TP1 - Découverte de la bibliothèque de contraintes à domaines finis

Paul Chaignon - Ulysse Goarant

6 février 2014

1 De Prolog à Prolog+ic

Listing 1 – prolog-ic.ecl

```
1  :- lib(ic).
2
3  voiture(rouge).
4  voiture(vert(clair)).
5  voiture(gris).
6  voiture(blanc).
7  bateau(vert).
8  bateau(blanc).
9  bateau(noir).
10
11 /**
12  * Question 1.1
13  * choixCouleur(?CouleurBateau, ?CouleurVoiture)
14  */
15 choixCouleur(Couleur, Couleur):-
16   voiture(Couleur),
17   bateau(Couleur).
18
19
20 minResistance(5000).
21 maxResistance(10000).
22 minCondensateur(9000).
23 maxCondensateur(20000).
24
25 /**
26  * Question 1.3
27  * isBetween(?Var, +Min, -Max)
28  */
29 isBetween(Var, Min, Max):-
30   not(free(Var)),
31   Var >= Min,
32   Var =< Max.
33 isBetween(Var, Min, Max):-
34   free(Var),
35   isBetweenIncremental(Var, Min, Max).
36
37 /**
38  * isBetweenIncremental(-Var, +Min, +Max)
```

```
 */
isBetweenIncremental(Min, Min, Max).
isBetweenIncremental(Var, Min, Max):-
  NextMin is Min + 1,
  NextMin =< Max,
  isBetweenIncremental(Var, NextMin, Max).

/**
 * Question 1.4
 * commande(-NbResistances, -NbCondensateurs)
 */
commande(NbResistances, NbCondensateurs):-
  isBetween(NbResistances, 5000, 10000),
  isBetween(NbCondensateurs, 9000, 20000),
  NbResistances > NbCondensateurs.

/**
 * Question 1.7
 * commandeIC(-NbResistances, -NbCondensateurs)
 */
commandeIC(NbResistances, NbCondensateurs):-
  NbResistances #:: 5000..10000,
  NbCondensateurs #:: 9000..20000,
  NbResistances #> NbCondensateurs.

/**
 * Question 1.8
 * commandeLabeling(-NbResistances, -NbCondensateurs)
 */
commandeLabeling(NbResistances, NbCondensateurs):-
  commandeIC(NbResistances, NbCondensateurs),
  labeling([NbResistances, NbCondensateurs]).


/**
 * Tests
 *//*
choixCouleur(blanc, blanc). => Yes
choixCouleur(noir, vert(clair)). => No
choixCouleur(vert, vert(clair)). => No
choixCouleur(CouleurBateau, CouleurVoiture).
  CouleurBateau = blanc
  CouleurVoiture = blanc
  Yes (0.00s cpu)

isBetween(4000000, 1000000, 8000000). => Yes
isBetween(10000000, 1000000, 8000000). => No
isBetween(X, 1, 3).
  X = 1
  Yes (0.00s cpu, solution 1, maybe more)
  X = 2
  Yes (0.00s cpu, solution 2, maybe more)
  X = 3
  Yes (0.00s cpu, solution 3, maybe more)
  No (0.00s cpu)

findall((NbResistances, NbCondensateurs), commande(NbResistances,
    NbCondensateurs), Results), length(Results, NbResults). => 500500
```

