP), fromto([],InC,OutC,FinC), fromto([],InB,  

23         OutB,FinB), fromto([],InV,OutV,FinV), fromto([],  

24         InA,OutA,FinA)  

25         do  

26             domaines_maison(Elem),  

27             Elemt = m(P,C,B,V,A,I),  

28             OutP=[P|InP],  

29             OutC=[C|InC],  

30             OutB=[B|InB],  

31             OutV=[V|InV],  

32             OutA=[A|InA]  

33             ),  

34             ic_symbolic: alldifferent(FinP),  

35             ic_symbolic: alldifferent(FinC),  

36             ic_symbolic: alldifferent(FinB),  

37             ic_symbolic: alldifferent(FinV),  

38             ic_symbolic: alldifferent(FinA).  

39
40 /*Tests
41 [eclipse 42]: rue(R).
42
43 R = [m(_296{[anglais , espagnol , ukrainien , norvegien , japonais ]}, _400{[rouge , verte , blanche , jaune , bleue ]}, _504{[cafe , the , lait , jusdOrange , eau ]}, _608{[bmw , toyota , ford , honda , datsun ]}, _712{[chien , serpents , renard , cheval , zebre ]}, 1), m(_894{[anglais , espagnol , ukrainien , norvegien , japonais ]}, _998{[rouge , verte , blanche , jaune , bleue ]}, _1102{[cafe , the , lait , jusdOrange , eau ]}, _1206{[bmw , toyota , ford , honda , datsun ]}, _1310{[chien , serpents , renard , cheval , zebre ]}, 2), m(_1492{[anglais , espagnol , ukrainien , norvegien , japonais ]}, _1596{[rouge , verte , blanche , jaune , bleue ]}, _1700{[cafe , the , lait , jusdOrange , eau ]}, _1804{[bmw , toyota , ford , honda , datsun ]}, _1908{[chien , serpents , renard , cheval , zebre ]}, 3), m(_2090{[anglais , espagnol , ukrainien , norvegien , japonais ]}, _2194{[rouge , verte , blanche , jaune , bleue ]}, _2298{[cafe , the , lait , jusdOrange , eau ]}, _2402{[bmw , toyota , ford , honda , datsun ]}, _2506{[chien , serpents , renard , cheval , zebre ]}, 4), m(_2688{[anglais , espagnol , ukrainien , norvegien , japonais ]}, _2792{[rouge , verte , blanche , jaune , bleue ]}, _2896{[cafe , the , lait , jusdOrange , eau ]}, _3000{[bmw , toyota , ford , honda , datsun ]}, _3104{[chien , serpents , renard , cheval , zebre ]}, 5)]
```

```

41
42 There are 30 delayed goals. Do you want to see them? (y/n)
43 Yes (0.10s cpu)
44 */
45
46 ecrit_maisons(Rue) :- (foreach(Elem,Rue)
47                                     do
48                                         writeln(Elem)
49                                     ).
50 % Permet de tester l'affichage
51 % Ruetest = [m(anglais ,rouge ,cafe ,bmw ,chien ,1) ,m(norvegien ,bleue ,lait ,honda ,
52 % zebre ,2)]
53 /* Test
54 [eclipse 48]: Ruetest = [m(anglais ,rouge ,cafe ,bmw ,chien ,1) ,m(norvegien ,bleue ,
55 % lait ,honda ,zebre ,2)],ecrit_maisons(Ruetest).
56 m(anglais ,rouge ,cafe ,bmw ,chien ,1)
57 m(norvegien ,bleue ,lait ,honda ,zebre ,2)
58 Ruetest = [m(anglais ,rouge ,cafe ,bmw ,chien ,1) ,m(norvegien ,bleue ,lait ,
59 % honda ,zebre ,2)]
60 Yes (0.00s cpu)
61 */
62
63 getVarList([],[]).
64 getVarList([m(P,C,B,V,A,_I)|Rest],[P,C,B,V,A | Liste]) :- getVarList(Rest,
65                                         Liste).
66 labeling_symbolic(Liste) :- (foreach(Elem,Liste)
67                                     do
68                                         ic_symbolic:
69                                         indomain(
70                                         Elemt
71                                         ).
72
73 rue(R),getVarList(R,L),labeling_symbolic(L).
74
75 R = [m(anglais ,rouge ,cafe ,bmw ,chien ,1) ,m(espagnol ,verte ,toyota ,
76 % serpents ,2) ,m(ukrainien ,blanche ,lait ,ford ,renard ,3) ,m(norvegien ,
```

```

jaune , jasdOrange , honda , cheval , 4) , m(japonais , bleue , eau , datsun ,
zebre , 5)]
76 L = [anglais , rouge , cafe , bmw , chien , espagnol , verte , the , toyota , serpents
, ukrainien , blanche , lait , ford , renard , norvegien , jaune , jasdOrange ,
honda , ...]
77 Yes (0.00s cpu, solution 1, maybe more) ? ;
78
79 R = [m(anglais , rouge , cafe , bmw , chien , 1) , m(espagnol , verte , the , toyota ,
serpents , 2) , m(ukrainien , blanche , lait , ford , renard , 3) , m(norvegien ,
jaune , jasdOrange , honda , zebre , 4) , m(japonais , bleue , eau , datsun ,
cheval , 5)]
80 L = [anglais , rouge , cafe , bmw , chien , espagnol , verte , the , toyota , serpents
, ukrainien , blanche , lait , ford , renard , norvegien , jaune , jasdOrange ,
honda , ...]
81 Yes (0.00s cpu, solution 2, maybe more) ? ;
82
83 R = [m(anglais , rouge , cafe , bmw , chien , 1) , m(espagnol , verte , the , toyota ,
serpents , 2) , m(ukrainien , blanche , lait , ford , renard , 3) , m(norvegien ,
jaune , jasdOrange , datsun , 4) , m(japonais , bleue , eau , honda ,
zebre , 5)]
84 L = [anglais , rouge , cafe , bmw , chien , espagnol , verte , the , toyota , serpents
, ukrainien , blanche , lait , ford , renard , norvegien , jaune , jasdOrange ,
datsun , ...]
85 */
86 */
87
88 contraintes(R) :-      (foreach(m(P,C,B,V,A,I),R)
89           do
90             (P &= anglais) #= (C &= rouge),
91             (P &= espagnol) #= (A &= chien),
92             (B &= cafe) #= (C &= verte),
93             (P &= ukrainien) #= (B &= the),
94             (A &= serpents) #= (V &= bmw),
95             (C &= jaune) #= (V &= toyota),
96             (B &= lait) #= (I #= 3),
97             (P &= norvegien) #= (I #= 1),
98             (B &= jasdOrange) #= (V &= honda),
99             (P &= japonais) #= (V &= datsun)
100            ).
101
102 contraintes2(R) :-      (foreach(m(P1,C1,_B1,V1,_A1,I1),R),param(R)
103           do
104             (foreach(m(_P2,C2,_B2
105               ,_V2,A2,I2),R),
106               param(I1,C1,V1,P1
107               )
108               do
109                 ((C1 &= verte) and (C2 &= blanche)) => (I1 #= I2+1),
110                 ((V1 &= ford) and (A2 &= renard)) => ((I2 #= I1+1) or (I2 #= I1
111                   -1)),
112                 ((V1 &= toyota) and (A2 &= cheval)) => ((I2 #= I1+1) or (I2 #=
113                   I1-1)),
114
115 resoudre(R) :- rue(R),contraintes(R),getVarList(R,L),labeling_symbolic(L),
116   ecrit_maisons(R).
117
118 resoudre2(R) :- rue(R),getVarList(R,L),contraintes2(R),
119   labeling_symbolic(L),ecrit_maisons(R).
```

Rapport Travaux Pratiques :

Programmation par Contraintes

- TP 3 : Contraintes Logiques

Nicolas Desfeux
Aurélien Texier

15 mars 2011

Dans ce T.P., nous allons utiliser la programmation par contraintes pour résoudre un problème d'ordonnancement de tâches à effectuer sur deux machines.

Dans un premier temps, nous définirons les prédicts 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 contrainte qui vise à empêcher que deux tâches se fassent simultanément sur la même machine.

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

Listing 1 – "taches"

```

1 taches(Taches) :-      Taches = [|(tache(3,[1],m1,_),
2                                     tache(8,[1],m1,_),
3                                     tache(8,[4,5],m1,_),
4                                     tache(6,[1],m2,_),
5                                     tache(3,[1],m2,_),
6                                     tache(4,[1,7],m1,_),
7                                     tache(8,[3,5],m1,_),
8                                     tache(6,[4],m2,_),
9                                     tache(6,[16,71],m2,_),
10                                    tache(6,[9,12],m2,_),
11                                    tache(3,[1],m2,_),
12                                    tache(6,[7,8],m2,_)).
13
14 /* Test
15
16 |eclipse 3]: taches(T).
17
18 T = [tache(3, [1], m1, _169), tache(8, [1], m1, _176), tache(8, [4, 5], m1,
19   _183), tache(6, [1], m2, _194), tache(3, [1], m2, _201), tache(4, [1, 7],
20   _210), tache(8, [3, 5], m1, _221), tache(6, [4], m2, _232), tache(6,
```

```

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

```

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

Listing 2 – "affiche"

```

1 affiche(Taches) :-      dim(Taches,[Dim]),
2                               (for(Indice,1,Dim),param(Taches)
3                                         do
4                                         Elem is Taches[Indice
5                                         ],
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, [16, 71], 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   [16, 71], m2, _307), tache(6, [9, 12], m2, _318), tache(3, [1], m2, _329),
28   tache(6, [7, 8], m2, _338)]
29
30 Yes (0.00s cpu)
30 */

```

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

Listing 3 – "domaines"

```

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

```

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

```

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

Listing 4 - "getVarList"

```

1 getVarList(Taches,Fin,ListFin) :- dim(Taches,[Dim]),
2     (for(Indice,1,Dim),fromto([],In,Out,List),
3      param(Taches)
4      do
5          Xi is Taches[Indice],
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, [], m1, _238), tache(8, [], m1, _243), tache(8, [4, 5], m1,
15     _248), tache(6, [], 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 Fin = Fin
20 L = [Fin, _319, _312, _303, _294, _287, _278, _269, _262, _257, _248, _243,
21     _238]
22 Yes (0.00s cpu)
23 */

```

```

19
20 */

```

Question 3.5 On définit le prédictat `solve(?Taches, ?Fin)` qui permet, en utilisant les trois prédictats précédents, de trouver un ordonnancement qui respecte les contraintes de domaines définies.

Le test effectué atteste bien la conformité du prédictat car solve rend bien comme première solution toutes les tâches avec des débuts à 0, puis il incrémenté 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.

Listing 5 - "solve1"

```

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

```

```

38 Taches = [](tache(3, [1], m1, 1), tache(8, [1], m1, 0), tache(8, [4, 5], m1, 0)
   , tache(6, [1], m2, 0), tache(3, [1], m2, 0), tache(4, [1, 7], m1, 0),
   tache(8, [3, 5], m1, 0), tache(6, [4], m2, 0), tache(6, [6, 7], m2, 0),
   tache(6, [9, 12], m2, 0), tache(3, [1], m2, 0), tache(6, [7, 8], m2, 0))
39 Fin = 8
40 Yes (0.01s cpu, solution 2, maybe more) ?
41 tache(3, [1], m1, 2)
42 tache(8, [1], m1, 0)
43 tache(8, [4, 5], m1, 0)
44 tache(6, [1], m2, 0)
45 tache(3, [1], m2, 0)
46 tache(4, [1, 7], m1, 0)
47 tache(8, [3, 5], m1, 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, [1], m1, 2), tache(8, [1], m1, 0), tache(8, [4, 5], m1, 0)
   , tache(6, [1], m2, 0), tache(3, [1], m2, 0), tache(4, [1, 7], m1, 0),
   tache(8, [3, 5], m1, 0), tache(6, [4], m2, 0), tache(6, [6, 7], m2, 0),
   tache(6, [9, 12], m2, 0), tache(3, [1], m2, 0), tache(6, [7, 8], m2, 0))
55 Fin = 8
56 Yes (0.01s cpu, solution 3, maybe more) ?
57 */

```

Question 3.6 On définit ici un prédictat *precedences(+Taches)* qui constraint chaque tâche à démarrer après la fin de ses tâches préliminaires.

On modifie alors solve pour prendre en compte ces contraintes.

Les résultats de solve sont bien conformes car les deux premières tâches commencent à 0 puisqu'elles n'ont besoin d'aucune autre tâche effectuée au préalable pour s'effectuer. La tâche 5 qui a besoin de la tâche 1 effectuée commence bien à 3, qui est la durée de la tâche 1. Tout est donc correct.

On peut donc en conclure que sans la contrainte des machines différentes qui va suivre, la solution optimale prendrait 38 unités de temps pour se faire (car le solveur unifie Fin à 38).

Listing 6 – "precedences"

```

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

```

```

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 [eclipse 44]: solve(T, Fin).
21
22 tache(3, [1], m1, 0)
23 tache(8, [1], m1, 0)
24 tache(8, [4, 5], m1, 6)
25 tache(6, [1], m2, 0)
26 tache(3, [1], m2, 3)
27 tache(4, [1, 7], m1, 22)
28 tache(8, [3, 5], m1, 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, [1], m1, 0), tache(8, [1], m1, 0), tache(8, [4, 5], m1, 6),
36           tache(6, [1], m2, 0), tache(3, [1], m2, 3), tache(4, [1, 7], m1, 22),
37           tache(8, [3, 5], m1, 14), tache(6, [4], m2, 6), tache(6, [6, 7], m2, 26),
38           tache(6, [9, 12], m2, 32), tache(3, [1], m2, 3), tache(6, [7, 8], m2,
39           22))
40 Fin = 38
41
42 */

```

Question 3.7 Enfin, on définit le prédictat *conflicts(+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 – "conflicts"

```

1 conflicts(Taches) :- dim(Taches,[Dim]),
2           (for(Indice,1,Dim),param(Taches,Dim)
3           do
4               Elemt is Taches[Indice],
5               I2 is Indice+1,
6               /* Il faut assurer que les indices soient
5               differents, sinon on va se retrouver a
5               comparer deux fois la meme taches*/
7               (for(I,I2,Dim),param(Taches,Elem)
8               do
9                   Elemt2 is Taches[I],

```

```

10                               machinesDifferentes(Elem,Elem2)
11
12   .
13
14 machinesDifferentes(tache( _, _, M1, _), tache( _, _, M2, _)) :- \=(M1,M2) ,! .
15 machinesDifferentes(tache(Duree, _, M, Debut), tache(Duree2, _, M, Debut2)) :- (((
16     Debut #>= Debut2+Duree2) or (Debut+Duree #< Debut2)).
17
18 solve2(Taches,Fin) :-    taches(Taches),
19
20                                     domaines(Taches,Fin),
21                                     precedences(Taches),
22                                     conflits(Taches),
23                                     getVarList(Taches,Fin,Liste),
24                                     labeling(Liste),
25                                     affiche(Taches).
26 /*
27 [eclipse 10]: solve2(Taches,Fin).
28 tache(3, [], ml, 0)
29 tache(8, [], ml, 29)
30 tache(8, [4], ml, 9)
31 tache(6, [], m2, 0)
32 tache(3, [1], m2, 6)
33 tache(4, [1], ml, 25)
34 tache(8, [3], ml, 17)
35 tache(6, [4], m2, 12)
36 tache(6, [6], ml, 31)
37 tache(6, [9], m2, 37)
38 tache(3, [1], m2, 9)
39 tache(6, [7], m2, 25)
40 Taches = [](tache(3, [], ml, 0), tache(8, [], ml, 29), tache(8, [4], ml, 9),
41             tache(6, [], m2, 0), tache(3, [1], m2, 6), tache(4, [1], ml, 25),
42             tache(8, [3], ml, 17), tache(6, [4], m2, 12), tache(6, [6], ml, 31),
43             , tache(6, [9], m2, 37), tache(3, [1], m2, 9), tache(6, [7], m2,
44             25))
45 Fin = 43
46 Yes (0.01s cpu, solution 1, maybe more) ? ;
47 tache(3, [], ml, 1)
48 tache(8, [], ml, 29)
49 tache(8, [4], ml, 9)
50 tache(6, [], m2, 0)
51 tache(3, [1], m2, 6)
52 tache(4, [1], ml, 25)
53 tache(8, [3], ml, 17)
54 tache(6, [4], m2, 12)
55 tache(6, [6], ml, 31)
56 tache(6, [9], m2, 37)
57 Taches = [](tache(3, [], ml, 1), tache(8, [], ml, 29), tache(8, [4], ml, 9),
58             tache(6, [], m2, 0), tache(3, [1], m2, 6), tache(4, [1], ml, 25),
59             , tache(6, [4], m2, 12), tache(6, [6], ml, 31),
60             tache(6, [9], m2, 37), tache(3, [1], m2, 9), tache(6, [7], m2,
61             25))
62 Fin = 43
63 Yes (0.01s cpu, solution 2, maybe more) ? ;
64 tache(3, [], ml, 2)
65 tache(8, [1], ml, 29)
66 tache(8, [4], ml, 9)
67 tache(6, [], m2, 0)
68 tache(3, [1], m2, 6)
69 tache(4, [1], ml, 25)
70 tache(8, [3], ml, 17)
71 tache(6, [4], m2, 12)
72 tache(6, [6], ml, 31)
73 tache(6, [9], m2, 37)
74 tache(3, [1], m2, 9)
75 tache(6, [7], m2, 25)
76 Taches = [](tache(3, [], ml, 2), tache(8, [1], ml, 29), tache(8, [4], ml, 9),
77             tache(6, [], m2, 0), tache(3, [1], m2, 6), tache(4, [1], ml, 25),
78             tache(8, [3], ml, 17), tache(6, [4], m2, 12), tache(6, [6], ml, 31),
79             , tache(6, [9], m2, 37), tache(3, [1], m2, 9), tache(6, [7], m2,
80             25))
81 Fin = 43
82 Yes (0.01s cpu, solution 3, maybe more) ? ;
83 */
84
85 /* */
86 [eclipse 11]: solve2(Taches,42).
87
88 No (0.00s cpu)
89 */

