CR TP6 Programmation par contraintes

Sur une balançoire

Julien LETOILE, Romain HUBERT

le 17/03/2021

Table des matières

I. Réponses rédigées

Question 6.1

```
1  /* Pose les contraintes sur les données */
2  getData(Poids, Personnes):-
3      Personnes = [](ron, zoe, jim, lou, luc, dan, ted, tom, max,
   kim),
4      Personnes &:: noms,
5      Poids = [](24, 39, 85, 60, 165, 6, 32, 123, 7, 14).
6
7  /* Pose les contraintes sur les variables */
8  getVar(Places, Longueur):-
9      dim(Places, [Longueur]),
10     Places #:: [-8 .. -1, 1 .. 8].
11
12
13 pose_contrainte(Places, Poids, Personnes):-
14     equilibre(Places, Poids),
15     contraindreLouEtTom(Places, Personnes),
16     contraindreDanEtMax(Places, Personnes),
17     cinqDeChaqueCote(Places),
18     enleveSymetrie(Places, Personnes),
19     ic:alldifferent(Places).
20
21 /* Garantit que la balançire est en équilibre */
22 equilibre(Places, Poids):-
23     produitScalaire(Places, Poids, Somme),
24     Somme #= 0.
25
26 /* Place Lou et Tom aux extrémités */
27 contraindreLouEtTom(Places, Personnes):-
28     ic:max(Places, Max),
29     ic:min(Places, Min),
30     indicePersonne(lou, Personnes, IndiceLou),
31     indicePersonne(tom, Personnes, IndiceTom),
32     Places[IndiceLou] #>= Max or Places[IndiceLou] #=< Min,
33     Places[IndiceTom] #>= Max or Places[IndiceTom] #=< Min.
```

```
34
35  /* Place Dan et Max devant Lou et Tom */
36  contraindreDanEtMax(Places, Personnes):-
37      indicePersonne(dan, Personnes, Dan),
38      indicePersonne(max, Personnes, Max),
39      indicePersonne(lou, Personnes, Lou),
40      indicePersonne(tom, Personnes, Tom),
41      (Places[Lou] > 0) => ((Places[Dan] #= Places[Lou] - 1 and
    Places[Max] #= Places[Tom] + 1) or (Places[Max] #= Places[Lou] -
    1 and Places[Dan] #= Places[Tom] + 1)),
42      (Places[Lou] < 0) => ((Places[Dan] #= Places[Lou] + 1 and
    Places[Max] #= Places[Tom] - 1) or (Places[Max] #= Places[Lou] +
    1 and Places[Dan] #= Places[Tom] - 1)).
43
44  /* Garantit que 5 persones seront de part de d'autre de la
    balançoire */
45  cinqDeChaqueCote(Places):-
46      (
47          foreachelem(Place, Places),
48          fromto(0, AncienCompteGauche, NouveauCompteGauche, 5)
49      do
50          NouveauCompteGauche #= AncienCompteGauche + (Place #< 0)
51      ).
52
53  /* Recherche l'indice d'une personne */
54  indicePersonne(Personne, Personnes, Indice):-
55      dim(Personnes, [Longueur]),
56      (
57          for(I, 1, Longueur),
58          param(Personne, Personnes, Indice)
59      do
60          X is Personnes[I],
61          /* Toujours mettre un else dans un if (cond -> true;
    false) */
62          (Personne &= X -> Indice #= I; true)
63      ).
64
65
66  getVarList(Places, List):-
67      term_variables(Places, List).
```

Question 6.2

```
/* Prédicat de résolution du problème */
solve:-
    getData(Poids, Personnes),
    dim(Poids, [Longueur]),
    getVar(Places, Longueur),
    pose_contrainte(Places, Poids, Personnes),
    norme(Places, Poids, Norme),
    labeling(Places),
    affiche(Places, Personnes, Norme).
```

Test:

```
tom    dan    ron              jim        ted | kim              zoe    luc    max    lou
.  -8     -7     -6     -5     -4     -3     -2     -1  |  1      2      3      4      5      6      7
8

Norme : 2914
```