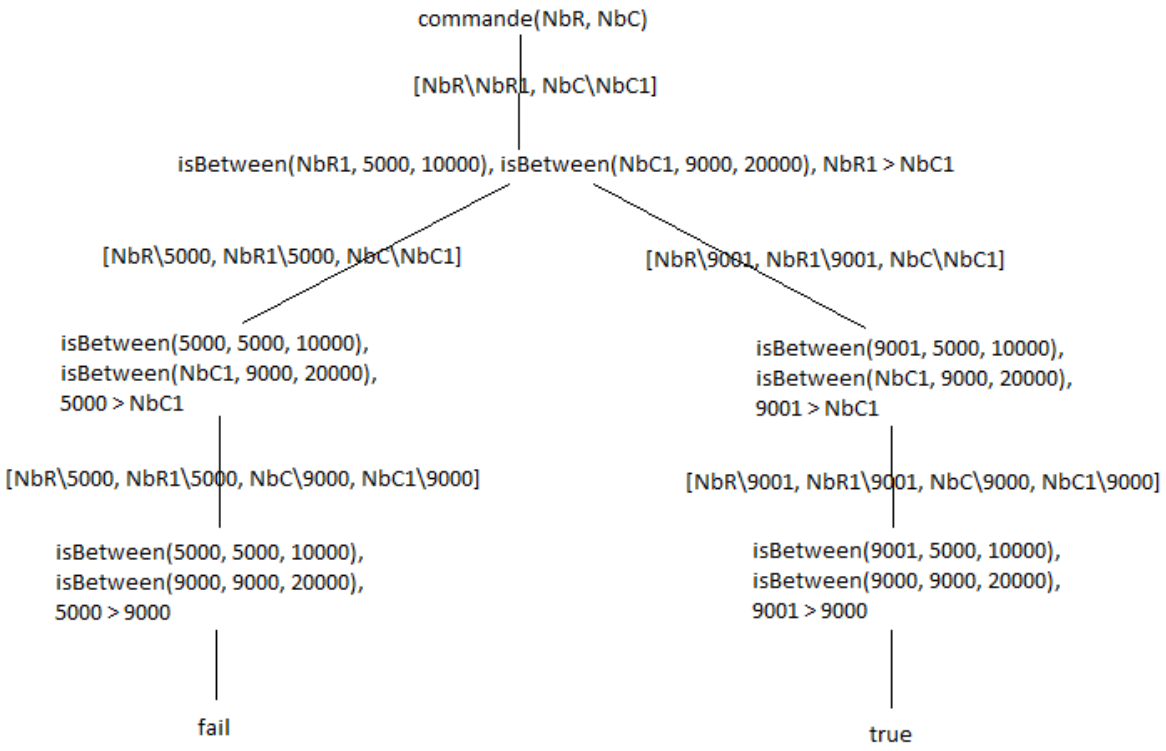

FIGURE 1 – Arbre de recherche de l'exécution du but *commande(-, -)*

```
96 commandeIC(NbResistances, NbCondensateurs).
97   NbResistances = NbResistances{9001 .. 10000}
98   NbCondensateurs = NbCondensateurs{9000 .. 9999}
99   Delayed goals:
100    -(NbCondensateurs{9000 .. 9999}) + NbResistances{9001 .. 10000} #> 0
101 findall((NbResistances, NbCondensateurs),
    commandeLabeling(NbResistances, NbCondensateurs), Results),
    length(Results, NbResults). => 500500*/
```

Question 1.2

Pourquoi Prolog peut être considéré comme un solveur de contraintes sur le domaine des arbres ? Prolog peut être considéré comme un solveur de contraintes sur le domaine des arbres parce qu'il réalise de l'unification et dispose d'un système de Backtracking.

Question 1.5

La figure 1 présente l'arbre de recherche Prolog lors de l'exécution du but *commande(-NbR, -NbC)*.

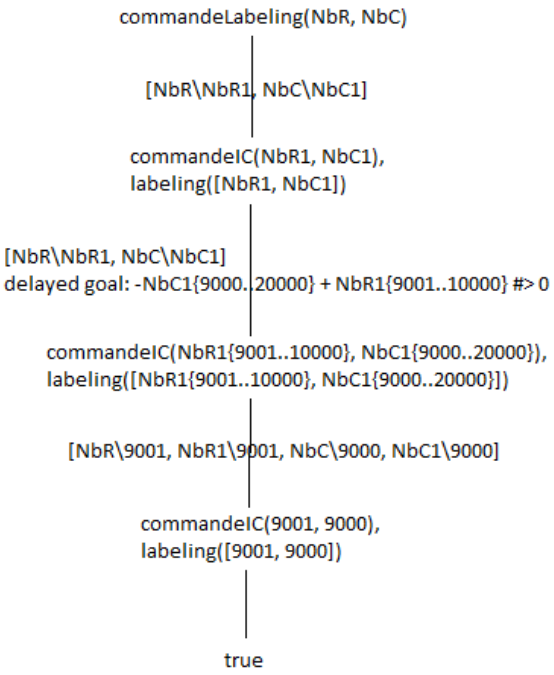

FIGURE 2 – Arbre de recherche de l'exécution du but *commandeLabeling(-, -)*

Question 1.6

Pourquoi peut-on dire que Prolog ne comprend pas les Maths ? Si on pose le prédicat > avant les appels à *isBetween* nous obtenons une erreur d'instanciation. En effet, ECLiPSe ne peut comparer deux nombres qui ne sont pas instanciés.