```

```

tache(8, [3], ml, 17), tache(6, [4], m2, 12), tache(6, [6], ml, 31),
, tache(6, [9], m2, 37), tache(3, [1], m2, 9), tache(6, [7], m2, 25))
57 Fin = 43
58 Yes (0.01s cpu, solution 2, maybe more) ? ;
59 tache(3, [], ml, 2)
60 tache(8, [1], ml, 29)
61 tache(8, [4], ml, 9)
62 tache(6, [], m2, 0)
63 tache(3, [1], m2, 6)
64 tache(4, [1], ml, 25)
65 tache(8, [3], ml, 17)
66 tache(6, [4], m2, 12)
67 tache(6, [6], ml, 31)
68 tache(6, [9], m2, 37)
69 tache(3, [1], m2, 9)
70 tache(6, [7], m2, 25)
71
72 Taches = [](tache(3, [], ml, 2), tache(8, [1], ml, 29), tache(8, [4], ml, 9),
73             tache(6, [], m2, 0), tache(3, [1], m2, 6), tache(4, [1], ml, 25),
74             tache(8, [3], ml, 17), tache(6, [4], m2, 12), tache(6, [6], ml, 31),
75             , tache(6, [9], m2, 37), tache(3, [1], m2, 9), tache(6, [7], m2,
76             25))
77 Fin = 43
78 Yes (0.01s cpu, solution 3, maybe more) ? ;
79 */
80
81 /* */
82 [eclipse 11]: solve2(Taches,42).
83
84 No (0.00s cpu)
85 */

```

Question 3.8 Oui, la solution est la meilleure ! Prolog résoud les contraintes en incrémentant le début des tâches, jusqu'à obtenir le respect des contraintes. Comme il incrémente, la première trouvée est candidate pour la solution optimale au problème, à savoir l'ordonnancement des tâches au plus tôt. Seulement, rien ne nous assure que la première est la meilleure, car cela dépend de l'ordre de la déclaration des tâches au début, dans le prédictat *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.

1 Code Complet, avec l'ensemble des tests

Listing 8 – "TP3"

```

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

```

45 tache(6, [9, 12], m2, _318)
46 tache(3, [1], m2, _329)
47 tache(6, [7, 8], m2, _338)
48
49 T = [tache(3, [], m1, _235), tache(8, [], m1, _242), tache(8, [4, 5], m1,
50      _249), tache(6, [], m2, _260), tache(3, [1], m2, _267), tache(4, [1, 7],
51      m1, _276), tache(8, [3, 5], m1, _287), tache(6, [4], m2, _298), tache(6,
52      [6, 7], m2, _307), tache(6, [9, 12], m2, _318), tache(3, [1], m2, _329),
53      tache(6, [7, 8], m2, _338)]
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                   Debut #>= 0
64 ). /* [eclipse 5]: taches(T),domaines(T,10).
65
66 T = [tache(3, [], m1, _421{0 .. 7}), tache(8, [], m1, _644{0 .. 2}), tache(8,
67      [4, 5], m1, _867{0 .. 2}), tache(6, [], m2, _1090{0 .. 4}), tache(3,
68      [1], m2, _1313{0 .. 7}), tache(4, [1, 7], m1, _1536{0 .. 6}), tache(8,
69      [3, 5], m1, _1759{0 .. 2}), tache(6, [4], m2, _1982{0 .. 4}), tache(6,
70      [6, 7], m2, _2205{0 .. 4}), tache(6, [9, 12], m2, _2428{0 .. 4}), tache
71      (3, [1], m2, _2651{0 .. 7}), tache(6, [7, 8], m2, _2874{0 .. 4})]
72
73 There are 12 delayed goals. Do you want to see them? (y\n)
74 Yes (0.00s cpu)
75 */
76
77 getVarList(Taches,Fin,ListFin) :- dim(Taches,[Dim]),
78          (for(Indice,1,Dim),fromto([],In,Out,List),
79             param(Taches)
80                 do
81                   Xi is Taches[Indice],
82                   Xi = tache(_,_,_Debut),
83                   Out=[Debut|In]
84             ),
85             ListFin=[Fin|List].
86
87
88 /*
```

```

84 [eclipse 6]: taches(T), getVarList(T, Fin, L).
85
86 T = [](tache(3, [], m1, _238), tache(8, [], m1, _243), tache(8, [4, 5], m1,
87 _248), tache(6, [], m2, _257), tache(3, [1], m2, _262), tache(4, [1, 7],
88 m1, _269), tache(8, [3, 5], m1, _278), tache(6, [4], m2, _287), tache(6,
89 [6, 7], m2, _294), tache(6, [9, 12], m2, _303), tache(3, [1], m2, _312),
90 tache(6, [7, 8], m2, _319))
91 Fin = Fin
92 L = [Fin, _319, _312, _303, _294, _287, _278, _269, _262, _257, _248, _243,
93 _238]
94 Yes (0.00s cpu)
95 */
96 solve(Taches, Fin) :- taches(Taches),
97 domaines(Taches, Fin),
98 precedences(Taches),
99 getVarList(Taches, Fin, Liste),
100 labeling(Liste),
101 affiche(Taches).
102 precedences(Taches) :- dim(Taches,[Dim]),
103 (for(Indice,1,Dim),param(Taches)
104 do
105 Elem is Taches[Indice],
106 Elemtache(_D,Noms,_M,Debut),
107 (foreach(I,Noms),param(Debut,Taches)
108 do
109 tache(Duree2,_N,_M,Debut2) is Taches[
110 I],
111 Debut #>= Debut2+Duree2
112
113 /* [eclipse 44]: taches(T), solve(T, Fin).
114 tache(3, [], m1, 0)
115 tache(8, [], m1, 0)
116 tache(8, [4, 5], m1, 6)
117 tache(6, [], m2, 0)
118 tache(3, [1], m2, 3)
119 tache(4, [1, 7], m1, 22)
120 tache(8, [3, 5], m1, 14)
121 tache(6, [4], m2, 6)
122 tache(6, [6, 7], m2, 26)
123 tache(6, [9, 12], m2, 32)
124 tache(3, [1], m2, 3)
125 tache(6, [7, 8], m2, 22)
126 T = [](tache(3, [], m1, 0), tache(8, [], m1, 0), tache(8, [4, 5], m1, 6),
127 tache(6, [], m2, 0), tache(3, [1], m2, 3), tache(4, [1, 7], m1, 22),
128 tache(8, [3, 5], m1, 14), tache(6, [4], m2, 6), tache(6, [6, 7], m2, 26),
129 tache(6, [9, 12], m2, 32), tache(3, [1], m2, 3), tache(6, [7, 8], m2,
130 22))

```

```

127 Fin = 38
128 */
129 conflits(Taches) :- dim(Taches,[Dim]),
130 (for(Indice,1,Dim),param(Taches,Dim)
131 do
132 Elemtache(_D,Noms,_M,Debut),
133 Elemtache(_D,Noms,_M,Debut),
134 (for(I,I2,Dim),param(Taches,Elem)
135 do
136 Elemtache(_D,Noms,_M,Debut),
137 Elemtache(_D,Noms,_M,Debut),
138 machinesDifferentes(Elem,Elem2)
139 machinesDifferentes(tache(_,_,_M1,_),tache(_,_,_M2,_)):- \=(M1,M2),!.
140 machinesDifferentes(tache(Duree,_,_M,Debut),tache(Duree2,_,_M,Debut2)) :- (
141 Debut #>= Debut2+Duree2) or (Debut+Duree #< Debut2)).
142
143 solve2(Taches, Fin) :- taches(Taches),
144 domaines(Taches, Fin),
145 precedences(Taches),
146 getVarList(Taches, Fin, Liste),
147 labeling(Liste),
148 affiche(Taches).
149 /* [eclipse 10]: solve2(Taches, Fin).
150 tache(3, [], m1, 0)
151 tache(8, [], m1, 29)
152 tache(8, [4, 5], m1, 9)
153 tache(6, [], m2, 0)
154 tache(3, [1], m2, 6)
155 tache(4, [1, 7], m1, 25)
156 tache(8, [3, 5], m1, 17)
157 tache(6, [4], m2, 12)
158 tache(6, [6, 7], m2, 31)
159 tache(6, [9, 12], m2, 37)
160 tache(3, [1], m2, 9)
161 tache(6, [7, 8], m2, 25)
162
163 Taches = [](tache(3, [], m1, 0), tache(8, [], m1, 29), tache(8, [4, 5], m1,
164 9), tache(6, [], m2, 0), tache(3, [1], m2, 6), tache(4, [1, 7], m1, 25),
165 tache(8, [3, 5], m1, 17), tache(6, [4], m2, 12), tache(6, [6, 7], m2, 31),
166 , tache(6, [9, 12], m2, 37), tache(3, [1], m2, 9), tache(6, [7, 8], m2,
167 25))
167 Fin = 43
168 Yes (0.01s cpu, solution 1, maybe more) ?
169 tache(3, [], m1, 1)
170 tache(8, [], m1, 29)

```

```

173 tache(8, [4, 5], ml, 9)
174 tache(6, [], m2, 0)
175 tache(3, [1], m2, 6)
176 tache(4, [1, 7], ml, 25)
177 tache(8, [3, 5], ml, 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, [], ml, 1), tache(8, [], ml, 29), tache(8, [4, 5], ml,
   9), tache(6, [], 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))
185 Fin = 43
186 Yes (0.01s cpu, solution 2, maybe more) ? ;
187 tache(3, [], ml, 2)
188 tache(8, [], ml, 29)
189 tache(8, [4, 5], ml, 9)
190 tache(6, [], 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, [], ml, 2), tache(8, [], ml, 29), tache(8, [4, 5], ml,
   9), tache(6, [], 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 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 */

```

Rapport Travaux Pratiques :

Programmation par Contraintes

- TP 3 : Contraintes Logiques

Nicolas Desfeux
Aurélien Texier

15 mars 2011

Dans ce T.P., nous allons utiliser la programmation par contraintes pour résoudre un problème d'ordonnancement de tâches à effectuer sur deux machines.

Dans un premier temps, nous définirons les prédicts 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 contrainte qui vise à empêcher que deux tâches se fassent simultanément sur la même machine.

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

Listing 1 – "taches"

```

1 taches(Taches) :-      Taches = [|(tache(3,[1],m1,_),
2                                     tache(8,[1],m1,_),
3                                     tache(8,[4,5],m1,_),
4                                     tache(6,[1],m2,_),
5                                     tache(3,[1],m2,_),
6                                     tache(4,[1,7],m1,_),
7                                     tache(8,[3,5],m1,_),
8                                     tache(6,[4],m2,_),
9                                     tache(6,[16,71],m2,_),
10                                    tache(6,[9,12],m2,_),
11                                    tache(3,[1],m2,_),
12                                    tache(6,[7,8],m2,_)).
13
14 /* Test
15
16 |eclipse 3]: taches(T).
17
18 T = [tache(3, [1], m1, _169), tache(8, [1], m1, _176), tache(8, [4, 5], m1,
19   _183), tache(6, [1], m2, _194), tache(3, [1], m2, _201), tache(4, [1, 7],
20   _210), tache(8, [3, 5], m1, _221), tache(6, [4], m2, _232), tache(6,
```

```

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

```

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

Listing 2 – "affiche"

```

1 affiche(Taches) :-      dim(Taches,[Dim]),
2                               (for(Indice,1,Dim),param(Taches)
3                                         do
4                                         Elem is Taches[Indice
5                                         ],
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, [16, 71], 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   [16, 71], m2, _307), tache(6, [9, 12], m2, _318), tache(3, [1], m2, _329),
28   tache(6, [7, 8], m2, _338)]
29
30 Yes (0.00s cpu)
30 */

```

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

Listing 3 – "domaines"

```

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

```

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

```

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

Listing 4 - "getVarList"

```

1 getVarList(Taches,Fin,ListFin) :- dim(Taches,[Dim]),
2     (for(Indice,1,Dim),fromto([],In,Out,List),
3      param(Taches)
4      do
5          Xi is Taches[Indice],
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, [], m1, _238), tache(8, [], m1, _243), tache(8, [4, 5], m1,
15     _248), tache(6, [], 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 Fin = Fin
20 L = [Fin, _319, _312, _303, _294, _287, _278, _269, _262, _257, _248, _243,
21     _238]
22 Yes (0.00s cpu)
23 */

```

```

19
20 */

```

Question 3.5 On définit le prédictat `solve(?Taches, ?Fin)` qui permet, en utilisant les trois prédictats précédents, de trouver un ordonnancement qui respecte les contraintes de domaines définies.

Le test effectué atteste bien la conformité du prédictat car solve rend bien comme première solution toutes les tâches avec des débuts à 0, puis il incrémenté 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.

Listing 5 - "solve1"

```

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

```

```

38 Taches = [](tache(3, [1], m1, 1), tache(8, [1], m1, 0), tache(8, [4, 5], m1, 0)
   , tache(6, [1], m2, 0), tache(3, [1], m2, 0), tache(4, [1, 7], m1, 0),
   tache(8, [3, 5], m1, 0), tache(6, [4], m2, 0), tache(6, [6, 7], m2, 0),
   tache(6, [9, 12], m2, 0), tache(3, [1], m2, 0), tache(6, [7, 8], m2, 0))
39 Fin = 8
40 Yes (0.01s cpu, solution 2, maybe more) ?
41 tache(3, [1], m1, 2)
42 tache(8, [1], m1, 0)
43 tache(8, [4, 5], m1, 0)
44 tache(6, [1], m2, 0)
45 tache(3, [1], m2, 0)
46 tache(4, [1, 7], m1, 0)
47 tache(8, [3, 5], m1, 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, [1], m1, 2), tache(8, [1], m1, 0), tache(8, [4, 5], m1, 0)
   , tache(6, [1], m2, 0), tache(3, [1], m2, 0), tache(4, [1, 7], m1, 0),
   tache(8, [3, 5], m1, 0), tache(6, [4], m2, 0), tache(6, [6, 7], m2, 0),
   tache(6, [9, 12], m2, 0), tache(3, [1], m2, 0), tache(6, [7, 8], m2, 0))
55 Fin = 8
56 Yes (0.01s cpu, solution 3, maybe more) ?
57 */

```

Question 3.6 On définit ici un prédictat *precedences(+Taches)* qui constraint chaque tâche à démarrer après la fin de ses tâches préliminaires.

On modifie alors solve pour prendre en compte ces contraintes.

Les résultats de solve sont bien conformes car les deux premières tâches commencent à 0 puisqu'elles n'ont besoin d'aucune autre tâche effectuée au préalable pour s'effectuer. La tâche 5 qui a besoin de la tâche 1 effectuée commence bien à 3, qui est la durée de la tâche 1. Tout est donc correct.

On peut donc en conclure que sans la contrainte des machines différentes qui va suivre, la solution optimale prendrait 38 unités de temps pour se faire (car le solveur unifie Fin à 38).