Question 6.3

Les personnes peuvent s'assoir sur le même siège mais de l'autre côté de la balancoire, entrainant ainsi la même solution mais à l'opposé.
On a alors une symetrie par rapport au milieu. Ainsi, on aura 2 fois moins de solutions, et on ne gardera que les solutions non triviales.

```
/* Pour enlever la symetrie, il suffit de choisir un côté pour soit Lou, soit Tom vu qu'on sait qu'ils sont de part et d'autres de la balancoire */
enleveSymetrie(Places, Personnes):-
    indicePersonne(lou, Personnes, IndiceLou),
    Places[IndiceLou] #> 0. /* Choisit de mettre la place de Lou à droite */
```

Question 6.4

Version de base

```
/* Prédicat de résolution avec Branch and Bound */
solve_minimize_v0:-
    getData(Poids, Personnes),
    dim(Poids, [Longueur]),
    getVar(Places, Longueur),

    pose_contrainte(Places, Poids, Personnes),
    norme(Places, Poids, Norme),

    /* Recherche par le plus contraint, dans l'ordre croissant
*/
    minimize(labeling(Places), Norme),
    affiche(Places, Personnes, Norme).
```

Test:

```
                tom  dan   ted   kim   zoe | luc   jim   ron       max   lou
-8   -7   -6   -5   -4   -3   -2   -1  |  1    2     3     4     5     6     7     8

Norme : 802

P = [](3, -1, 2, 6, 1, -4, -3, -5, 5, -2)
Yes (2.13s cpu)
```

Version 1

```
/* On essaie d'échouer le plus tôt possible pour éviter de développer des branches inutiles */
solve_minimize_v1:-
    getData(Poids, Personnes),
    dim(Poids, [Longueur]),
    getVar(Places, Longueur),

    pose_contrainte(Places, Poids, Personnes),
    norme(Places, Poids, Norme),

    /* Recherche par le plus contraint, dans l'ordre croissant */
    minimize(search(Places, 0, most_constrained, indomain_min, complete, []), Norme),
    affiche(Places, Personnes, Norme).
```

Test:

```
                tom  dan   ted   kim   zoe | luc   jim   ron       max   lou
-8    -7    -6    -5    -4    -3    -2    -1  |  1     2     3     4     5     6     7     8

Norme : 802

P = [](3, -1, 2, 6, 1, -4, -3, -5, 5, -2)
Yes (0.23s cpu)
```

Version 2

```
/* Les variables sont triées pour correspondre au problème */
solve_minimize_v2:-
    getData(Poids, Personnes),
    dim(Poids, [Longueur]),
    getVar(Places, Longueur),

    pose_contrainte(Places, Poids, Personnes),
    norme(Places, Poids, Norme),

    minimize(search(Places, 0, input_order, indomain_middle,
complete, []), Norme),
    affiche(Places, Personnes, Norme).
```

Test:

```
Found a solution with cost 1277
Found a solution with cost 1176
Found a solution with cost 1138
Found a solution with cost 1053
Found a solution with cost 972
Found a solution with cost 933
Found a solution with cost 875
Found a solution with cost 852
Found a solution with cost 802
Found no solution with cost 30.0 .. 801.0
ron : 3
zoe : -1
jim : 2
lou : 6
luc : 1
dan : -4
ted : -3
tom : -5
max : 5
kim : -2

                tom  dan   ted   kim   zoe | luc   jim   ron       max   lou
-8    -7    -6    -5    -4    -3    -2    -1  |  1    2    3    4    5    6    7    8
```

Norme : 802

Yes (1.35s cpu)

Version 3

```
1  /* Combinaison des versions 1 et 2 */
2  solve_minimize_v3:-
3      getData(Poids, Personnes),
4      dim(Poids, [Longueur]),
5      getVar(Places, Longueur),
6
7      pose_contrainte(Places, Poids, Personnes),
8      norme(Places, Poids, Norme),
9
10     minimize(search(Places, 0, most_constrained,
   indomain_middle, complete, []), Norme),
11     affiche(Places, Personnes, Norme).
```

Test:

Found a solution with cost 945
Found a solution with cost 802
Found no solution with cost 30.0 .. 801.0
ron : 3
zoe : -1
jim : 2
lou : 6
luc : 1
dan : -4
ted : -3
tom : -5
max : 5
kim : -2

			tom	dan	ted	kim	zoe	luc	jim	ron		max	lou		
-8	-7	-6	-5	-4	-3	-2	-1	1	2	3	4	5	6	7	8

> Norme : 802
>
> Yes (0.14s cpu)

