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

Très bon compte-rendu.

Rapport Travaux Pratiques : Programmation par Contraintes - TP 1 : **Découverte de la bibliothèque de contraintes à domaines finis**

Nicolas Desfeux
Aurélien Texier

23 février 2011

Table des matières

1 De Prolog à Prolog+ic

1.1 Des contraintes sur les arbres

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

Listing 1 – "Prédicat choixCouleur"

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

Bien

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

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

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

Question 1.3 Voici le code du prédicat isBetween :

Listing 2 – "Prédicat isBetween"

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

Tests ??

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

Listing 3 – "Prédicat commande"

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

Bien

Estimation du nombre de solutions ?

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

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

OK

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

Pas très rigoureux dans la forme...

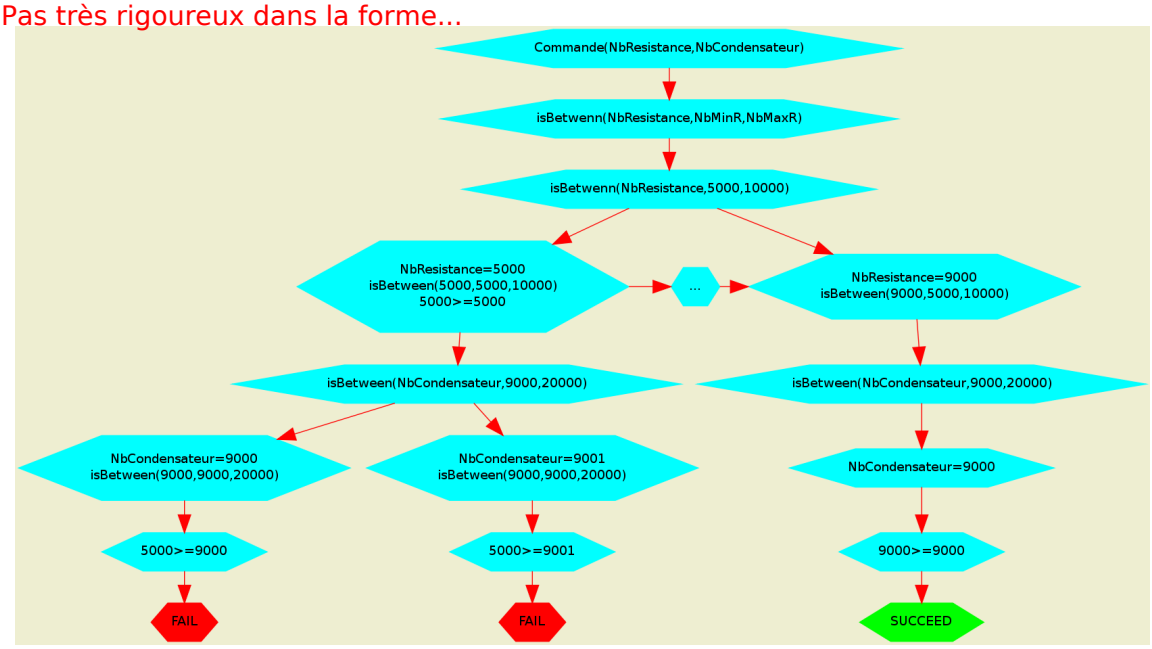

FIGURE 1 – Arbre Prolog 1

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

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

1.3 Le solveur ic à la rescousse

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

Listing 4 – "prédicat Commande"

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

```
15  Yes (0.00s cpu)
16  */
```

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

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

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

Listing 5 – "prédicat Commande"