Listing 6 – "precedences"

```

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

```

```

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 [eclipse 44]: solve(T, Fin).
21
22 tache(3, [1], m1, 0)
23 tache(8, [1], m1, 0)
24 tache(8, [4, 5], m1, 6)
25 tache(6, [1], m2, 0)
26 tache(3, [1], m2, 3)
27 tache(4, [1, 7], m1, 22)
28 tache(8, [3, 5], m1, 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, [1], m1, 0), tache(8, [1], m1, 0), tache(8, [4, 5], m1, 6),
36           tache(6, [1], m2, 0), tache(3, [1], m2, 3), tache(4, [1, 7], m1, 22),
37           tache(8, [3, 5], m1, 14), tache(6, [4], m2, 6), tache(6, [6, 7], m2, 26),
38           tache(6, [9, 12], m2, 32), tache(3, [1], m2, 3), tache(6, [7, 8], m2,
39           22))
40 Fin = 38
41
42 */

```

Question 3.7 Enfin, on définit le prédictat *conflicts(+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 – "conflicts"

```

1 conflicts(Taches) :- dim(Taches,[Dim]),
2           (for(Indice,1,Dim),param(Taches,Dim)
3           do
4               Elemt is Taches[Indice],
5               I2 is Indice+1,
6               /* Il faut assurer que les indices soient
5               differents, sinon on va se retrouver a
5               comparer deux fois la meme taches*/
7               (for(I,I2,Dim),param(Taches,Elem)
8               do
9                   Elemt2 is Taches[I],

```

```

10                               machinesDifferentes(Elem,Elem2)
11
12   .
13
14 machinesDifferentes(tache( _, _, M1, _), tache( _, _, M2, _)) :- \=(M1,M2) ,! .
15 machinesDifferentes(tache(Duree, _, M, Debut), tache(Duree2, _, M, Debut2)) :- (((
16     Debut #>= Debut2+Duree2) or (Debut+Duree #< Debut2)).
17
18 solve2(Taches,Fin) :-    taches(Taches),
19
20                                     domaines(Taches,Fin),
21                                     precedences(Taches),
22                                     conflits(Taches),
23                                     getVarList(Taches,Fin,Liste),
24                                     labeling(Liste),
25                                     affiche(Taches).
26 /*
27 [eclipse 10]: solve2(Taches,Fin).
28 tache(3, [], ml, 0)
29 tache(8, [], ml, 29)
30 tache(8, [4], ml, 9)
31 tache(6, [], m2, 0)
32 tache(3, [1], m2, 6)
33 tache(4, [1], ml, 25)
34 tache(8, [3], ml, 17)
35 tache(6, [4], m2, 12)
36 tache(6, [6], ml, 31)
37 tache(6, [9], m2, 37)
38 tache(3, [1], m2, 9)
39 tache(6, [7], m2, 25)
40 Taches = [](tache(3, [], ml, 0), tache(8, [], ml, 29), tache(8, [4], ml, 9),
41             tache(6, [], m2, 0), tache(3, [1], m2, 6), tache(4, [1], ml, 25),
42             tache(8, [3], ml, 17), tache(6, [4], m2, 12), tache(6, [6], ml, 31),
43             , tache(6, [9], m2, 37), tache(3, [1], m2, 9), tache(6, [7], m2,
44             25))
45 Fin = 43
46 Yes (0.01s cpu, solution 1, maybe more) ? ;
47 tache(3, [], ml, 1)
48 tache(8, [], ml, 29)
49 tache(8, [4], ml, 9)
50 tache(6, [], m2, 0)
51 tache(3, [1], m2, 6)
52 tache(4, [1], ml, 25)
53 tache(8, [3], ml, 17)
54 tache(6, [4], m2, 12)
55 tache(6, [6], ml, 31)
56 tache(6, [9], m2, 37)
57 Taches = [](tache(3, [], ml, 1), tache(8, [], ml, 29), tache(8, [4], ml, 9),
58             tache(6, [], m2, 0), tache(3, [1], m2, 6), tache(4, [1], ml, 25),
59             , tache(6, [4], m2, 12), tache(6, [6], ml, 31),
60             tache(6, [9], m2, 37), tache(3, [1], m2, 9), tache(6, [7], m2,
61             25))
62 Fin = 43
63 Yes (0.01s cpu, solution 2, maybe more) ? ;
64 tache(3, [], ml, 2)
65 tache(8, [1], ml, 29)
66 tache(8, [4], ml, 9)
67 tache(6, [], m2, 0)
68 tache(3, [1], m2, 6)
69 tache(4, [1], ml, 25)
70 tache(8, [3], ml, 17)
71 tache(6, [4], m2, 12)
72 tache(6, [6], ml, 31)
73 tache(6, [9], m2, 37)
74 tache(3, [1], m2, 9)
75 tache(6, [7], m2, 25)
76 Taches = [](tache(3, [], ml, 2), tache(8, [1], ml, 29), tache(8, [4], ml, 9),
77             tache(6, [], m2, 0), tache(3, [1], m2, 6), tache(4, [1], ml, 25),
78             tache(8, [3], ml, 17), tache(6, [4], m2, 12), tache(6, [6], ml, 31),
79             , tache(6, [9], m2, 37), tache(3, [1], m2, 9), tache(6, [7], m2,
80             25))
81 Fin = 43
82 Yes (0.01s cpu, solution 3, maybe more) ? ;
83 */
84
85 /* */
86 [eclipse 11]: solve2(Taches,42).
87
88 No (0.00s cpu)
89 */

```

```

tache(8, [3], ml, 17), tache(6, [4], m2, 12), tache(6, [6], ml, 31),
, tache(6, [9], m2, 37), tache(3, [1], m2, 9), tache(6, [7], m2, 25))
57 Fin = 43
58 Yes (0.01s cpu, solution 2, maybe more) ? ;
59 tache(3, [], ml, 2)
60 tache(8, [1], ml, 29)
61 tache(8, [4], ml, 9)
62 tache(6, [], m2, 0)
63 tache(3, [1], m2, 6)
64 tache(4, [1], ml, 25)
65 tache(8, [3], ml, 17)
66 tache(6, [4], m2, 12)
67 tache(6, [6], ml, 31)
68 tache(6, [9], m2, 37)
69 tache(3, [1], m2, 9)
70 tache(6, [7], m2, 25)
71
72 Taches = [](tache(3, [], ml, 2), tache(8, [1], ml, 29), tache(8, [4], ml, 9),
73             tache(6, [], m2, 0), tache(3, [1], m2, 6), tache(4, [1], ml, 25),
74             tache(8, [3], ml, 17), tache(6, [4], m2, 12), tache(6, [6], ml, 31),
75             , tache(6, [9], m2, 37), tache(3, [1], m2, 9), tache(6, [7], m2,
76             25))
77 Fin = 43
78 Yes (0.01s cpu, solution 3, maybe more) ? ;
79 */
80
81 /* */
82 [eclipse 11]: solve2(Taches,42).
83
84 No (0.00s cpu)
85 */

```

Question 3.8 Oui, la solution est la meilleure ! Prolog résoud les contraintes en incrémentant le début des tâches, jusqu'à obtenir le respect des contraintes. Comme il incrémente, la première trouvée est candidate pour la solution optimale au problème, à savoir l'ordonnancement des tâches au plus tôt. Seulement, rien ne nous assure que la première est la meilleure, car cela dépend de l'ordre de la déclaration des tâches au début, dans le prédictat *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.

1 Code Complet, avec l'ensemble des tests

Listing 8 – "TP3"

```

1 :-lib(ic).
2 :-lib(ic_symbolic).
3
4 taches(Taches) :- Taches = []
5   [](tache(3,[],ml,_),
6    tache(8,[],ml,_),
7    tache(8,[4,5],ml,_),
8    tache(6,[],m2,_),
9    tache(3,[1],m2,_),
10   tache(4,[1,7],ml,_),
11   tache(8,[3,5],ml,_),
12   tache(6,[4],m2,_),
13   tache(6,[6,7],m2,_),
14   tache(6,[9,12],m2,_),
15   tache(3,[1],m2,_),
16   tache(6,[7,8],m2,_)).
17 /* Test
18
19 [eclipse 3]: taches(T).
20
21 T = [tache(3, [], ml, _169), tache(8, [], ml, _176), tache(8, [4, 5], ml,
22   _183), tache(6, [], m2, _194), tache(3, [1], m2, _201), tache(4, [1, 7],
23   ml, _210), tache(8, [3, 5], ml, _221), tache(6, [4], m2, _232), tache(6,
24   [6, 7], m2, _241), tache(6, [9, 12], m2, _252), tache(3, [1], m2, _263),
25   tache(6, [7, 8], m2, _272)]
26 Yes (0.00s cpu)
27 */
28
29 affiche(Taches) :- dim(Taches,[Dim]),
30   (for(Indice,1,Dim),param(Taches)
31     do
32       Elem is Taches[Indice]
33       writeln(Elem))
34 /*
35 [eclipse 4]: taches(T),affiche(T).
36 tache(3, [], ml, _235)
37 tache(8, [], ml, _242)
38 tache(8, [4, 5], ml, _249)
39 tache(6, [], 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], m1, _235), tache(8, [], m1, _242), tache(8, [4, 5], m1,
    _249), tache(6, [], m2, _260), tache(3, [1], m2, _267), tache(4, [1, 7],
    m1, _276), tache(8, [3, 5], m1, _287), tache(6, [4], m2, _298), tache(6,
    [6, 7], m2, _307), tache(6, [9, 12], m2, _318), tache(3, [1], m2, _329),
    tache(6, [7, 8], m2, _338)]
50 Yes (0.00s cpu)
51 */
52 */
53 */
54
55 domaines(Taches ,Fin) :- dim(Taches ,[Dim]) ,
56             (for(Indice ,1,Dim),param(Taches ,Fin)
57              do
58                  tache(Duree ,_Nom,
59                         _Machine ,Debut)
60                         is Taches[Indice
61                         ],
62                         Debut + Duree #=< Fin
63                         ,
64                         Debut #>= 0
65             ). .
66
67 */
68 There are 12 delayed goals. Do you want to see them? (y\n)
69 Yes (0.00s cpu)
70 */
71
72
73 getVarList(Taches ,Fin ,ListFin):- dim(Taches ,[Dim]),
74             (for(Indice ,1,Dim),fromto([],In ,Out ,List),
75                 param(Taches)
76                 do
77                     Xi is Taches[Indice ],
78                     Xi = tache(_,_,_ ,Debut) ,
79                     Out=[Debut | In]
80                     ),
81                     ListFin=[Fin | List].
82
83 /*

```

```

84 [eclipse 6]: taches(T), getVarList(T, Fin, L).
85
86 T = [](tache(3, [], m1, _238), tache(8, [], m1, _243), tache(8, [4, 5], m1,
87 _248), tache(6, [], m2, _257), tache(3, [1], m2, _262), tache(4, [1, 7],
88 m1, _269), tache(8, [3, 5], m1, _278), tache(6, [4], m2, _287), tache(6,
89 [6, 7], m2, _294), tache(6, [9, 12], m2, _303), tache(3, [1], m2, _312),
90 tache(6, [7, 8], m2, _319))
91 Fin = Fin
92 L = [Fin, _319, _312, _303, _294, _287, _278, _269, _262, _257, _248, _243,
93 _238]
94 Yes (0.00s cpu)
95 */
96 solve(Taches, Fin) :- taches(Taches),
97 domaines(Taches, Fin),
98 precedences(Taches),
99 getVarList(Taches, Fin, Liste),
100 labeling(Liste),
101 affiche(Taches).
102 precedences(Taches) :- dim(Taches,[Dim]),
103 (for(Indice,1,Dim),param(Taches)
104 do
105 Elem is Taches[Indice],
106 Elemtache(_D,Noms,_M,Debut),
107 (foreach(I,Noms),param(Debut,Taches)
108 do
109 tache(Duree2,_N,_M,Debut2) is Taches[
110 I],
111 Debut #>= Debut2+Duree2
112
113 /* [eclipse 44]: taches(T), solve(T, Fin).
114 tache(3, [], m1, 0)
115 tache(8, [], m1, 0)
116 tache(8, [4, 5], m1, 6)
117 tache(6, [], m2, 0)
118 tache(3, [1], m2, 3)
119 tache(4, [1, 7], m1, 22)
120 tache(8, [3, 5], m1, 14)
121 tache(6, [4], m2, 6)
122 tache(6, [6, 7], m2, 26)
123 tache(6, [9, 12], m2, 32)
124 tache(3, [1], m2, 3)
125 tache(6, [7, 8], m2, 22)
126 T = [](tache(3, [], m1, 0), tache(8, [], m1, 0), tache(8, [4, 5], m1, 6),
127 tache(6, [], m2, 0), tache(3, [1], m2, 3), tache(4, [1, 7], m1, 22),
128 tache(8, [3, 5], m1, 14), tache(6, [4], m2, 6), tache(6, [6, 7], m2, 26),
129 tache(6, [9, 12], m2, 32), tache(3, [1], m2, 3), tache(6, [7, 8], m2,
130 22))

```

```

127 Fin = 38
128 */
129 conflits(Taches) :- dim(Taches,[Dim]),
130 (for(Indice,1,Dim),param(Taches,Dim)
131 do
132 Elemtache(_D,Noms,_M,Debut),
133 Elemtache(_D,Noms,_M,Debut),
134 (for(I,I2,Dim),param(Taches,Elem)
135 do
136 Elemtache(_D,Noms,_M,Debut),
137 Elemtache(_D,Noms,_M,Debut),
138 machinesDifferentes(Elem,Elem2)
139 machinesDifferentes(tache(_,_,_M1,_),tache(_,_,_M2,_)):- \=(M1,M2),!.
140 machinesDifferentes(tache(Duree,_,_M,Debut),tache(Duree2,_,_M,Debut2)) :- (
141 Debut #>= Debut2+Duree2) or (Debut+Duree #< Debut2)).
142
143 solve2(Taches, Fin) :- taches(Taches),
144 domaines(Taches, Fin),
145 precedences(Taches),
146 getVarList(Taches, Fin, Liste),
147 labeling(Liste),
148 affiche(Taches).
149 /* [eclipse 10]: solve2(Taches, Fin).
150 tache(3, [], m1, 0)
151 tache(8, [], m1, 29)
152 tache(8, [4, 5], m1, 9)
153 tache(6, [], m2, 0)
154 tache(3, [1], m2, 6)
155 tache(4, [1, 7], m1, 25)
156 tache(8, [3, 5], m1, 17)
157 tache(6, [4], m2, 12)
158 tache(6, [6, 7], m2, 31)
159 tache(6, [9, 12], m2, 37)
160 tache(3, [1], m2, 9)
161 tache(6, [7, 8], m2, 25)
162
163 Taches = [](tache(3, [], m1, 0), tache(8, [], m1, 29), tache(8, [4, 5], m1,
164 9), tache(6, [], m2, 0), tache(3, [1], m2, 6), tache(4, [1, 7], m1, 25),
165 tache(8, [3, 5], m1, 17), tache(6, [4], m2, 12), tache(6, [6, 7], m2, 31),
166 , tache(6, [9, 12], m2, 37), tache(3, [1], m2, 9), tache(6, [7, 8], m2,
167 25))
167 Fin = 43
168 Yes (0.01s cpu, solution 1, maybe more) ?
169 tache(3, [], m1, 1)
170 tache(8, [], m1, 29)

```

```

173 tache(8, [4, 5], ml, 9)
174 tache(6, [], m2, 0)
175 tache(3, [1], m2, 6)
176 tache(4, [1, 7], ml, 25)
177 tache(8, [3, 5], ml, 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, [], ml, 1), tache(8, [], ml, 29), tache(8, [4, 5], ml,
   9), tache(6, [], 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))
185 Fin = 43
186 Yes (0.01s cpu, solution 2, maybe more) ? ;
187 tache(3, [], ml, 2)
188 tache(8, [], ml, 29)
189 tache(8, [4, 5], ml, 9)
190 tache(6, [], 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, [], ml, 2), tache(8, [], ml, 29), tache(8, [4, 5], ml,
   9), tache(6, [], 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 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 */

```

Rapport Travaux Pratiques :

Programmation par Contraintes

- TP 4 :

Contraintes Logiques

Nicolas Desfeux
Aurélien Texier

30 mars 2011

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

Question 4.1 Nous définissons ici un prédictat `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 */

```

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

Listing 2 – "defineVars"

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

```

2         dim(T, [ NbEquipes ,NbConf]) ,
3         ( for(Ind1 ,1 ,NbEquipes ),param(T,NbBateaux ,NbConf)
4           do
5             ( for(Ind2 ,1 ,NbConf),param(T,Ind1 ,NbBateaux )
6               do
7                 T[ Ind1 ,Ind2 ] #:: 1..NbBateaux
8               )
9             ).
10
11 /* Tests
12 [eclipse 8]: getData(TailleEq ,NbEq ,CapaBat ,NbBat ,NbConf) ,defineVars(T,NbEq ,
13 NbConf ,NbBat).
14
15 TailleEq = [](5, 5, 2, 1)
16 NbEq = 4
17 CapaBat = [](7, 6, 5)
18 NbBat = 3
19 NbConf = 3
20
21 T = [](1)(-419{1 .. 3}, -488{1 .. 3}, -557{1 .. 3}), [](-628{1 .. 3}), -697{1
22 .. 3}, -766{1 .. 3}), [](-837{1 .. 3}), -906{1 .. 3}, -975{1 .. 3}), [](
23 -1046{1 .. 3}, -1115{1 .. 3}, -1184{1 .. 3}))
24
25 Yes (0.00s cpu)
26 */

```

Question 4.3 Nous définissons ici un prédictat `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
23
24 */