Version 4

```
/* Les variables sont triées selon une heuristique indéfinie our le moment */
solve_minimize_v4:-
    getData(Poids, Personnes),
    dim(Poids, [Longueur]),
    getVar(Places, Longueur),

    pose_contrainte(Places, Poids, Personnes),
    norme(Places, Poids, Norme),

    /* Recherche par le plus contraint, dans l'ordre croissant */
    getVarListOpti(Places, Liste),

    minimize(search(Liste, 0, most_constrained, indomain_middle, complete, []), Norme),
    affiche(Places, Personnes, Norme).

/* Tri des données par ordre décroissant des poids */
getVarListOpti(Places, Liste):-
    PlaceRon is Places[1],
    PlaceZoe is Places[2],
    PlaceJim is Places[3],
    PlaceLou is Places[4],
    PlaceLuc is Places[5],
    PlaceDan is Places[6],
    PlaceTed is Places[7],
    PlaceTom is Places[8],
    PlaceMax is Places[9],
    PlaceKim is Places[10],
    Liste = [PlaceLuc, PlaceTom, PlaceJim, PlaceLou, PlaceZoe, PlaceTed, PlaceRon, PlaceKim, PlaceMax, PlaceDan].
```

Test:

```
Found a solution with cost 945
Found a solution with cost 848
Found a solution with cost 802
Found no solution with cost 30.0 .. 801.0
ron : 3
zoe : -1
jim : 2
lou : 6
luc : 1
dan : -4
ted : -3
tom : -5
max : 5
kim : -2

.                 tom  dan   ted   kim   zoe  |  luc   jim   ron        max   lou
-8    -7    -6    -5   -4    -3    -2    -1   |   1     2     3     4     5     6     7     8

Norme : 802

Yes (0.14s cpu)
```

Question de compréhension

Le labeling original prend les variables dans l'ordre de la liste en prenant les valeurs croissantes. En faisant cette technique, on risque de commencer par des variables avec un grand domaine, ce qui accroît le temps pour trouver une solution. Il n'est donc pas efficace puisque, dans notre cas, on commence avec une variable à grand domaine en partant de faibles valeurs. Ainsi, le temps de résolution est plus important qu'avec notre propre labeling.

Choix des stratégies

Version 2 : Nous avons choisi de chercher les solutions vers le centre des domaines parce qu'il est fort probable qu'une des solutions se trouve vers le milieu.

Version 4 : Nous avons placé les personnes les plus lourdes en premier pour augmenter leur priorité. En effet, une personne plus lourde affecte plus le moment qu'une personne légère donc en procédant de cette manière on minimise les choix pour se placer sur la balançoire. On réduit ainsi le domaine pour obtenir plus rapidement une solution.