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

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

OK

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

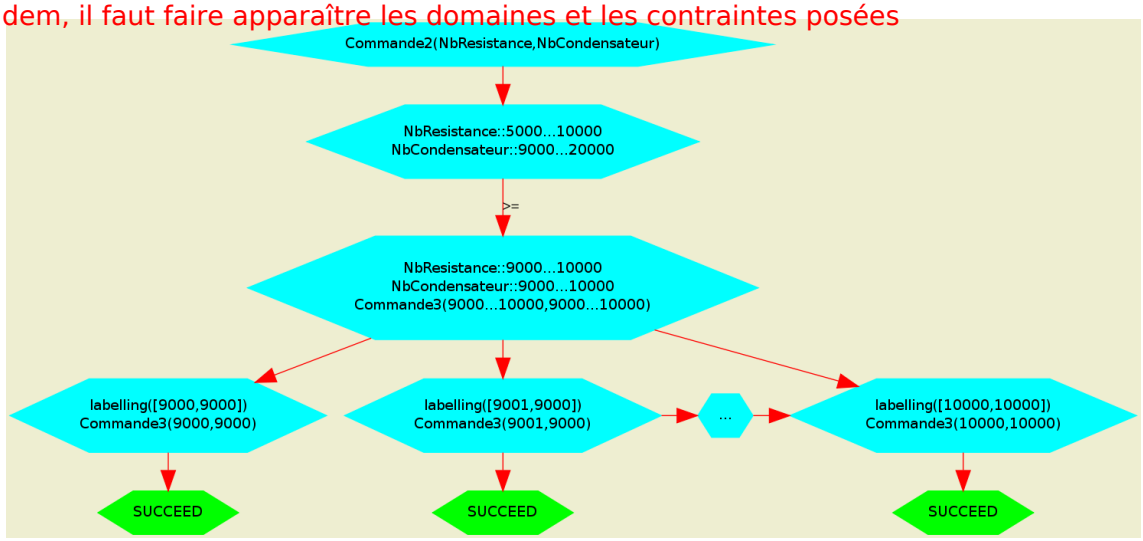

FIGURE 2 – Arbre Prolog 2

2 Zoologie

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

Listing 6 – "Prédicat chapie"

```
1  nbChatsMax(1000).
2  nbPiesMax(1000).
3
4  chapie(Chats,Pies,Pattes,Tetes) :-      nbChatsMax(NbChatsMax),
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 0,0,0,0 puis 1,1,6,2, etc.

Vraiment ???

Listing 8 – "Prédicat chapie - test"

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

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

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

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

3 Le "OU" en contraintes

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

Listing 9 – "Prédicat vabs - vabs2"

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

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

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

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

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

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

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

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

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

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

X = X{-1.0Inf .. 1.0Inf}


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

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

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

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

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.

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

Question 1.13 Définition du prédicat *faitListe/4* :

Listing 10 – "Prédicat faitListe"

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

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

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

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

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

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

Indentez mieux...

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

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

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

Question 1.14 Définition du prédicat *suite/1* :

Listing 11 – "Prédicat suite"

```
1  suite(ListVar) :-       dim(ListVar,[Taille]),
2                                  (for(I,1,Taille-2),param(ListVar) do
3                                          (Temp #= ListVar[I+1],
4                                          vabs1(Temp,VabsDeuxElem),
5                                          ListVar[I+2] #= VabsDeuxElem
                                               - ListVar[I]
6                                          )
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 prédicat avec les 2 premiers éléments de cette suite compris entre 1 et 10. Effectivement, il semble y avoir périodicité entre le premier et le dixième élément (et donc une période de 9).

Listing 12 – "Réquête"

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

Utilisez plutôt labeling !!!

Il fallait utiliser faitListe

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

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

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

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

4 Code Complet, avec tests

Listing 13 – "test"

```
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),
48                                                  NbResistance>=NbCondensateur.
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                                                  NbResistance>=NbCondensateur,
70                                                  isBetween(NbResistance,NbMinR
                                                        ,NbMaxR),
71                                                  isBetween(NbCondensateur,
                                                        NbMinC,NbMaxC).
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                                                  NbResistance #:: NbMinR..
                                                        NbMaxR,
85                                                  NbCondensateur #:: NbMinC..
                                                        NbMaxC,
86                                                  NbResistance #>=
                                                        NbCondensateur.
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)
        ,nbMaxC(NbMaxC),nbMinC(NbMinC),
103                                                 NbResistance #:: NbMinR..
                                                        NbMaxR,
104                                                 NbCondensateur #:: NbMinC..
                                                        NbMaxC,
105                                                 NbResistance #>=
                                                        NbCondensateur,
106                                                 labeling([NbResistance,
                                                        NbCondensateur]).
107
108 /* Tests
109 [eclipse 91]: commande3(NbResistance,NbCondensateur).
110
111 NbResistance = 9000
112 NbCondensateur = 9000
113 Yes (0.00s cpu, solution 1, maybe more) ? ;
114
115 NbResistance = 9001
116 NbCondensateur = 9000
117 Yes (0.00s cpu, solution 2, maybe more) ? ;
118
119 NbResistance = 9001
120 NbCondensateur = 9001
121 Yes (0.00s cpu, solution 3, maybe more) ? ;
122
123 NbResistance = 9002
124 NbCondensateur = 9000
125 Yes (0.00s cpu, solution 4, maybe more) ?
126
127 */
128
129 %Zoologie
130 %Question 1.9
131
132 nbChatsMax(1000).
133 nbPiesMax(1000).
134
135 chapie(Chats,Pies,Pattes,Tetes) :-          nbChatsMax(NbChatsMax),
136                                             nbPiesMax(NbPiesMax),
137                                             Chats #:: 0 .. NbChatsMax,
138                                             Pies #:: 0 .. NbPiesMax,
139                                             Pattes #= Chats*4+Pies*2,
```

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

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

% Question 1.10

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

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

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

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

% Question 1.11

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

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

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

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

Y = 2
Yes (0.00s cpu)


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

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

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

X = X{-1.0Inf .. 1.0Inf}


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

% Question 1.13

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

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

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

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

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

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

Autre test
```

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

No (0.00s cpu)

*/

% Question 1.14

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

Yes (0.00s cpu)

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

No (0.00s cpu)
*/

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

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

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