```

```

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

```

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

Listing 4 – "solve1"

```

1 solve1(T) :-  

2     getData(_TailleEquipes ,NbEquipes ,_CapaBateaux ,NbBateaux ,NbConf),  

3     defineVars(T,NbEquipes ,NbConf ,NbBateaux ),  

4     getVarList(T,L),  

5     labeling(L).  

6  

7 /* Tests  

8 [eclipse 10]: solve1(T).  

9  

10 T = [J(I)(1, 1, 1), IJ(1, 1, 1), II(1, 1, 1), IJ(1, 1, 1))  

11 Yes (0.00s cpu, solution 1, maybe more) ? ;  

12  

13 T = [J(I)(1, 1, 1), II(1, 1, 1), II(1, 1, 1), IJ(1, 1, 2))  

14 Yes (0.00s cpu, solution 2, maybe more) ? ;  

15  

16 T = [J(I)(1, 1, 1), II(1, 1, 1), II(1, 1, 1), II(1, 1, 3))  

17 Yes (0.00s cpu, solution 3, maybe more) ?  

18 */

```

Question 4.5 Nous définissons ici un prédictat *pasMemeBateaux(+T,+NbEquipes,+NbConf)* qui impose qu'une même équipe ne retourne pas deux fois sur le même bateau. On modifie ensuite le prédictat *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
        )
        do
        Bat is T[Indice1 ,Indice2 ],
        append(In ,[ Bat ],Out)
        ),
        allDifferent(L)
    ).
12

```

```

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 = [J(I)(1, 2, 3), IJ(1, 2, 3), II(1, 2, 3), IJ(1, 2, 3))  

24 Yes (0.00s cpu, solution 1, maybe more) ? ;  

25  

26 T = [J(I)(1, 2, 3), II(1, 2, 3), II(1, 2, 3), IJ(1, 3, 2))  

27 Yes (0.00s cpu, solution 2, maybe more) ? ;  

28  

29 T = [J(I)(1, 2, 3), II(1, 2, 3), II(1, 3, 2), II(1, 2, 3))  

30 Yes (0.00s cpu, solution 3, maybe more) ?  

31 */

```

Question 4.6 Nous définissons ici un prédictat *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 prédictat *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             ( for(Conf ,1,NbConf) ,param(T,Equipe1 ,Equipe2) ,fromto
9                 (0,In ,Out,Tot)
10                do
11                    Bateau1 is T[Equipe1 ,Conf ],
12                    Bateau2 is T[Equipe2 ,Conf ],
13                    #=(Bateau1 ,Bateau2 ,Ans) ,
14                    Out #= In + Ans
15                ),
16                Tot #=< 1
17            )
18        .
19  

20 solve3(T) :-  

21     getData(_TailleEquipes ,NbEquipes ,_CapaBateaux ,NbBateaux ,NbConf),  

22     defineVars(T,NbEquipes ,NbConf ,NbBateaux ),  

23     pasMemeBateaux(T,NbEquipes ,NbConf),
24     pasMemePartenaires(T,NbEquipes ,NbConf),
25     getVarList(T,L),
26     labeling(L).

```

```

27  /* Tests
28  [eclipse 12]: solve3(T).
29
30  T = [J([1, 2, 3), [J(1, 3, 2), [J(2, 1, 3), [J(2, 3, 1))
31  Yes (0.00s cpu, solution 1, maybe more) ?
32
33  T = [J([1, 2, 3), [J(1, 3, 2), [J(2, 3, 1), [J(2, 1, 3))
34  Yes (0.00s cpu, solution 2, maybe more) ?
35
36  T = [J([1, 3, 2), [J(1, 2, 3), [J(2, 1, 3), [J(2, 3, 1))
37  Yes (0.00s cpu, solution 3, maybe more) ?
38
39 */

```

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

Listing 7 – "capaBateaux"

```

1 capaBateaux(T, TailleEquipes ,NbEquipes ,CapaBateaux ,NbBateaux ,NbConf):-
2     dim(T,[ NbEquipes ,NbConf]),
3     ( for(Bateau,1,NbBateaux), param(T,NbEquipes ,NbConf, CapaBateaux ,
4           TailleEquipes)
5       do
6         ( for(Conf,1,NbConf), param(T,NbEquipes ,Bateau ,CapaBateaux ,
7             TailleEquipes)
8           do
9             ( for(Equipe,1,NbEquipes ),param(T,Bateau ,Conf,
10               TailleEquipes ),fromto(0,In ,Out ,Total)
11                 do
12                   BateauI is T[Equipe ,Conf],
13                   #=(Bateau ,BateauI ,Cond),
14                   Inc #= TailleEquipes [Equipe] * Cond,
15                   Out #= In + Inc
16               ),
17               Capacite is CapaBateaux [Bateau ],
18               Total #=< Capacite
19           )
20       )
21 solve4(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     capaBateaux(T,TailleEquipes ,NbEquipes ,CapaBateaux ,NbBateaux ,NbConf),
27     getVarList(T,L),
28     labeling(L).
29 /* Tests
29 [eclipse 13]: solve4(T).
30

```

```

31  T = [J([1, 2, 3), [J(2, 3, 1), [J(3, 1, 2), [J(3, 2, 1))
32  Yes (0.01s cpu, solution 1, maybe more) ?
33
34  T = [J([1, 3, 2), [J(2, 1, 3), [J(3, 2, 1), [J(3, 1, 2))
35  Yes (0.01s cpu, solution 2, maybe more) ?
36
37  T = [J([1, 2, 3), [J(3, 1, 2), [J(2, 3, 1), [J(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 voiliers et de 29 équipes qui doivent effectuer la régate comportant 7 confrontations. Le temps d'exécution étant relativement long, il nous a été proposé d'améliorer le labeling. Pour cela, nous avons 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 labeling"

```

1 getData2(TailleEquipes ,NbEquipes ,CapaBateaux ,NbBateaux ,NbConf):-
2     TailleEquipes =
3     [ ](7,6,5,5,4,4,4,4,4,4,4,4,3,3,2,2,2,2,2,2,2,2,2,2,2),
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
8 solve5(T):-
9     getData2(TailleEq ,NbEq ,CapaBat ,NbBat ,NbConf),
10    defineVars(T,NbEq ,NbConf ,NbBat ),
11    pasMemeBateaux(T,NbEq ,NbConf),
12    pasMemePartenaires(T,NbEq ,NbConf),
13    capaBateaux(T,TailleEq ,NbEq ,CapaBat ,NbBat ,NbConf),
14    getVarList(T,L),
15    labeling(L).
16
17 /* Tests
18 [eclipse 14]: solve5(T).
19
20 T = [J([1, 2, 3, 4, 5, 6, 7), [J(2, 1, 4, 3, 6, 5, 8), [J(3, 4, 1, 2, 7, 8,
21   5), [J(4, 3, 1, 5, 2, 7, 6), [J(5, 6, 2, 1, 3, 4, 9), [J(2, 3, 5, 1, 4,
22   9, 10), [J(3, 1, 2, 6, 4, 10, 11), [J(6, 5, 7, 2, 1, 3, 4), [J(6, 7, 5,
23   8, 2, 1, 3), [J(7, 5, 6, 8, 3, 2, 1), [J(7, 8, 9, 6, 1, 11, 2), [J(8, 7,
24   6, 9, 10, 12, 2), [J(8, 9, 7, 10, 11, 1, 12), [J(1, 4, 8, 3, 9, 7, 10),
25   13), [J(9, 8, 10, 12, 13, 2, 11), [J(9, 10, 8, 11, 12, 13, 2), [J(9, 11,
26   12, 13, 1, 10, 3), [J(10, 9, 11, 7, 12, 3, 13), [J(10, 11, 9, 12, 8, 1,
27   4), [J(10, 12, 13, 11, 9, 2, 3), [J(11, 9, 10, 13, 8, 4, 5), [J(11, 10,
28   12, 9, 13, 5, 1), [J(11, 12, 9, 7, 10, 13, 8), [J(12, 10, 13, 7, 11, 9,
29   4), [J(12, 13, 11, 9, 8, 2, 6), [J(13, 11, 10, 7, 9, 8, 1))?
30
31 Yes (55.23s cpu, solution 1, maybe more) ?
32 */

```

```

23 % Amelioration du labeling pour gagner du temps
24 getLast([A],A,[ ]).
25 getLast([AIR],B,[AIL]):- getLast(R,B,L).
26 getLast([A|L],A,L):- getLast(R,B,L).
27 getLast([A,B,C],A,[A,C,B],C).
28 debutfin([AIR],A,L,B):- getLast(R,B,L).
29 debutfin([A,B,C],A,[A,C,B],C).
30 debutfin([A,B|L2]):- getLast(R,B,L),melangListe(L1,L2).
31 melangListe([ ],[ ]).
32 melangListe([A,B,C],[A,C,B]):-!.
33 melangListe(L,[A,B|L2]):- debutfin(L,A,L1,B),melangListe(L1,L2).
34 melangListe(L,[A,B|L2]):- debutfin(L,A,L1,B),melangListe(L1,L2).
35
36 getVarList2(T,L):-
37     dim(T,[NbEquipes,NbConf]),
38     ( for(Indice1,1,NbConf), fromto([ ],In,Out,L), param(T,NbEquipes)
39       do
40         ( for(Indice2,1,NbEquipes), fromto([ ],In2,Out2,L2), param(T,
41           Indice1)
42             do
43               Var is T[Indice2,Indice1],
44               append(In2,[Var],Out2)
45             ),
46             melangListe(L2,L3),
47             append(In,L3,Out)
48           ).
49
50 solve6(T):-
51     getData2(TailleEq ,NBEq,CapaBat ,NbBat ,NbConf),
52     defineVars(T,NBEq,NbConf,NbBat),
53     pasMemeBateaux(T,NBEq,NbConf),
54     pasMemePartenaires(T,NBEq,NbConf),
55     capaBateaux(T,TailleEq ,NBEq,CapaBat ,NbBat ,NbConf),
56     getVarList2(T,L),
57     labeling(L).
58
59 /* Tests
60 [eclipse 15]: solve6(T).
61
62 T = !](I)(I, 2, 3, 4, 5, 6, 7), !](2, 1, 4, 3, 6, 5, 8), !](3, 4, 1, 2, 8, 7,
63   9), !](4, 3, 5, 1, 2, 9, 10), !](5, 6, 2, 7, 1, 3, 4), !](6, 5, 7, 2, 1,
64   4, 3), !](6, 7, 8, 5, 2, 1, 11), !](7, 5, 6, 8, 9, 1, 2), !](8, 9, 7,
65   10, 4, 11, 5), !](8, 10, 9, 6, 11, 3, 2), !](9, 8, 12, 13, 7, 2, 6),
66   !](10, 9, 8, 11, 13, 12, 1), !](12, 13, 11, 9, 10, 8, 1), !](11, 10, 13,
67   3, 8, 9, 4), !](11, 12, 10, 7, 3, 13, 9), !](13, 11, 10, 9, 6, 1, 5),
68   !](13, 8, 11, 12, 4, 10, 3), !](10, 11, 9, 8, 12, 2, 3), !](9, 11, 6, 12,
69   10, 5, 13), !](9, 7, 10, 11, 12, 8, 2), !](7, 8, 9, 10, 3, 4, 1), !](7,
70   6, 5, 9, 11, 2, 8), !](5, 7, 1, 8, 3, 10, 13), !](4, 1, 6, 5, 3, 2, 12),
71   !](3, 2, 4, 1, 7, 11, 12), !](3, 1, 2, 6, 9, 10, 11), !](2, 4, 3, 1, 9,
72   8, 6), !](2, 3, 1, 6, 7, 4, 5), !](1, 3, 2, 5, 4, 7, 6))
73
74 Yes (7.31s cpu, solution 2, maybe more) ?
75 */

```

```

64 T = !](I)(I, 2, 3, 4, 5, 6, 7), !](2, 1, 4, 3, 6, 5, 8), !](3, 4, 1, 2, 8, 7,
65   9), !](4, 3, 5, 1, 2, 9, 10), !](5, 6, 2, 7, 1, 3, 4), !](6, 5, 7, 2, 1,
66   4, 11), !](6, 7, 8, 5, 2, 1, 3), !](7, 5, 6, 8, 9, 1, 2), !](8, 9, 7,
67   10, 4, 11, 5), !](8, 10, 9, 6, 11, 3, 2), !](9, 8, 12, 13, 7, 2, 6),
68   !](10, 9, 8, 11, 13, 12, 1), !](12, 13, 11, 9, 10, 8, 1), !](11, 10, 13,
69   3, 8, 9, 4), !](11, 12, 10, 7, 3, 13, 9), !](13, 11, 10, 9, 6, 1, 5),
70   !](13, 8, 11, 12, 4, 10, 3), !](10, 11, 9, 8, 12, 2, 3), !](9, 11, 6, 12,
71   10, 5, 13), !](9, 7, 10, 11, 12, 8, 2), !](7, 8, 9, 10, 3, 4, 1), !](7,
72   6, 5, 9, 11, 2, 8), !](5, 7, 1, 8, 3, 10, 13), !](4, 1, 6, 5, 3, 2, 12),
73   !](3, 2, 4, 1, 7, 11, 12), !](3, 1, 2, 6, 9, 10, 11), !](2, 4, 3, 1, 9,
74   8, 6), !](2, 3, 1, 6, 7, 4, 5), !](1, 3, 2, 5, 4, 7, 6))
75
76 Yes (6.98s cpu, solution 1, maybe more) ?
77 */

```

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 ,NbConf ,NbBat).

35

36 TailleEq = [](5, 5, 2, 1)

37 NbEq = 4

38 CapaBat = [](7, 6, 5)

39 NbBat = 3

40 NbConf = 3

41 T = [](][(-419{1 .. 3}, -488{1 .. 3}, -557{1 .. 3}),][(-628{1 .. 3}, -697{1 .. 3}), -766{1 .. 3}),][(-837{1 .. 3}, -906{1 .. 3}, -975{1 .. 3}),][(-1046{1 .. 3}, -1115{1 .. 3}, -1184{1 .. 3}))

42 Yes (0.00s cpu)

43 */

44

45 % Q4.3

46 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,
51             Indice1)
52             do
53                 Var is T[Indice2 ,Indice1] ,
54                 append(In2,[Var],Out2)
55             ),
56             append(In ,L2,Out)
57         ).
58
59 /* Tests
60 [eclipse 9]: getData(TailleEq ,NbEq ,CapaBat ,NbBat ,NbConf) ,defineVars(T,NbEq ,
61 NbConf ,NbBat) ,getVarList(T,L).
62
63 TailleEq = [](5, 5, 2, 1)
64 NbEq = 4
65 CapaBat = [](7, 6, 5)
66 NbBat = 3
67 NbConf = 3
68 T = [](][(-484{1 .. 3}, -553{1 .. 3}, -622{1 .. 3}), ][(-693{1 .. 3}, -762{1 .. 3}), -831{1 .. 3}), ][(-902{1 .. 3}, -971{1 .. 3}, -1040{1 .. 3}), ][(-1111{1 .. 3}, -1180{1 .. 3}, -1249{1 .. 3}))
```

69 L = [](484{1 .. 3}, -693{1 .. 3}, -902{1 .. 3}, -1111{1 .. 3}, -553{1 .. 3}, -762{1 .. 3}, -971{1 .. 3}, -1180{1 .. 3}, -622{1 .. 3}, -831{1 .. 3}, -1040{1 .. 3}, -1249{1 .. 3})

70

71 % Q4.4

72

73 solve1(T) :-

74 getData(_TailleEquipes ,NbEquipes ,_CapaBateaux ,NbBateaux ,NbConf) ,

75 defineVars(T,NbEquipes ,NbConf ,NbBateaux) ,

76 getVarList(T,L) ,

77 labeling(L).

78

79 /* Tests

80 [eclipse 10]: solve1(T).