Prédicats utilitaires

```
/* Calcule la norme d'un vecteur */
norme(Places, Poids, Norme):-
    absVecteur(Places, AbsVecteur),
    produitScalaire(AbsVecteur, Poids, Scalaire),
    Norme #= Scalaire / 2.

/* Calcule le produit cartésien de deux vecteurs */
produitVecteur(Vect1, Vect2, Res):-
    dim(Vect1, [Longueur]),
    dim(Vect2, [Longueur]),
    dim(Res, [Longueur]),
    (
        for(I, 1, Longueur),
        param(Vect1, Vect2, Res)
    do
        Res[I] #= Vect1[I] * Vect2[I]
    ).

/* Calcule le produit scalaire de deux vecteurs */
produitScalaire(Vect1, Vect2, Res):-
    produitVecteur(Vect1, Vect2, Vect3),
    sommeVecteur(Vect3, Res).

sommeVecteur(Vect, Res):-
    (
        foreachelem(X, Vect),
        fromto(0, AncienneValeur, NouvelleValeur, Res)
    do
        NouvelleValeur #= AncienneValeur + X
    ).

/* Calcule la valeur absolue de chaque élément du vecteur Vect
*/
```

```
33  absVecteur(Vect, Res):-
34      dim(Vect, [Longueur]),
35      dim(Res, [Longueur]),
36      (
37          for(I, 1, Longueur),
38          param(Vect, Res)
39      do
40          X is Vect[I],
41          Res[I] #= abs(X)
42      ).
43
44  /* Trouve l'indice d'un élément dans un vecteur */
45  indiceDe(Element, Vecteur, Indice):-
46      dim(Vecteur, [Longueur]),
47      (
48          for(I, 1, Longueur),
49          fromto(0, AncienneValeur, NouvelleValeur, Indice),
50          param(Element, Vecteur)
51      do
52          X is Vecteur[I],
53          NouvelleValeur #= (Element #= X) * I + AncienneValeur *
    (Element #\= X)
54      ).
55
56  /* Affichage des resultats */
57  affiche(Places, Personnes, Norme):-
58      dim(Places, [Longueur]),
59
60      /* Affiche persone : place */
61      (
62          for(I, 1, Longueur),
63          param(Places, Personnes)
64      do
65          Place is Places[I],
66          Personne is Personnes[I],
67          write(Personne),
68          write(" : "),
69          writeln(Place)
70      ),
71      writeln(""),
72      writeln(""),
```

```
73
74      /* Affiche les noms des personnes au bon endroit sur la
    balancoire */
75      (
76          for(I, -8, 8),
77          param(Places, Personnes)
78      do
79          indiceDe(I, Places, Indice),
80          (I #= 0 ->
81              write("|")
82          ;
83              (Indice #> 0 ->
84                  Personne is Personnes[Indice],
85                  write("  "),
86                  write(Personne),
87                  write("  ")
88              ;
89                  write("       ")
90              )
91          )
92      ),
93      writeln(""),
94
95      /* Affiche la barre de la balancoire */
96      (
97          for(I, -8, 8)
98      do
99          (I #= 0 ->
00              write(" ")
01          ;
02              write(" ----- ")
03          )
04      ),
05      writeln(""),
06
07      /* Affiche la place de la balancoire en dessous */
08      (
09          for(I, -8, 8)
10      do
11          (I #> 0 -> write("   "), write(I), write("   "); true),
12          (I #< 0 -> write("  "), write(I), write("   "); true),
```

```
13          (I #= 0 -> write("|"); true)
14      ),
15      writeln(""),
16      writeln(""),
17      write("Norme : "),
18      writeln(Norme),
19      writeln("").
```

II. Code source

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


?- local domain(noms(ron, zoe, jim, lou, luc, dan, ted, tom, max, kim)).


/* minimize(+Goal, ?Cost, +Options) : A solution of the goal Goal is found that minimizes the value of Cost using branch and bound */
/* search(+L, ++Arg, ++Select, +Choice, ++Method, +Option) : */

/* Q 6.2 */
/* Prédicat de résolution du problème */
solve:-
    getData(Poids, Personnes),
    dim(Poids, [Longueur]),
    getVar(Places, Longueur),
    pose_contrainte(Places, Poids, Personnes),
    norme(Places, Poids, Norme),
    labeling(Places),
    affiche(Places, Personnes, Norme).

/* Prédicats de résolution avec Branch and Bound */
solve_minimize_v0:-
    getData(Poids, Personnes),
    dim(Poids, [Longueur]),
    getVar(Places, Longueur),

    pose_contrainte(Places, Poids, Personnes),
    norme(Places, Poids, Norme),

    /* Recherche par le plus contraint, dans l'ordre croissant */
```

```
33      minimize(labeling(Places), Norme),
34      affiche(Places, Personnes, Norme).
35
36  solve_minimize_v1:-
37      getData(Poids, Personnes),
38      dim(Poids, [Longueur]),
39      getVar(Places, Longueur),
40
41      pose_contrainte(Places, Poids, Personnes),
42      norme(Places, Poids, Norme),
43
44      /* Recherche par le plus contraint, dans l'ordre croissant
    */
45      minimize(search(Places, 0, most_constrained, indomain_min,
    complete, []), Norme),
46      affiche(Places, Personnes, Norme).
47
48  solve_minimize_v2:-
49      getData(Poids, Personnes),
50      dim(Poids, [Longueur]),
51      getVar(Places, Longueur),
52
53      pose_contrainte(Places, Poids, Personnes),
54      norme(Places, Poids, Norme),
55
56      minimize(search(Places, 0, input_order, indomain_middle,
    complete, []), Norme),
57      affiche(Places, Personnes, Norme).
58
59  solve_minimize_v3:-
60      getData(Poids, Personnes),
61      dim(Poids, [Longueur]),
62      getVar(Places, Longueur),
63
64      pose_contrainte(Places, Poids, Personnes),
65      norme(Places, Poids, Norme),
66
67      minimize(search(Places, 0, most_constrained,
    indomain_middle, complete, []), Norme),
68      affiche(Places, Personnes, Norme).
69
```