81

82 T = [](][(1, 1, 1),][(1, 1, 1),][(1, 1, 1),][(1, 1, 1))

83 Yes (0.00s cpu, solution 1, maybe more) ? ;

84

85 T = [](][(1, 1, 1),][(1, 1, 1),][(1, 1, 1),][(1, 1, 2))

86 Yes (0.00s cpu, solution 2, maybe more) ? ;

87

88 T = [](][(1, 1, 1),][(1, 1, 1),][(1, 1, 1),][(1, 1, 3))

89 Yes (0.00s cpu, solution 3, maybe more) ?

90 */

91

92 % Q4.5

93 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         )
99         do
100           Bat is T[Indice1,Indice2],
101             append(In,[Bat],Out)
102           ),
103             alldifferent(L)
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 /* Tests
112 [eclipse 11]: solve2(T).
113
114 T = [[[[1, 2, 3), [[1, 2, 3), [[1, 2, 3), [[1, 2, 3))
115 Yes (0.00s cpu, solution 1, maybe more) ? ;
116
117 T = [[[[1, 2, 3), [[1, 2, 3), [[1, 2, 3), [[1, 3, 2))
118 Yes (0.00s cpu, solution 2, maybe more) ? ;
119
120 T = [[[[1, 2, 3), [[1, 2, 3), [[1, 3, 2), [[1, 2, 3))]
121 Yes (0.00s cpu, solution 3, maybe more) ? ;
122
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                   Tot #=< 1
141                 )
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, 2, 3), [[1, 3, 2), [[2, 3, 1), [[2, 1, 3))
160 Yes (0.00s cpu, solution 2, maybe more) ? ;
161
162 T = [[[1, 2, 3), [[1, 2, 3), [[2, 1, 3), [[2, 3, 1))
163 Yes (0.00s cpu, solution 3, maybe more) ? ;
164 */
165
166 % Q4.7
167 capaBateaux(T,TailleEquipes,NbEquipes,CapaBateaux,NbBateaux,NbConf):-
168   dim(T,[NbEquipes,NbConf]),
169   ( for(Bateau,1,NbBateaux), param(T,NbEquipes,NbConf,CapaBateaux,
170     TailleEquipes)
171     do
172       ( for(Conf,1,NbConf), param(T,NbEquipes,Bateau,CapaBateaux,
173         TailleEquipes)
174         do
175           ( for(Equipe,1,NbEquipes), param(T,Bateau,Conf,
176             TailleEquipes), fromto(0,In,Out,Tot)
177               do
178                 Bateau1 is T[Equipe,Conf],
179                 #=(Bateau,Bateau1,Cond),
180                   Inc #= TailleEquipes[Equipe] * Cond,
181                     Out #= In + Inc
182                   ),
183                   Capacite is CapaBateaux[Bateau],
184                     Total #<= Capacite
185               )
186             .
187
188 solve4(T) :-
189   getData(TailleEquipes,NbEquipes,CapaBateaux,NbBateaux,NbConf),
190   defineVars(T,NbEquipes,NbConf,NbBateaux),
191   pasMemeBateaux(T,NbEquipes,NbConf),
192   pasMemePartenaires(T,NbEquipes,NbConf),
193   capaBateaux(T,TailleEquipes,NbEquipes,CapaBateaux,NbBateaux,NbConf),
194   getVarList(T,L),
195   labeling(L).
196
197 /* Tests

```

```

195 [eclipse 13]: solve4(T).
196
197 T = [](1)(1, 2, 3), [](2, 3, 1), [](3, 1, 2), [](3, 2, 1))
198 Yes (0.01s cpu, solution 1, maybe more) ? ;
199
200 T = [](1)(1, 3, 2), [](2, 1, 3), [](3, 2, 1), [](3, 1, 2))
201 Yes (0.01s cpu, solution 2, maybe more) ? ;
202
203 T = [](1)(1, 2, 3), [](3, 1, 2), [](2, 3, 1), [](1, 3, 2))
204 Yes (0.01s cpu, solution 3, maybe more) ? ;
205 */
206
207 % Q4.8
208
209 getData2(TailleEquipes ,NbEquipes ,CapaBateaux ,NbBateaux ,NbConf):-
210     TailleEquipes =
211         [](7,6,5,5,4,4,4,4,4,4,4,3,3,2,2,2,2,2,2,2,2,2,2,2,2,2),
212     NbEquipes = 29,
213     CapaBateaux = [](10,10,9,8,8,8,8,8,7,6,4,4),
214     NbBateaux = 13,
215     NbConf = 7.
216
217 solve5(T):-
218     getData2(TailleEq ,NbEq ,CapaBat ,NbBat ,NbConf) ,
219     defineVars(T,NbEq,NbConf,NbBat),
220     pasMemeBateaux(T,NbEq,NbConf),
221     pasMemePartenaires(T,NbEq,NbConf),
222     capaBateaux(T,TailleEq ,NbEq ,CapaBat ,NbBat ,NbConf) ,
223     getVarList(T,L),
224     labeling(L).
225
226 /* Tests
227 [eclipse 14]: solve5(T).
228 T = [](1)(1, 2, 3, 4, 5, 6, 7), [](2, 1, 4, 3, 6, 5, 8), [](3, 4, 1, 2, 7, 8,
229      5), [](4, 3, 1, 5, 2, 7, 6), [](5, 6, 2, 1, 3, 4, 9), [](2, 3, 5, 1, 4,
230      9, 10), [](3, 1, 2, 6, 4, 10, 11), [](6, 5, 7, 2, 1, 3, 4), [](6, 7, 5,
231      8, 2, 1, 3), [](7, 5, 6, 8, 3, 2, 1), [](7, 8, 9, 6, 1, 11, 2), [](8, 7,
232      6, 9, 10, 12, 2), [](8, 9, 7, 10, 11, 1, 12), [](1, 4, 8, 3, 9, 7, 10),
233      [](4, 2, 8, 10, 7, 3, 9), [](5, 8, 3, 11, 6, 9, 1), [](9, 6, 4, 5, 8, 11,
234      13), [](9, 8, 10, 12, 13, 2, 11), [](9, 10, 8, 11, 12, 13, 2), [](9, 11,
235      12, 13, 1, 10, 3), [](10, 9, 11, 7, 12, 3, 13), [](10, 11, 9, 12, 8, 1,
236      4), [](10, 12, 13, 11, 9, 2, 3), [](11, 9, 10, 13, 8, 4, 5), [](11, 10,
237      12, 9, 13, 5, 1), [](11, 12, 9, 7, 10, 13, 8), [](12, 10, 13, 7, 11, 9,
238      4), [](12, 13, 11, 9, 8, 2, 6), [](13, 11, 10, 7, 9, 8, 1))
```

229 Yes (55.23s cpu, solution 1, maybe more) ? ;

230 */

231 % Amelioration du labeling pour gagner du temps

232 getLast([A],A,[]).

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

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

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

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

237 debutfin([AIR],A,L,B) :- getLast(R,B,L).

238 debutfin([A,B,C],A,[A,C,B],C).

239

240 melangListe([],[]).

241 melangListe([A,B,C],[A,C,B]) :- !.

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

243

244 getVarList2(T,L):-

245 dim(T,[NbEquipes ,NbConf]),

246 (for(Indice1,1,NbConf), fromto([],In ,Out ,L), param(T,NbEquipes)

247 do

248 (for(Indice2,1,NbEquipes), fromto([],In2 ,Out2 ,L2), param(T,

249 Indice1)

250 do

251 Var is T[Indice2 ,Indice1],

252 append(In2 ,[Var],Out2)

253),

254 melangListe(L2 ,L3),

255 append(In ,L3 ,Out)

256).

257 solve6(T):-

258 getData2(TailleEq ,NbEq ,CapaBat ,NbBat ,NbConf) ,

259 defineVars(T,NbEq,NbConf,NbBat),

260 pasMemeBateaux(T,NbEq,NbConf),

261 pasMemePartenaires(T,NbEq,NbConf),

262 capaBateaux(T,TailleEq ,NbEq ,CapaBat ,NbBat ,NbConf) ,

263 getVarList2(T,L) ,

264 labeling(L).

265

266 /* Tests

267 [eclipse 15]: solve6(T).

268

269 T = [](1)(1, 2, 3, 4, 5, 6, 7), [](2, 1, 4, 3, 6, 5, 8), [](3, 4, 1, 2, 8, 7,
270 9), [](4, 3, 5, 1, 2, 9, 10), [](5, 6, 2, 7, 1, 3, 4), [](6, 5, 7, 2, 1,
271 4, 3), [](6, 7, 8, 5, 2, 1, 11), [](7, 5, 6, 8, 9, 1, 2), [](8, 9, 7,
272 10, 4, 11, 5), [](8, 10, 9, 6, 11, 3, 2), [](9, 8, 12, 13, 7, 2, 6),
273 [](10, 9, 8, 11, 13, 12, 1), [](12, 13, 11, 9, 10, 8, 1), [](11, 10,
274 13, 11, 10, 9, 6, 1, 5), [](11, 12, 10, 7, 3, 13, 9), [](13, 11, 10, 9,
275 6, 1, 5), [](13, 8, 11, 12, 4, 10, 3), [](10, 11, 9, 8, 12, 2, 3), [](9,
276 11, 6, 12, 10, 5, 13), [](9, 7, 10, 11, 12, 8, 2), [](8, 9, 10, 3, 4,
277 1), [](7, 6, 5, 9, 11, 2, 8), [](5, 7, 1, 8, 3, 10, 13), [](4, 1, 6, 5,
278 3, 2, 12), [](3, 2, 4, 1, 7, 11, 12), [](3, 1, 2, 6, 9, 10, 11), [](2,
279 4, 3, 1, 9, 8, 6), [](2, 3, 1, 6, 7, 4, 5), [](1, 3, 2, 5, 4, 7, 6))

280 Yes (6.98s cpu, solution 1, maybe more) ? ;

281

282 T = [](1)(1, 2, 3, 4, 5, 6, 7), [](2, 1, 4, 3, 6, 5, 8), [](3, 4, 1, 2, 8, 7,
283 9), [](4, 3, 5, 1, 2, 9, 10), [](5, 6, 2, 7, 1, 3, 4), [](6, 5, 7, 2, 1,
284 4, 11), [](6, 7, 8, 5, 2, 1, 3), [](7, 5, 6, 8, 9, 1, 2), [](8, 9, 7,
285 10, 4, 11, 5), [](8, 10, 9, 6, 11, 3, 2), [](9, 8, 12, 13, 7, 2, 6),
286 [](10, 9, 8, 11, 13, 12, 1), [](12, 13, 11, 9, 10, 8, 1), [](11, 10,
287 13, 8, 9, 4), [](11, 12, 10, 7, 3, 13, 9), [](13, 11, 10, 9, 6, 1, 5),
288 [](13, 8, 11, 12, 4, 10, 3), [](10, 11, 9, 8, 12, 2, 3), [](9, 11, 6,
289 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))

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

274 */

Rapport Travaux Pratiques :

Programmation par Contraintes

- TP 4 :

Contraintes Logiques

Nicolas Desfeux
Aurélien Texier

30 mars 2011

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

Question 4.1 Nous définissons ici un prédictat `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 */

```

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

Listing 2 – "defineVars"

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

```

2         dim(T, [ NbEquipes ,NbConf]) ,
3         ( for(Ind1 ,1 ,NbEquipes ),param(T,NbBateaux ,NbConf)
4           do
5             ( for(Ind2 ,1 ,NbConf),param(T,Ind1 ,NbBateaux )
6               do
7                 T[ Ind1 ,Ind2 ] #:: 1..NbBateaux
8               )
9             ).
10
11 /* Tests
12 [eclipse 8]: getData(TailleEq ,NbEq ,CapaBat ,NbBat ,NbConf) ,defineVars(T,NbEq ,
13 NbConf ,NbBat).
14
15 TailleEq = [](5, 5, 2, 1)
16 NbEq = 4
17 CapaBat = [](7, 6, 5)
18 NbBat = 3
19 NbConf = 3
20
21 T = [](1)(-419{1 .. 3}, -488{1 .. 3}, -557{1 .. 3}), [](-628{1 .. 3}), -697{1
22 .. 3}, -766{1 .. 3}), [](-837{1 .. 3}), -906{1 .. 3}, -975{1 .. 3}), [](
23 -1046{1 .. 3}, -1115{1 .. 3}, -1184{1 .. 3}))
24
25 Yes (0.00s cpu)
26 */

```

Question 4.3 Nous définissons ici un prédictat `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

```

```

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

```

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

Listing 4 – "solve1"

```

1 solve1(T) :-  

2     getData(_TailleEquipes ,NbEquipes ,_CapaBateaux ,NbBateaux ,NbConf),  

3     defineVars(T,NbEquipes ,NbConf ,NbBateaux ),  

4     getVarList(T,L),  

5     labeling(L).  

6  

7 /* Tests  

8 [eclipse 10]: solve1(T).  

9  

10 T = [J(I)(1, 1, 1), IJ(1, 1, 1), II(1, 1, 1), IJ(1, 1, 1))  

11 Yes (0.00s cpu, solution 1, maybe more) ? ;  

12  

13 T = [J(I)(1, 1, 1), II(1, 1, 1), II(1, 1, 1), IJ(1, 1, 2))  

14 Yes (0.00s cpu, solution 2, maybe more) ? ;  

15  

16 T = [J(I)(1, 1, 1), II(1, 1, 1), II(1, 1, 1), II(1, 1, 3))  

17 Yes (0.00s cpu, solution 3, maybe more) ?  

18 */

```

Question 4.5 Nous définissons ici un prédictat *pasMemeBateaux(+T,+NbEquipes,+NbConf)* qui impose qu'une même équipe ne retourne pas deux fois sur le même bateau. On modifie ensuite le prédictat *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
        )
        do
        Bat is T[Indice1 ,Indice2 ],
        append(In ,[ Bat ],Out)
        ),
        allDifferent(L)
    ).
12

```

```

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 = [J(I)(1, 2, 3), IJ(1, 2, 3), II(1, 2, 3), IJ(1, 2, 3))  

24 Yes (0.00s cpu, solution 1, maybe more) ? ;  

25  

26 T = [J(I)(1, 2, 3), II(1, 2, 3), II(1, 2, 3), IJ(1, 3, 2))  

27 Yes (0.00s cpu, solution 2, maybe more) ? ;  

28  

29 T = [J(I)(1, 2, 3), II(1, 2, 3), II(1, 3, 2), II(1, 2, 3))  

30 Yes (0.00s cpu, solution 3, maybe more) ?  

31 */

```

Question 4.6 Nous définissons ici un prédictat *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 prédictat *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             ( for(Conf ,1,NbConf) ,param(T,Equipe1 ,Equipe2) ,fromto
        (0,In ,Out ,Tot)
        do
        Bateau1 is T[Equipe1 ,Conf ],
        Bateau2 is T[Equipe2 ,Conf ],
        #=(Bateau1 ,Bateau2 ,Ans) ,
        Out #= In + Ans
        ),
        Tot #=< 1
    17         )
    18     ).
19  

20 solve3(T) :-  

21     getData(_TailleEquipes ,NbEquipes ,_CapaBateaux ,NbBateaux ,NbConf),  

22     defineVars(T,NbEquipes ,NbConf ,NbBateaux ),  

23     pasMemeBateaux(T,NbEquipes ,NbConf),
24     pasMemePartenaires(T,NbEquipes ,NbConf),
25     getVarList(T,L),
26     labeling(L).

```

```

27 /* Tests
28 [eclipse 12]: solve3(T).
29
30 T = [J([1, 2, 3), [J(1, 3, 2), [J(2, 1, 3), [J(2, 3, 1))
31 Yes (0.00s cpu, solution 1, maybe more) ? ;
32
33 T = [J([1, 2, 3), [J(1, 3, 2), [J(2, 3, 1), [J(2, 1, 3))
34 Yes (0.00s cpu, solution 2, maybe more) ? ;
35
36 T = [J([1, 3, 2), [J(1, 2, 3), [J(2, 1, 3), [J(2, 3, 1))
37 Yes (0.00s cpu, solution 3, maybe more) ? ;
38
39 */

```

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

Listing 7 – "capaBateaux"

```

1 capaBateaux(T, TailleEquipes ,NbEquipes ,CapaBateaux ,NbBateaux ,NbConf):-
2     dim(T,[ NbEquipes ,NbConf ]) ,
3     ( for(Bateau,1,NbBateaux) , param(T,NbEquipes ,NbConf ,CapaBateaux ,
4         TailleEquipes)
5     do
6         ( for(Conf,1,NbConf) , param(T,NbEquipes ,Bateau ,CapaBateaux ,
7             TailleEquipes)
8             do
9                 ( for(Equipe,1,NbEquipes) ,param(T,Bateau ,Conf ,
10                     TailleEquipes) ,fromto(0,In ,Out ,Total)
11                     do
12                         BateauI is T[Equipe ,Conf] ,
13                         #=(Bateau ,BateauI ,Cond) ,
14                         Inc #= TailleEquipes [Equipe] * Cond ,
15                         Out #= In + Inc
16                     ),
17                     Capacite is CapaBateaux [Bateau] ,
18                     Total #=< Capacite
19                 )
20             )
21 solve4(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     capaBateaux(T,TailleEquipes ,NbEquipes ,CapaBateaux ,NbBateaux ,NbConf) ,
27     getVarList(T,L) ,
28     labeling(L) .
29 /* Tests
30 [eclipse 13]: solve4(T).
31
32
33
34
35
36
37
38
39 */

```

```

31 T = [J([1, 2, 3), [J(2, 3, 1), [J(3, 1, 2), [J(3, 2, 1))
32 Yes (0.01s cpu, solution 1, maybe more) ? ;
33
34 T = [J([1, 3, 2), [J(2, 1, 3), [J(3, 2, 1), [J(3, 1, 2))
35 Yes (0.01s cpu, solution 2, maybe more) ? ;
36
37 T = [J([1, 2, 3), [J(3, 1, 2), [J(2, 3, 1), [J(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 voiliers et de 29 équipes qui doivent effectuer la régate comportant 7 confrontations.

Le temps d'exécution étant relativement long, il nous a été proposé d'améliorer le labeling. Pour cela, nous avons mélanger la liste des 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 labeling"

```

1 getData2(TailleEquipes ,NbEquipes ,CapaBateaux ,NbBateaux ,NbConf):-
2     TailleEquipes =
3         [1|7,6,5,5,4,4,4,4,4,4,4,4,4,3,3,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2],
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
8 solve5(T):-  

9     getData2(TailleEq ,NbEq ,CapaBat ,NbBat ,NbConf) ,
10    defineVars(T,NbEq ,NbConf ,NbBat),
11    pasMemeBateaux(T,NbEq ,NbConf) ,
12    pasMemePartenaires(T,NbEq ,NbConf) ,
13    capaBateaux(T,TailleEq ,NbEq ,CapaBat ,NbBat ,NbConf) ,
14    getVarList(T,L) ,
15    labeling(L).
16
17 /* Tests
18 [eclipse 14]: solve5(T).
19
20 T = [J([1, 2, 3, 4, 5, 6, 7), [J(2, 1, 4, 3, 6, 5, 8), [J(3, 4, 1, 2, 7, 8,
21 5), [J(4, 3, 1, 5, 2, 7, 6), [J(5, 6, 2, 1, 3, 4, 9), [J(2, 3, 5, 1, 4,
22 9, 10), [J(3, 1, 2, 6, 4, 10, 11), [J(6, 5, 7, 2, 1, 3, 4), [J(6, 7, 5,
23 8, 2, 1, 3), [J(7, 5, 6, 8, 3, 2, 1), [J(7, 8, 9, 6, 1, 11, 2), [J(8, 7,
24 6, 9, 10, 12, 2), [J(8, 9, 7, 10, 11, 1, 12), [J(1, 4, 8, 3, 9, 7, 10),
25 13), [J(4, 2, 8, 10, 7, 3, 9), [J(5, 8, 3, 11, 6, 9, 1), [J(9, 6, 4, 5, 8, 11,
26 13), [J(9, 8, 10, 12, 13, 2, 11), [J(9, 10, 8, 11, 12, 13, 2), [J(9, 11,
27 12, 13, 1, 10, 3), [J(10, 9, 11, 7, 12, 3, 13), [J(10, 11, 9, 12, 8, 1,
28 4), [J(10, 12, 13, 11, 9, 2, 3), [J(11, 9, 10, 13, 8, 4, 5), [J(11, 10,
29 12, 9, 13, 5, 1), [J(11, 12, 9, 7, 10, 13, 8), [J(12, 10, 13, 7, 11, 9,
4), [J(12, 13, 11, 9, 8, 2, 6), [J(13, 11, 10, 7, 9, 8, 1))]
30 Yes (55.23s cpu, solution 1, maybe more) ?
31 */

```

```

23 % Amelioration du labeling pour gagner du temps
24 getLast([A],A,[ ]).
25 getLast([AIR],B,[AIL]):- getLast(R,B,L).
26 getLast([AIR],A,L,B) :- getLast(R,B,L).
27 debutfin([AIR],A,L,B) :- getLast(R,B,L).
28 debutfin([A,B,C],A,[A,C,B],C).
29 melangListe([ ],[ ]).
30 melangListe([A,B,C],[A,C,B]):-!.
31 melangListe(L,[A,B|L2]):- debutfin(L,A,L1,B),melangListe(L1,L2).
32
33 getVarList2(T,L):-
34     dim(T,[NbEquipes,NbConf]),
35     ( for(Indice1,1,NbConf), fromto([ ],In,Out,L), param(T,NbEquipes)
36       do
37         ( for(Indice2,1,NbEquipes), fromto([ ],In2,Out2,L2), param(T,
38           Indice1)
39             do
40               Var is T[Indice2,Indice1],
41               append(In2,[Var],Out2)
42             ),
43             melangListe(L2,L3),
44             append(In,L3,Out)
45           ).
46
47 solve6(T):-
48     getData2(TailleEq ,NBEq,CapaBat ,NbBat ,NbConf),
49     defineVars(T,NBEq,NbConf,NbBat),
50     pasMemeBateaux(T,NBEq,NbConf),
51     pasMemePartenaires(T,NBEq,NbConf),
52     capaBateaux(T,TailleEq ,NBEq,CapaBat ,NbBat ,NbConf),
53     getVarList2(T,L),
54     labeling(L).
55
56 /* Tests
57 [eclipse 15]: solve6(T).
58
59
60
61 T = !](I)(I, 2, 3, 4, 5, 6, 7), !](2, 1, 4, 3, 6, 5, 8), !](3, 4, 1, 2, 8, 7,
62   9), !](4, 3, 5, 1, 2, 9, 10), !](5, 6, 2, 7, 1, 3, 4), !](6, 5, 7, 2, 1,
63   4, 3), !](6, 7, 8, 5, 2, 1, 11), !](7, 5, 6, 8, 9, 1, 2), !](8, 9, 7,
64   10, 4, 11, 5), !](8, 10, 9, 6, 11, 3, 2), !](9, 8, 12, 13, 7, 2, 6),
65   !](10, 9, 8, 11, 13, 12, 1), !](12, 13, 11, 9, 10, 8, 1), !](11, 10, 13,
66   3, 8, 9, 4), !](11, 12, 10, 7, 3, 13, 9), !](13, 11, 10, 9, 6, 1, 5),
67   !](13, 8, 11, 12, 4, 10, 3), !](10, 11, 9, 8, 12, 2, 3), !](9, 11, 6, 12,
68   10, 5, 13), !](9, 7, 10, 11, 12, 8, 2), !](7, 8, 9, 10, 3, 4, 1), !](7,
69   6, 5, 9, 11, 2, 8), !](5, 7, 1, 8, 3, 10, 13), !](4, 1, 6, 5, 3, 2, 12),
70   !](3, 2, 4, 1, 7, 11, 12), !](3, 1, 2, 6, 9, 10, 11), !](2, 4, 3, 1, 9,
71   8, 6), !](2, 3, 1, 6, 7, 4, 5), !](1, 3, 2, 5, 4, 7, 6))
72 Yes (6.98s cpu, solution 1, maybe more) ?
73

```

```

64 T = !](I)(I, 2, 3, 4, 5, 6, 7), !](2, 1, 4, 3, 6, 5, 8), !](3, 4, 1, 2, 8, 7,
65   9), !](4, 3, 5, 1, 2, 9, 10), !](5, 6, 2, 7, 1, 3, 4), !](6, 5, 7, 2, 1,
66   4, 11), !](6, 7, 8, 5, 2, 1, 3), !](7, 5, 6, 8, 9, 1, 2), !](8, 9, 7,
67   10, 4, 11, 5), !](8, 10, 9, 6, 11, 3, 2), !](9, 8, 12, 13, 7, 2, 6),
68   !](10, 9, 8, 11, 13, 12, 1), !](12, 13, 11, 9, 10, 8, 1), !](11, 10, 13,
69   3, 8, 9, 4), !](11, 12, 10, 7, 3, 13, 9), !](13, 11, 10, 9, 6, 1, 5),
70   !](13, 8, 11, 12, 4, 10, 3), !](10, 11, 9, 8, 12, 2, 3), !](9, 11, 6, 12,
71   10, 5, 13), !](9, 7, 10, 11, 12, 8, 2), !](7, 8, 9, 10, 3, 4, 1), !](7,
72   6, 5, 9, 11, 2, 8), !](5, 7, 1, 8, 3, 10, 13), !](4, 1, 6, 5, 3, 2, 12),
73   !](3, 2, 4, 1, 7, 11, 12), !](3, 1, 2, 6, 9, 10, 11), !](2, 4, 3, 1, 9,
74   8, 6), !](2, 3, 1, 6, 7, 4, 5), !](1, 3, 2, 5, 4, 7, 6))
75 Yes (7.31s cpu, solution 2, maybe more) ?
76 */

```

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 ,NbConf ,NbBat).

35

36 TailleEq = [](5, 5, 2, 1)

37 NbEq = 4

38 CapaBat = [](7, 6, 5)

39 NbBat = 3

40 NbConf = 3

41 T = [](][(-419{1 .. 3}, -488{1 .. 3}, -557{1 .. 3}),][(-628{1 .. 3}, -697{1 .. 3}), -766{1 .. 3}),][(-837{1 .. 3}, -906{1 .. 3}, -975{1 .. 3}),][(-1046{1 .. 3}, -1115{1 .. 3}, -1184{1 .. 3}))

42 Yes (0.00s cpu)

43 */

44

45 % Q4.3

46 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,
51             Indice1)
52             do
53                 Var is T[Indice2 ,Indice1] ,
54                 append(In2,[Var],Out2)
55             ),
56             append(In ,L2,Out)
57         ).
```

58 /* Tests

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

60

61 TailleEq = [](5, 5, 2, 1)

62 NbEq = 4

63 CapaBat = [](7, 6, 5)

64 NbBat = 3

65 NbConf = 3

66 T = [](][(-484{1 .. 3}, -553{1 .. 3}, -622{1 .. 3}),][(-693{1 .. 3}, -762{1 .. 3}), -831{1 .. 3}),][(-902{1 .. 3}, -971{1 .. 3}, -1040{1 .. 3}),][(-1111{1 .. 3}, -1180{1 .. 3}, -1249{1 .. 3}))

67 L = [](484{1 .. 3}, -693{1 .. 3}, -902{1 .. 3}, -1111{1 .. 3}, -553{1 .. 3}, -762{1 .. 3}, -971{1 .. 3}, -1180{1 .. 3}, -622{1 .. 3}, -831{1 .. 3}, -1040{1 .. 3}, -1249{1 .. 3})

68 Yes (0.00s cpu)

69 */

70

71 % Q4.4

72

73 solve1(T) :-

74 getData(_TailleEquipes ,NbEquipes ,_CapaBateaux ,NbBateaux ,NbConf) ,

75 defineVars(T,NbEquipes ,NbConf ,NbBateaux) ,

76 getVarList(T,L) ,

77 labeling(L).

78

79 /* Tests

80 [eclipse 10]: solve1(T).

81

82 T = [](][(1, 1, 1),][(1, 1, 1),][(1, 1, 1),][(1, 1, 1))

83 Yes (0.00s cpu, solution 1, maybe more) ? ;

84

85 T = [](][(1, 1, 1),][(1, 1, 1),][(1, 1, 1),][(1, 1, 2))

86 Yes (0.00s cpu, solution 2, maybe more) ? ;

87

88 T = [](][(1, 1, 1),][(1, 1, 1),][(1, 1, 1),][(1, 1, 3))

89 Yes (0.00s cpu, solution 3, maybe more) ?

90 */

91

92 % Q4.5

93 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         )
99         do
100           Bat is T[Indice1,Indice2],
101             append(In,[Bat],Out)
102           ),
103             alldifferent(L)
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 /* Tests
112 [eclipse 11]: solve2(T).
113
114 T = [[[[1, 2, 3), [[1, 2, 3), [[1, 2, 3), [[1, 2, 3))
115 Yes (0.00s cpu, solution 1, maybe more) ? ;
116
117 T = [[[[1, 2, 3), [[1, 2, 3), [[1, 2, 3), [[1, 3, 2))
118 Yes (0.00s cpu, solution 2, maybe more) ? ;
119
120 T = [[[[1, 2, 3), [[1, 2, 3), [[1, 3, 2), [[1, 2, 3))]
121 Yes (0.00s cpu, solution 3, maybe more) ? ;
122
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                   Tot #=< 1
141                 )
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, 2, 3), [[1, 3, 2), [[2, 3, 1), [[2, 1, 3))
160 Yes (0.00s cpu, solution 2, maybe more) ? ;
161
162 T = [[[1, 2, 3), [[1, 2, 3), [[2, 1, 3), [[2, 3, 1))
163 Yes (0.00s cpu, solution 3, maybe more) ? ;
164 */
165
166 % Q4.7
167 capaBateaux(T,TailleEquipes,NbEquipes,CapaBateaux,NbBateaux,NbConf):-
168   dim(T,[NbEquipes,NbConf]),
169   ( for(Bateau,1,NbBateaux), param(T,NbEquipes,NbConf,CapaBateaux,
170     TailleEquipes)
171     do
172       ( for(Conf,1,NbConf), param(T,NbEquipes,Bateau,CapaBateaux,
173         TailleEquipes)
174         do
175           ( for(Equipe,1,NbEquipes), param(T,Bateau,Conf,
176             TailleEquipes), fromto(0,In,Out,Tot)
177               do
178                 Bateau1 is T[Equipe,Conf],
179                 #=(Bateau1,Bateau,Cond),
180                   Inc #= TailleEquipes[Equipe] * Cond,
181                     Out #= In + Inc
182                   ),
183                   Capacite is CapaBateaux[Bateau],
184                     Total #=< Capacite
185               )
186             .
187
188 solve4(T) :-
189   getData(TailleEquipes,NbEquipes,CapaBateaux,NbBateaux,NbConf),
190   defineVars(T,NbEquipes,NbConf,NbBateaux),
191   pasMemeBateaux(T,NbEquipes,NbConf),
192   pasMemePartenaires(T,NbEquipes,NbConf),
193   capaBateaux(T,TailleEquipes,NbEquipes,CapaBateaux,NbBateaux,NbConf),
194   getVarList(T,L),
195   labeling(L).
196
197 /* Tests