```
70  solve_minimize_v4:-
71      getData(Poids, Personnes),
72      dim(Poids, [Longueur]),
73      getVar(Places, Longueur),
74
75      pose_contrainte(Places, Poids, Personnes),
76      norme(Places, Poids, Norme),
77
78      /* Recherche par le plus contraint, dans l'ordre croissant
    */
79      getVarListOpti(Places, Liste),
80
81      minimize(search(Liste, 0, most_constrained, indomain_middle,
    complete, []), Norme),
82      affiche(Places, Personnes, Norme).
83
84
85  /* Q 6.1 */
86  /* Pose les contraintes sur les données */
87  getData(Poids, Personnes):-
88      Personnes = [](ron, zoe, jim, lou, luc, dan, ted, tom, max,
    kim),
89      Personnes &:: noms,
90      Poids = [](24, 39, 85, 60, 165, 6, 32, 123, 7, 14).
91
92  /* Pose les contraintes sur les variables */
93  getVar(Places, Longueur):-
94      dim(Places, [Longueur]),
95      Places #:: [-8 .. -1, 1 .. 8].
96
97
98  pose_contrainte(Places, Poids, Personnes):-
99      equilibre(Places, Poids),
00      contraindreLouEtTom(Places, Personnes),
01      contraindreDanEtMax(Places, Personnes),
02      cinqDeChaqueCote(Places),
03      enleveSymetrie(Places, Personnes),
04      ic:alldifferent(Places).
05
06  /* Garantit que la balançire est en équilibre */
07  equilibre(Places, Poids):-
```

```
08      produitScalaire(Places, Poids, Somme),
09      Somme #= 0.
10
11  /* Place Lou et Tom aux extrémités */
12  contraindreLouEtTom(Places, Personnes):-
13      ic:max(Places, Max),
14      ic:min(Places, Min),
15      indicePersonne(lou, Personnes, IndiceLou),
16      indicePersonne(tom, Personnes, IndiceTom),
17      Places[IndiceLou] #>= Max or Places[IndiceLou] #=< Min,
18      Places[IndiceTom] #>= Max or Places[IndiceTom] #=< Min.
19
20  /* Place Dan et Max devant Lou et Tom */
21  contraindreDanEtMax(Places, Personnes):-
22      indicePersonne(dan, Personnes, Dan),
23      indicePersonne(max, Personnes, Max),
24      indicePersonne(lou, Personnes, Lou),
25      indicePersonne(tom, Personnes, Tom),
26      (Places[Lou] > 0) => ((Places[Dan] #= Places[Lou] - 1 and
    Places[Max] #= Places[Tom] + 1) or (Places[Max] #= Places[Lou] -
    1 and Places[Dan] #= Places[Tom] + 1)),
27      (Places[Lou] < 0) => ((Places[Dan] #= Places[Lou] + 1 and
    Places[Max] #= Places[Tom] - 1) or (Places[Max] #= Places[Lou] +
    1 and Places[Dan] #= Places[Tom] - 1)).
28
29  /* Garantit que 5 persones seront de part de d'autre de la
    balançoire */
30  cinqDeChaqueCote(Places):-
31      (
32          foreachelem(Place, Places),
33          fromto(0, AncienCompteGauche, NouveauCompteGauche, 5)
34      do
35          NouveauCompteGauche #= AncienCompteGauche + (Place #< 0)
36      ).
37
38  /* Pour enlever la symetrie, il suffit de choisir un côté pour
    soit Lou, soit Tom vu qu'on sait qu'ils sont de part et d'autres
    de la balancoire */
39  enleveSymetrie(Places, Personnes):-
40      indicePersonne(lou, Personnes, IndiceLou),
```