```

```

195 [eclipse 13]: solve4(T).
196
197 T = [](1)(1, 2, 3), [](2, 3, 1), [](3, 1, 2), [](3, 2, 1))
198 Yes (0.01s cpu, solution 1, maybe more) ? ;
199
200 T = [](1)(1, 3, 2), [](2, 1, 3), [](3, 2, 1), [](3, 1, 2))
201 Yes (0.01s cpu, solution 2, maybe more) ? ;
202
203 T = [](1)(1, 2, 3), [](3, 1, 2), [](2, 3, 1), [](1, 3, 2))
204 Yes (0.01s cpu, solution 3, maybe more) ? ;
205 */
206
207 % Q4.8
208
209 getData2(TailleEquipes ,NbEquipes ,CapaBateaux ,NbBateaux ,NbConf):-
210     TailleEquipes =
211         [](7,6,5,5,4,4,4,4,4,4,4,3,3,2,2,2,2,2,2,2,2,2,2,2,2,2),
212     NbEquipes = 29,
213     CapaBateaux = [](10,10,9,8,8,8,8,8,7,6,4,4),
214     NbBateaux = 13,
215     NbConf = 7.
216
217 solve5(T):-
218     getData2(TailleEq ,NbEq ,CapaBat ,NbBat ,NbConf) ,
219     defineVars(T,NbEq,NbConf,NbBat),
220     pasMemeBateaux(T,NbEq,NbConf),
221     pasMemePartenaires(T,NbEq,NbConf),
222     capaBateaux(T,TailleEq ,NbEq ,CapaBat ,NbBat ,NbConf) ,
223     getVarList(T,L),
224     labeling(L).
225
226 /* Tests
227 [eclipse 14]: solve5(T).
228 T = [](1)(1, 2, 3, 4, 5, 6, 7), [](2, 1, 4, 3, 6, 5, 8), [](3, 4, 1, 2, 7, 8,
229      5), [](4, 3, 1, 5, 2, 7, 6), [](5, 6, 2, 1, 3, 4, 9), [](2, 3, 5, 1, 4,
230      9, 10), [](3, 1, 2, 6, 4, 10, 11), [](6, 5, 7, 2, 1, 3, 4), [](6, 7, 5,
231      8, 2, 1, 3), [](7, 5, 6, 8, 3, 2, 1), [](7, 8, 9, 6, 1, 11, 2), [](8, 7,
232      6, 9, 10, 12, 2), [](8, 9, 7, 10, 11, 1, 12), [](1, 4, 8, 3, 9, 7, 10),
233      [](4, 2, 8, 10, 7, 3, 9), [](5, 8, 3, 11, 6, 9, 1), [](9, 6, 4, 5, 8, 11,
234      13), [](9, 8, 10, 12, 13, 2, 11), [](9, 10, 8, 11, 12, 13, 2), [](9, 11,
235      12, 13, 1, 10, 3), [](10, 9, 11, 7, 12, 3, 13), [](10, 11, 9, 12, 8, 1,
236      4), [](10, 12, 13, 11, 9, 2, 3), [](11, 9, 10, 13, 8, 4, 5), [](11, 10,
237      12, 9, 13, 5, 1), [](11, 12, 9, 7, 10, 13, 8), [](12, 10, 13, 7, 11, 9,
238      4), [](12, 13, 11, 9, 8, 2, 6), [](13, 11, 10, 7, 9, 8, 1))).
239 Yes (55.23s cpu, solution 1, maybe more) ? ;
240
241 melangListe([],[]).
242 melangListe([A,B,C],[A,C,B]) :- !.
243 melangListe(L,[A,B|L2]) :- debutfin(L,A,L1,B),melangListe(L1,L2).
244
245 getVarList2(T,L):-!
246     dim(T,[NbEquipes ,NbConf]),
247     ( for(Indice1 ,1 ,NbConf),fromto([],In ,Out ,L),param(T,NbEquipes )
248     do
249         ( for(Indice2 ,1 ,NbEquipes ),fromto([],In2 ,Out2 ,L2),param(T,
250             Indice1)
251             do
252                 Var is T[Indice2 ,Indice1 ],
253                 append(In2 ,[Var],Out2 )
254             ),
255             melangListe(L2 ,L3),
256             append(In ,L3 ,Out )
257         ).
258
259 solve6(T):-!
260     getData2(TailleEq ,NbEq ,CapaBat ,NbBat ,NbConf) ,
261     defineVars(T,NbEq,NbConf,NbBat),
262     pasMemeBateaux(T,NbEq,NbConf),
263     pasMemePartenaires(T,NbEq,NbConf),
264     capaBateaux(T,TailleEq ,NbEq ,CapaBat ,NbBat ,NbConf) ,
265     getVarList2(T,L),
266     labeling(L).
267
268 /* Tests
269 [eclipse 15]: solve6(T).
270 T = [](1)(1, 2, 3, 4, 5, 6, 7), [](2, 1, 4, 3, 6, 5, 8), [](3, 4, 1, 2, 8, 7,
271      9), [](4, 3, 5, 1, 2, 9, 10), [](5, 6, 2, 7, 1, 3, 4), [](6, 5, 7, 2, 1,
272      4, 3), [](6, 7, 8, 5, 2, 1, 11), [](7, 5, 6, 8, 9, 1, 2), [](8, 9, 7,
273      10, 4, 11, 5), [](8, 10, 9, 6, 11, 3, 2), [](9, 8, 12, 13, 7, 2, 6),
274      [](10, 9, 8, 11, 13, 12, 1), [](12, 13, 11, 9, 10, 8, 1), [](11, 10,
275      13, 3, 8, 9, 4), [](11, 12, 10, 7, 3, 13, 9), [](13, 11, 10, 9, 6, 1, 5),
276      [](13, 8, 11, 12, 4, 10, 3), [](10, 11, 9, 8, 12, 2, 3), [](9, 11, 6, 12,
277      10, 5, 13), [](9, 7, 10, 11, 12, 8, 2), [](7, 8, 9, 10, 3, 4, 1), [](7,
278      6, 5, 9, 11, 2, 8), [](5, 7, 1, 8, 3, 10, 13), [](4, 1, 6, 5, 3, 2, 12),
279      [](3, 2, 4, 1, 7, 11, 12), [](3, 1, 2, 6, 9, 10, 11), [](2, 4, 3, 1, 9,
280      8, 6), [](2, 3, 1, 6, 7, 4, 5), [](1, 3, 2, 5, 4, 7, 6)).
281 Yes (6.98s cpu, solution 1, maybe more) ? ;
282
283 T = [](1)(1, 2, 3, 4, 5, 6, 7), [](2, 1, 4, 3, 6, 5, 8), [](3, 4, 1, 2, 8, 7,
284      9), [](4, 3, 5, 1, 2, 9, 10), [](5, 6, 2, 7, 1, 3, 4), [](6, 5, 7, 2, 1,
285      4, 11), [](6, 7, 8, 5, 2, 1, 3), [](7, 5, 6, 8, 9, 1, 2), [](8, 9, 7,
286      10, 4, 11, 5), [](8, 10, 9, 6, 11, 3, 2), [](9, 8, 12, 13, 7, 2, 6),
287      [](10, 9, 8, 11, 13, 12, 1), [](12, 13, 11, 9, 10, 8, 1), [](11, 10,
288      13, 3, 8, 9, 4), [](11, 12, 10, 7, 3, 13, 9), [](13, 11, 10, 9, 6, 1, 5),
289      [](13, 8, 11, 12, 4, 10, 3), [](10, 11, 9, 8, 12, 2, 3), [](9, 11, 6, 12,
290      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))

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

274 */

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.

2 Modéliser et contraindre

Question 5.1 On définit ici différents prédictats 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):-
```

```

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        ).
```

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

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

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

```

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

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

Listing 3 – "Prédicat pose_contraintes"

```
1  pose_contraintes(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     labeling(L).

```

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 été instances.

Listing 4 - "Prédicat **test** avec minimize"

```

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

```

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.

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

```

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

```

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

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

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

```

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

```

```

19 F = IJ(1, 0, 1, 0, 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.

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(Fabriquer,NbSortes):-
14     dim(Fabriquer,[NbSortes]),
15     ( for(I,1,NbSortes),param(Fabriquer)
16     do
17         Fabriquer[I] #:: 0..1
18     ).
19
20 getVarList(Fabriquer,NbSortes,L) :-
21     ( for(Ind,1,NbSortes),fromto([],In,Out,L),param(Fabriquer)
22     do
23         Var #= Fabriquer[Ind],
24         append(In,[Var],Out)
25     ).
26
27 nbOuvriersNecessaires(Fabriquer,Techniciens,NbSortes,NbOuvriersTotal) :-
28     ( for(J,1,NbSortes),fromto(0,In,Out,NbOuvriersTotal),param(Fabriquer,
29     Techniciens)
30     do
31         Var is Fabriquer[J],
32         NbTech is Techniciens[J],
33         Out #= In + NbTech * Var
34     ).
```

35 vectBeneficeTotal(Fabriquer,Benefice,Quantite,NbSortes,VectBenefTotal) :-
36 (for(K,1,NbSortes),fromto([],In,Out,VectBenefTotal),param(Fabriquer,
37 Benefice,Quantite)
38 do
39 Var is Fabriquer[K],
40 Benef is Benefice[K],
41 Quan is Quantite[K],
42 ELEM #= Var * Benef * Quan,
43 append(In,[ELEM],Out)
44).

45 profitTotal(Fabriquer,Benefice,Quantite,NbSortes,ProfitTotal) :-
46 vectBeneficeTotal(Fabriquer,Benefice,Quantite,NbSortes,VectBenefTotal
47),

```

47      ProfitTotal #= sum(VectBenefTotal).
48
49 pose_constraintes(Fabriquer ,NbTechniciensTotal ,Profit) :-  

50     getData(NbTech,NbSor,Tech,Quan,Bene),  

51     defineVars(Fabriquer ,NbSor),  

52     nbOuvriersNecessaires(Fabriquer ,Tech ,NbSor ,NbTechniciensTotal),  

53     NbTechniciensTotal #=< NbTech,  

54     getVarList(Fabriquer ,NbSor,L),  

55     profitTotal(Fabriquer ,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 ete instancees.  

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 reponses  

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#P*(-1), minimize(pose_constraintes(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

```

```

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 -1.0Inf .. -2666
107
108 P2 = -2665
109 P = 2665
110 F = [J(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_constraintes2(Fabriquer ,NbTechniciensTotal ,Profit) :-  

118     getData(NbTech,NbSor,Tech,Quan,Bene),  

119     defineVars(Fabriquer ,NbSor),  

120     nbOuvriersNecessaires(Fabriquer ,Tech ,NbSor ,NbTechniciensTotal),  

121     NbTechniciensTotal #=< NbTech,  

122     getVarList(Fabriquer ,NbSor,L),  

123     profitTotal(Fabriquer ,Bene ,Quan ,NbSor ,Profit),  

124     Profit #>=1000,  

125     labeling(L).
126
127 /* Tests
128 [eclipse 74]: minimize(pose_constraintes2(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 -1.0Inf .. 6
134
135 F = [J(1, 0, 1, 0, 0, 0, 0, 0, 0)
136 N = 7
137 P = 1040
138 Yes (0.00s cpu)
139 */


```

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.

2 Modéliser et contraindre

Question 5.1 On définit ici différents prédictats 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):-
```

```

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        ).
```

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

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

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

```

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

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

Listing 3 – "Prédicat pose_contraintes"

```
1  pose_contraintes(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     labeling(L).

```

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 été instantierées.

Listing 4 - "Prédicat **test** avec minimize"

```

1 test(X,Y,Z,W) :- 
2   [X,Y,Z,W] #:: [0..10],
3   X #= Z+Y+2*W,
4   X#!= Z+Y+W.
5 
6 /* Test
7 
8 [eclipse 53]: minimize(test(X,Y,Z,W),X).
9 bb_min: search did not instantiate cost variable
10 Aborting execution ...
11 Abort
12 
13 Avec le labeling sur [X,Y,Z,W], cela fonctionne, Prolog trouve les réponses
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.

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

```

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

```

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

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

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

```

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

```

```

19 F = IJ(1, 0, 1, 0, 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.

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(Fabriquer,NbSortes):-
14     dim(Fabriquer,[NbSortes]),
15     ( for(I,1,NbSortes),param(Fabriquer)
16     do
17         Fabriquer[I] #:: 0..1
18     ).
19
20 getVarList(Fabriquer,NbSortes,L) :-
21     ( for(Ind,1,NbSortes),fromto([],In,Out,L),param(Fabriquer)
22     do
23         Var #= Fabriquer[Ind],
24         append(In,[Var],Out)
25     ).
26
27 nbOuvriersNecessaires(Fabriquer,Techniciens,NbSortes,NbOuvriersTotal) :-
28     ( for(J,1,NbSortes),fromto(0,In,Out,NbOuvriersTotal),param(Fabriquer,
29     Techniciens)
30     do
31         Var is Fabriquer[J],
32         NbTech is Techniciens[J],
33         Out #= In + NbTech * Var
34     ).
```

35 vectBeneficeTotal(Fabriquer,Benefice,Quantite,NbSortes,VectBenefTotal) :-
36 (for(K,1,NbSortes),fromto([],In,Out,VectBenefTotal),param(Fabriquer,
37 Benefice,Quantite)
38 do
39 Var is Fabriquer[K],
40 Benef is Benefice[K],
41 Quan is Quantite[K],
42 ELEM #= Var * Benef * Quan,
43 append(In,[ELEM],Out)
44).

45 profitTotal(Fabriquer,Benefice,Quantite,NbSortes,ProfitTotal) :-
46 vectBeneficeTotal(Fabriquer,Benefice,Quantite,NbSortes,VectBenefTotal
47),

```

47      ProfitTotal #= sum(VectBenefTotal).
48
49 pose_constraintes(Fabriquer ,NbTechniciensTotal ,Profit) :-  

50     getData(NbTech,NbSor,Tech,Quan,Bene),  

51     defineVars(Fabriquer ,NbSor),  

52     nbOuvriersNecessaires(Fabriquer ,Tech ,NbSor ,NbTechniciensTotal),  

53     NbTechniciensTotal #=< NbTech,  

54     getVarList(Fabriquer ,NbSor,L),  

55     profitTotal(Fabriquer ,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 ete instancees.  

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 reponses  

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#P*(-1), minimize(pose_constraintes(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

```

```

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 -1.0Inf .. -2666
107
108 P2 = -2665
109 P = 2665
110 F = [J(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_constraintes2(Fabriquer ,NbTechniciensTotal ,Profit) :-  

118     getData(NbTech,NbSor,Tech,Quan,Bene),  

119     defineVars(Fabriquer ,NbSor),  

120     nbOuvriersNecessaires(Fabriquer ,Tech ,NbSor ,NbTechniciensTotal),  

121     NbTechniciensTotal #=< NbTech,  

122     getVarList(Fabriquer ,NbSor,L),  

123     profitTotal(Fabriquer ,Bene ,Quan ,NbSor ,Profit),  

124     Profit #>=1000,  

125     labeling(L).
126
127 /* Tests
128 [eclipse 74]: minimize(pose_constraintes2(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 -1.0Inf .. 6
134
135 F = [J(1, 0, 1, 0, 0, 0, 0, 0, 0)
136 N = 7
137 P = 1040
138 Yes (0.00s cpu)
139 */

```

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 Trouver une solution au problème

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

Listing 1 – "Solution 1"

```

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 balançoireEquilibree(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
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] ,

```

```

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 solve(Places) :-  

76     getData(Poids ,NbPers ,NbPlac),  

77     defineVars(Places ,NbPlac ,NbPers),  

78     pasMemesPlaces(Places),  

79     balancoireEquilibree(Poids ,NbPers ,Places),  

80     cinqChaqueCote(Places ,NbPers),  

81     parentsEncadrentEnfants(Places),  

82     danEtMaxDevantParents(Places),  

83     getVarList(Places ,NbPers ,L),  

84     labeling(L).
85
86 /* Tests
87 [eclipse 29]: solve(P).
88
89 P = [I(-6, -5, -4, -8, 2, 5, 3, 6, -7, 1)
90 Yes (0.02s cpu, solution 1, maybe more) ? ;
91
92 P = [I(-6, -5, -1, 8, 5, 7, 3, -8, -7, 1)
93 Yes (0.02s cpu, solution 2, maybe more) ? ;

```

```

94
95 P = [I(-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 = [I(-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 balancoire
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 = [I(-6, -5, -4, -8, 2, 5, 3, 6, -7, 1)
17 Yes (0.02s cpu, solution 1, maybe more) ? ;
18
19 P = [I(-6, -5, 1, -8, -2, 6, 5, 7, -7, 4)
20 Yes (0.04s cpu, solution 2, maybe more) ? ;
21
22 P = [I(-6, -5, 3, -8, -3, -7, 4, 7, 6, 5)
23 Yes (0.01s cpu, solution 3, maybe more) ?  

24 */

```

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 effectuer 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 prédictat solve, afin qu'il permettent de rechercher le minimum des moments de forces que l'on vient de calculer.

Listing 3 – "Solution 3"

```

1 balancoireEquilibree2(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            (Places[J] #=< 0) => (Out2 #= In2)  
11        ),  
12        SommeMoments #= 0.  
13  
14 solve3(Places) :-  
15     getData(Poids ,NbPers ,NbPlac) ,  
16     defineVars(Places ,NbPlac ,NbPers) ,  
17     elimineSymetrie(Places) ,%Gain de temps monstrueux  
18     pasMemesPlaces(Places) ,  
19     cinqChaqueCote(Places ,NbPers) ,  
20     balancoireEquilibree2(Poids ,NbPers ,Places ,SNormeMom) ,  
21     parentsEncadrentEnfants(Places) ,  
22     danEtMaxDevantParents(Places) ,  
23     getVarList(Places ,NbPers ,L) ,  
24     minimize(labeling(L) ,SNormeMom) .  
25 /* Tests  
26 [eclipse 25]: solve3(P).  
27 Found a solution with cost 874  
28 Found a solution with cost 872  
29 Found a solution with cost 802  
30 Found no solution with cost -1.0Inf .. 801  
31 P = [J(-3, 1, -2, -6, -1, 4, 3, 5, -5, 2)  
32 Yes (1.66s cpu)  
33 */

```

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

Utilisation du prédictat search/6

Dans cette partie, nous avons utiliser le prédictat 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 testée 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     balancoireEquilibree2(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 -1.0Inf .. 801  
20  
21 P = [J(-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(_H,[],[],[]).  
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

```

```

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     balancoireEquilibree2(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 -1.0Inf .. 801
43
44 P = [1(-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     balancoireEquilibree2(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

```

```

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 -1.0Inf .. 801
23
24 P = [1(-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.

3 Code Complet, avec l'ensemble des tests

Listing 7 – "TP6"

```

1 %TP6
2 :- lib(ic).
3 :- lib(branch_and_bound).
4 :- lib(ic_global).
5 :- lib(branch_and_bound).
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
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     ),.

```

```

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 - 1) and (PlacePere #= PlaceMax + 1)))
68     or
69     (((PlaceMere #<0) and (PlaceMax #<0)) and ((PlaceMere #= PlaceMax - 1) and (PlacePere #= PlaceDan + 1)))
70     or
71     (((PlacePere #<0) and (PlaceDan #<0)) and ((PlacePere #= PlaceDan - 1) and (PlaceMere #= PlaceMax + 1)))
72     or
73     (((PlacePere #<0) and (PlaceMax #<0)) and ((PlacePere #= PlaceMax - 1) and (PlaceMere #= PlaceDan + 1)))
74
75 solve(Places) :-+
76     getData(Poids, NbPers, NbPlaces),
77     defineVars(Places, NbPlaces, NbPers),
78     pasMemesPlaces(Places),
79     balancoireEquilibree(Poids, NbPers, Places),
80     cinqChaqueCote(Places, NbPers),
81     parentsEncadrentEnfants(Places),
82     danEtMaxDevantParents(Places),
83     getVarList(Places, NbPers, L),
84     labeling(L).
85
86 /* Tests
87 [eclipse 29]: solve(P).
88
89 P = [J(-6, -5, -4, -8, 2, 5, 3, 6, -7, 1)
90 Yes (0.13s cpu, solution 1, maybe more) ? ;
91
92 P = [J(-6, -5, -1, 8, 5, 7, 3, -8, -7, 1)
93 Yes (0.15s cpu, solution 2, maybe more) ? ;
94
95 P = [J(-6, -5, 1, -8, -2, 6, 5, 7, -7, 4)
96 Yes (0.02s cpu, solution 3, maybe more) ?
```

```

97  */
98
99 % Impose que la mere soit a gauche sur la balancoire
100 elimineSymetrie(Places) :-  

101   Places[4] #< 0.  

102
103 solve2(Places) :-  

104   getData(Poids ,NbPers ,NbPlac ),  

105   defineVars(Places ,NbPlac ,NbPers ),  

106   elimineSymetrie(Places) ,%Gain de temps monstrueux  

107   pasMemesPlaces(Places) ,  

108   cinqChaqueCote(Places ,NbPers ) ,  

109   balancoireEquilibree(Poids ,NbPers ,Places) ,  

110   parentsEncadrentEnfants(Places) ,  

111   danEtMaxDevantParents(Places) ,  

112   getVarList(Places ,NbPers ,L) ,  

113   labeling(L).  

114
115 /* Tests  

116 [eclipse 31]: solve2(P).  

117
118 P = [J(-6, -5, -4, -8, 2, 5, 3, 6, -7, 1)  

119 Yes (0.02s cpu, solution 1, maybe more) ? ;  

120
121 P = [J(-6, -5, 1, -8, -2, 6, 5, 7, -7, 4)  

122 Yes (0.04s cpu, solution 2, maybe more) ? ;  

123
124 P = [J(-6, -5, 3, -8, -3, -7, 4, 7, 6, 5)  

125 Yes (0.01s cpu, solution 3, maybe more) ?  

126 */
127
128 balancoireEquilibree2(Poids ,NbPersonnes ,Places ,SNormeMoments) :-  

129   ( for(J,1,NbPersonnes) ,fromto(0,In ,Out ,SommeMoments) ,fromto(0,In2 ,  

130     Out2 ,SNormeMoments) ,param(Places ,Poids )  

131     do  

132       Distance #= Places[J],  

133       Poids1 #= Poids[J],  

134       Out #= In + Distance * Poids1 ,  

135       (Places[J] #> 0 => (Out2 #= In2 + Distance * Poids1)  

136         ,  

137         (Places[J] #=< 0 => (Out2 #= In2)  

138       ).  

139 SommeMoments #= 0.  

140
141 solve3(Places) :-  

142   getData(Poids ,NbPers ,NbPlac ),  

143   defineVars(Places ,NbPlac ,NbPers ),  

144   elimineSymetrie(Places) ,%Gain de temps monstrueux  

145   pasMemesPlaces(Places) ,  

146   cinqChaqueCote(Places ,NbPers ) ,  

147   balancoireEquilibree2(Poids ,NbPers ,Places ,SNormeMoments) ,  

148   parentsEncadrentEnfants(Places) ,  

149   danEtMaxDevantParents(Places) ,  

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 -1.0Inf .. 801  

157
158 P = [J(-3, 1, -2, -6, -1, 4, 3, 5, -5, 2)  

159 Yes (1.66s 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 ,SNormeMoments) ,  

171   parentsEncadrentEnfants(Places) ,  

172   danEtMaxDevantParents(Places) ,  

173   getVarList(Places ,NbPers ,L) ,  

174   minimize(search(L,0,occurrence ,indomain_min ,complete ,[]),SNormeMoments).  

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 -1.0Inf .. 801  

183
184 P = [J(-3, 1, -2, -6, -1, 4, 3, 5, -5, 2)  

185 Yes (2.50s cpu)  

186 */
187 % Ici , il semble que most_constrained soit plus adapte que occurrence , vu la  

188 vitesse de reponse (2.13s au lieu de 2.50s)  

189
190 % Version 2 de search :  

191 /* Ici on vise a essayer de mettre les places les plus au centre en premier ,  

192 parce que c'est dans ce cas que l'on minimise le moment total . */  

193 quick_sort([],[]).  

194 quick_sort([H|T],Sorted):-  

195   quick_sort(T,Sorted1),  

196   pivoting(H,T,L1,L2),quick_sort(L1,Sorted1),quick_sort(L2,Sorted2),  

197   append(Sorted1,[H|Sorted2],Sorted).  

198 pivoting(_H,[ ],[ ],[ ]).  

199 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 /* Tests
202 [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     balançoireEquilibree2(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 -1.0Inf .. 801
235
236 P = [1(-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     balançoireEquilibree2(Poids ,NbPers ,Places ,SNormeMom),
249     parentsEncadrentEnfants(Places),
250     danEtMaxDevantParents(Places),
251     getVarList(Places ,NbPers ,L),
252     minimize(search(L,0,occurrence,choice,complete,[]),SNormeMom).
```

```

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 -1.0Inf .. 801
264
265 P = [1(-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     balançoireEquilibree2(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
284 ).
```

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 Trouver une solution au problème

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

Listing 1 – "Solution 1"

```

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 balançoireEquilibree(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
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] ,

```

```

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 solve(Places) :-  

76     getData(Poids ,NbPers ,NbPlac),  

77     defineVars(Places ,NbPlac ,NbPers),  

78     pasMemesPlaces(Places),  

79     balancoireEquilibree(Poids ,NbPers ,Places),  

80     cinqChaqueCote(Places ,NbPers),  

81     parentsEncadrentEnfants(Places),  

82     danEtMaxDevantParents(Places),  

83     getVarList(Places ,NbPers ,L),  

84     labeling(L).
85
86 /* Tests
87 [eclipse 29]: solve(P).
88
89 P = [I(-6, -5, -4, -8, 2, 5, 3, 6, -7, 1)
90 Yes (0.02s cpu, solution 1, maybe more) ? ;
91
92 P = [I(-6, -5, -1, 8, 5, 7, 3, -8, -7, 1)
93 Yes (0.02s cpu, solution 2, maybe more) ? ;

```

```

94
95 P = [I(-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 = [I(-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 balancoire
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 = [I(-6, -5, -4, -8, 2, 5, 3, 6, -7, 1)
17 Yes (0.02s cpu, solution 1, maybe more) ? ;
18
19 P = [I(-6, -5, 1, -8, -2, 6, 5, 7, -7, 4)
20 Yes (0.04s cpu, solution 2, maybe more) ? ;
21
22 P = [I(-6, -5, 3, -8, -3, -7, 4, 7, 6, 5)
23 Yes (0.01s cpu, solution 3, maybe more) ?  

24 */

```

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 effectuer 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 prédictat solve, afin qu'il permettent de rechercher le minimum des moments de forces que l'on vient de calculer.

Listing 3 – "Solution 3"

```

1 balancoireEquilibree2(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            (Places[J] #=< 0) => (Out2 #= In2)  
11        ),  
12        SommeMoments #= 0.  
13  
14 solve3(Places) :-  
15     getData(Poids ,NbPers ,NbPlac) ,  
16     defineVars(Places ,NbPlac ,NbPers) ,  
17     elimineSymetrie(Places) ,%Gain de temps monstrueux  
18     pasMemesPlaces(Places) ,  
19     cinqChaqueCote(Places ,NbPers) ,  
20     balancoireEquilibree2(Poids ,NbPers ,Places ,SNormeMom) ,  
21     parentsEncadrentEnfants(Places) ,  
22     danEtMaxDevantParents(Places) ,  
23     getVarList(Places ,NbPers ,L) ,  
24     minimize(labeling(L) ,SNormeMom) .  
25 /* Tests  
26 [eclipse 25]: solve3(P).  
27 Found a solution with cost 874  
28 Found a solution with cost 872  
29 Found a solution with cost 802  
30 Found no solution with cost -1.0Inf .. 801  
31 P = [J(-3, 1, -2, -6, -1, 4, 3, 5, -5, 2)  
32 Yes (1.66s cpu)  
33 */

```

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

Utilisation du prédictat search/6

Dans cette partie, nous avons utiliser le prédictat 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 testée 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     balancoireEquilibree2(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 -1.0Inf .. 801  
20  
21 P = [J(-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(_H,[],[],[]).  
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

```

```

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     balancoireEquilibree2(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 -1.0Inf .. 801
43
44 P = [1(-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     balancoireEquilibree2(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

```

```

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 -1.0Inf .. 801
23
24 P = [1(-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.

3 Code Complet, avec l'ensemble des tests

Listing 7 – "TP6"

```

1 %TP6
2 :- lib(ic).
3 :- lib(branch_and_bound).
4 :- lib(ic_global).
5 :- lib(branch_and_bound).
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
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
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96

```

```

NbPersGauche #= NbPersonnesAGauche .  

parentsEncadrentEnfants(Places) :-  

    PlaceMere #= Places[4],  

    PlacePere #= Places[8],  

    ((PlaceMere #= ic:max(Places)) and (PlacePere #= ic:min(Places)))  

    or  

    ((PlacePere #= ic:max(Places)) and (PlaceMere #= ic:min(Places))).  

/*  

(maxlist(Places, PlaceMere) and minlist(Places, PlacePere))  

or  

(maxlist(Places, PlacePere) and minlist(Places, PlaceMere)).  

*/  

danEtMaxDevantParents(Places) :-  

    PlaceDan #= Places[6],  

    PlaceMax #= Places[9],  

    PlaceMere #= Places[4],  

    PlacePere #= Places[8],  

    (((PlaceMere #<0) and (PlaceDan #<0)) and ((PlaceMere #= PlaceDan -  

        1) and (PlacePere #= PlaceMax + 1)))  

    or  

    (((PlaceMere #<0) and (PlaceMax #<0)) and ((PlaceMere #= PlaceMax -  

        1) and (PlacePere #= PlaceDan + 1)))  

    or  

    (((PlacePere #<0) and (PlaceDan #<0)) and ((PlacePere #= PlaceDan -  

        1) and (PlaceMere #= PlaceMax + 1)))  

    or  

    (((PlacePere #<0) and (PlaceMax #<0)) and ((PlacePere #= PlaceMax -  

        1) and (PlaceMere #= PlaceDan + 1))).  

solve(Places) :-  

    getData(Poids, NbPers, NbPlaces),  

    defineVars(Places, NbPlaces, NbPers),  

    pasMemesPlaces(Places),  

    balancoireEquilibree(Poids, NbPers, Places),  

    cinqChaqueCote(Places, NbPers),  

    parentsEncadrentEnfants(Places),  

    danEtMaxDevantParents(Places),  

    getVarList(Places, NbPers, L),  

    labeling(L).  

/* Tests  

[eclipse 29]: solve(P).  

88  

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

90 Yes (0.13s cpu, solution 1, maybe more) ? ;  

91  

92 P = [J(-6, -5, -1, 8, 5, 7, 3, -8, -7, 1)  

93 Yes (0.15s cpu, solution 2, maybe more) ? ;  

94  

95 P = [J(-6, -5, 1, -8, -2, 6, 5, 7, -7, 4)  

96 Yes (0.02s cpu, solution 3, maybe more) ?
```

```

97  */
98
99 % Impose que la mere soit a gauche sur la balancoire
100 elimineSymetrie(Places) :-  

101   Places[4] #< 0.  

102
103 solve2(Places) :-  

104   getData(Poids ,NbPers ,NbPlac ),  

105   defineVars(Places ,NbPlac ,NbPers ),  

106   elimineSymetrie(Places) ,%Gain de temps monstrueux  

107   pasMemesPlaces(Places) ,  

108   cinqChaqueCote(Places ,NbPers ) ,  

109   balancoireEquilibree(Poids ,NbPers ,Places) ,  

110   parentsEncadrentEnfants(Places) ,  

111   danEtMaxDevantParents(Places) ,  

112   getVarList(Places ,NbPers ,L) ,  

113   labeling(L).  

114
115 /* Tests  

116 [eclipse 31]: solve2(P).  

117
118 P = [J(-6, -5, -4, -8, 2, 5, 3, 6, -7, 1)  

119 Yes (0.02s cpu, solution 1, maybe more) ? ;  

120
121 P = [J(-6, -5, 1, -8, -2, 6, 5, 7, -7, 4)  

122 Yes (0.04s cpu, solution 2, maybe more) ? ;  

123
124 P = [J(-6, -5, 3, -8, -3, -7, 4, 7, 6, 5)  

125 Yes (0.01s cpu, solution 3, maybe more) ?  

126 */
127
128 balancoireEquilibree2(Poids ,NbPersonnes ,Places ,SNormeMoments) :-  

129   ( for(J,1,NbPersonnes) ,fromto(0,In ,Out ,SommeMoments) ,fromto(0,In2 ,  

130     Out2 ,SNormeMoments) ,param(Places ,Poids )
131     do
132       Distance #= Places[J] ,
133       Poids1 #= Poids[J] ,
134       Out #= In + Distance * Poids1 ,
135       (Places[J] #> 0) => (Out2 #= In2 + Distance * Poids1)
136       ,
137       (Places[J] #=< 0) => (Out2 #= In2)
138     ),
139   SommeMoments #= 0.
140
141 solve3(Places) :-  

142   getData(Poids ,NbPers ,NbPlac ),  

143   defineVars(Places ,NbPlac ,NbPers ),  

144   elimineSymetrie(Places) ,%Gain de temps monstrueux  

145   pasMemesPlaces(Places) ,  

146   cinqChaqueCote(Places ,NbPers ) ,  

147   balancoireEquilibree2(Poids ,NbPers ,Places ,SNormeMoments) ,  

148   parentsEncadrentEnfants(Places) ,  

149   danEtMaxDevantParents(Places) ,  

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 -1.0Inf .. 801  

157
158 P = [J(-3, 1, -2, -6, -1, 4, 3, 5, -5, 2)  

159 Yes (1.66s 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 ,SNormeMoments) ,  

171   parentsEncadrentEnfants(Places) ,  

172   danEtMaxDevantParents(Places) ,  

173   getVarList(Places ,NbPers ,L) ,  

174   minimize(search(L,0,occurrence ,indomain_min ,complete ,[]),SNormeMoments).
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 -1.0Inf .. 801  

183
184 P = [J(-3, 1, -2, -6, -1, 4, 3, 5, -5, 2)  

185 Yes (2.50s cpu)  

186 */
187 % Ici , il semble que most_constrained soit plus adapte que occurrence , vu la  

188 vitesse de reponse (2.13s au lieu de 2.50s)
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 moment total . */
194 quick_sort([],[]).
195 quick_sort([H|T],Sorted):-
196   quick_sort(T,Sorted1),
197   quick_sort([H|T],Sorted2),
198   pivoting(H,T,L1,L2),
199   quick_sort(L1,Sorted1),
200   quick_sort(L2,Sorted2),
201   append(Sorted1,[H|Sorted2],Sorted).
202
203 pivoting(_H,[ ],[ ],[ ]).
204 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 /* Tests
202 [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     balançoireEquilibree2(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 -1.0Inf .. 801
235
236 P = [1(-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     balançoireEquilibree2(Poids ,NbPers ,Places ,SNormeMom),
249     parentsEncadrentEnfants(Places),
250     danEtMaxDevantParents(Places),
251     getVarList(Places ,NbPers ,L),
252     minimize(search(L,0,occurrence,choice,complete,[]),SNormeMom).
```

```

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 -1.0Inf .. 801
264
265 P = [1(-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     balançoireEquilibree2(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
284 ).
```

Table of Content

Page 1/1

Table of Contents

- | | | | | | | |
|----|---|--------------|----------|-------------|---------|---------|
| 1 | <i>Contraintes-TP1-DESFEUX-TEXIER.pdf</i> | sheets | 1 to 9 | (9) pages | 2- 18 | 1 lines |
| 2 | <i>cr-tp2.pdf</i> | sheets | 9 to 14 | (6) pages | 18- 28 | 1 lines |
| 3 | <i>TP2 Contraintes - Desfeux - Texier.pdf</i> | sheets | 14 to 19 | (6) pages | 28- 38 | 1 lines |
| 4 | <i>cr-tp2.pdf</i> | sheets | 19 to 26 | (8) pages | 38- 52 | 1 lines |
| 5 | <i>TP3_prog_constraintes_DESFEUX_TEXIER_G22.pdf</i> | sheets | 26 to 33 | (8) pages | 52- 66 | 1 lines |
| 6 | <i>cr-tp2.pdf</i> | sheets | 33 to 41 | (9) pages | 66- 82 | 1 lines |
| 7 | <i>DESFEUX_TEXIER_TP4.pdf</i> | sheets | 41 to 49 | (9) pages | 82- 98 | 1 lines |
| 8 | <i>cr-tp5.pdf</i> | sheets | 49 to 53 | (5) pages | 98-106 | 1 lines |
| 9 | <i>TP5-DESFEUX-TEXIER.pdf</i> | sheets | 53 to 57 | (5) pages | 106-114 | 1 lines |
| 10 | <i>cr-tp6.pdf</i> | sheets | 57 to 64 | (8) pages | 114-128 | 1 lines |
| 11 | <i>DESFEUX-TEXIER-TP6.pdf</i> | sheets | 64 to 71 | (8) pages | 128-142 | 1 lines |