```
41      Places[IndiceLou] #> 0. /* Choisi de mettre la place de Lou
    à droite */
42
43  /* Calcule la norme d'un vecteur */
44  norme(Places, Poids, Norme):-
45      absVecteur(Places, AbsVecteur),
46      produitScalaire(AbsVecteur, Poids, Scalaire),
47      Norme #= Scalaire / 2.
48
49
50  /* Toujours mettre un else dans un if (cond -> true; false) */
51  /* Recherche l'indice d'une personne */
52  indicePersonne(Personne, Personnes, Indice):-
53      dim(Personnes, [Longueur]),
54      (
55          for(I, 1, Longueur),
56          param(Personne, Personnes, Indice)
57      do
58          X is Personnes[I],
59          (Personne &= X -> Indice #= I; true)
60      ).
61
62
63  getVarList(Places, List):-
64      term_variables(Places, List).
65
66
67  /* Tri des données par ordre décroissant des poids */
68  getVarListOpti(Places, Liste):-
69      PlaceRon is Places[1],
70      PlaceZoe is Places[2],
71      PlaceJim is Places[3],
72      PlaceLou is Places[4],
73      PlaceLuc is Places[5],
74      PlaceDan is Places[6],
75      PlaceTed is Places[7],
76      PlaceTom is Places[8],
77      PlaceMax is Places[9],
78      PlaceKim is Places[10],
79      Liste = [PlaceLuc, PlaceTom, PlaceJim, PlaceLou, PlaceZoe,
    PlaceTed, PlaceRon, PlaceKim, PlaceMax, PlaceDan].
```

```
80
81  /* Affichage des resultats */
82  affiche(Places, Personnes, Norme):-
83      dim(Places, [Longueur]),
84
85      /* Affiche persone : place */
86      (
87          for(I, 1, Longueur),
88          param(Places, Personnes)
89      do
90          Place is Places[I],
91          Personne is Personnes[I],
92          write(Personne),
93          write(" : "),
94          writeln(Place)
95      ),
96      writeln(""),
97      writeln(""),
98
99      /* Affiche les noms des personnes au bon endroit sur la
    balancoire */
00      (
01          for(I, -8, 8),
02          param(Places, Personnes)
03      do
04          indiceDe(I, Places, Indice),
05          (I #= 0 ->
06              write("|")
07          ;
08              (Indice #> 0 ->
09                  Personne is Personnes[Indice],
10                  write("  "),
11                  write(Personne),
12                  write("  ")
13              ;
14                  write("       ")
15              )
16          )
17      ),
18      writeln(""),
19
```

```
20      /* Affiche la barre de la balancoire */
21      (
22          for(I, -8, 8)
23      do
24          (I #= 0 ->
25              write(" ")
26          ;
27              write(" ----- ")
28          )
29      ),
30      writeln(""),
31
32      /* Affiche la place de la balancoire en dessous */
33      (
34          for(I, -8, 8)
35      do
36          (I #> 0 -> write("   "), write(I), write("   "); true),
37          (I #< 0 -> write("  "), write(I), write("   "); true),
38          (I #= 0 -> write("|"); true)
39      ),
40      writeln(""),
41      writeln(""),
42      write("Norme : "),
43      writeln(Norme),
44      writeln("").
45
46
47  /* Q 6.3 */
48
49
50  /****************** FONCTIONS UTILITAIRES ******************/
51
52  /* Calcule le produit cartésien de deux vecteurs */
53  produitVecteur(Vect1, Vect2, Res):-
54      dim(Vect1, [Longueur]),
55      dim(Vect2, [Longueur]),
56      dim(Res, [Longueur]),
57      (
58          for(I, 1, Longueur),
59          param(Vect1, Vect2, Res)
60      do
```

```
        Res[I] #= Vect1[I] * Vect2[I]
    ).

/* Calcule le produit scalaire de deux vecteurs */
produitScalaire(Vect1, Vect2, Res):-
    produitVecteur(Vect1, Vect2, Vect3),
    sommeVecteur(Vect3, Res).

sommeVecteur(Vect, Res):-
    (
        foreachelem(X, Vect),
        fromto(0, AncienneValeur, NouvelleValeur, Res)
    do
        NouvelleValeur #= AncienneValeur + X
    ).

/* Calcule la valeur absolue de chaque élément du vecteur Vect
*/
absVecteur(Vect, Res):-
    dim(Vect, [Longueur]),
    dim(Res, [Longueur]),
    (
        for(I, 1, Longueur),
        param(Vect, Res)
    do
        X is Vect[I],
        Res[I] #= abs(X)
    ).

/* Trouve l'indice d'un élément dans un vecteur */
indiceDe(Element, Vecteur, Indice):-
    dim(Vecteur, [Longueur]),
    (
        for(I, 1, Longueur),
        fromto(0, AncienneValeur, NouvelleValeur, Indice),
        param(Element, Vecteur)
    do
        X is Vecteur[I],
        NouvelleValeur #= (Element #= X) * I + AncienneValeur *
(Element #\= X)
    ).
```