Nous pouvons dire que Prolog ne comprend pas les maths car il se contente finalement de générer, puis de vérifier, toutes les solutions possibles. Il n'est donc pas capable de résoudre un ensemble d'équation comme nous le faisons manuellement.

Question 1.7

Une contrainte est insatisfaisable parce que les contraintes primitives n'ont pas de valeurs fixées. Le problème peut être résolu en utilisant du *labeling*.

Question 1.8

La figure 2 présente l'arbre de recherche Prolog lors de l'exécution du but *commandeLabeling(-NbR, -NbC)* qui utilise cette fois le solveur *ic* avec du *labeling*.

2 Zoologie

Listing 2 – zoologie.ecl

```
1 :- lib(ic).
2
3 /**
4  * chapieBug(-Chats, -Pies, -Pattes, -Tetes)
```

```
 * chapie tourne dans le vide a partir de la seconde solution pour la question 1.10.
 */
chapieBug(Chats, Pies, Pattes, Tetes):-
  Chats #:: 0..inf,
  Pies #:: 0..inf,
  Pattes #:: 0..inf,
  Tetes #:: 0..inf,
  Pattes #= Chats * 4 + Pies * 2,
  Tetes #= Chats + Pies.

/**
 * chapie(-Chats, -Pies, -Pattes, -Tetes)
 */
chapie(Chats, Pies, Pattes, Tetes):-
  Chats #:: 0..1000,
  Pies #:: 0..1000,
  Pattes #:: 0..1000,
  Tetes #:: 0..1000,
  Pattes #= Chats * 4 + Pies * 2,
  Tetes #= Chats + Pies.

/**
 * Question 1.9
 */
chapie(2, Pies, Pattes, 5).
  Pies = 3
  Pattes = 14
  Yes (0.00s cpu)

/**
 * Question 1.10
 */
chapie(Chats, Pies, Pattes, Tetes), Pattes #= 3 * Tetes.
  Chats = Chats{0 .. 249}
  Pies = Pies{0 .. 333}
  Pattes = Pattes{0 .. 999}
  Tetes = Tetes{0 .. 333}
  Delayed goals:
          Pattes{0 .. 999} - 2 * Pies{0 .. 333} - 4 * Chats{0 .. 249} #= 0
          Tetes{0 .. 333} - Pies{0 .. 333} - Chats{0 .. 249} #= 0
          -3 * Tetes{0 .. 333} + Pattes{0 .. 999} #= 0
  Yes (0.00s cpu)

chapie(Chats, Pies, Pattes, Tetes), Pattes #= 3 * Tetes, labeling([Chats, Pies, Pattes, Tetes]).
  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)
  ... (167 solutions avec des limites Ã  1000)
```

Question 1.9

Il faut 3 pies et 14 pattes pour totaliser 5 têtes et 2 chats.

Question 1.10

Nous trouvons en posant les équations qu'il faut un même nombre de chats et pies pour avoir 3 fois plus de pattes que de têtes. Cependant, comme on peut le voir lorsqu'on n'effectue pas de *labeling*, le solveur de la bibliothèque *ic* ne semble pas capable d'effectuer cette réduction.

3 Le OU en contraintes

Listing 3 – ou-constraint.ecl

```
:- lib(ic).

/**
 * Question 1.11
 * vabs(?Val, ?AbsVal)
 */
vabs(Val, AbsVal):-
  AbsVal #> 0,
  (
    Val #= AbsVal
  ;
    Val #= -AbsVal
  ),
  labeling([Val, AbsVal]).

/**
 * vabsIC(?Val, ?AbsVal)
 */
vabsIC(Val, AbsVal):-
  AbsVal #> 0,
  Val #= AbsVal or Val #= -AbsVal,
  labeling([Val, AbsVal]).

/**
 * Question 1.12
 */
X #:: -10..10, vabs(X, Y).
  X = 1
  Y = 1
  Yes (0.01s cpu, solution 1, maybe more)
  X = 2
  Y = 2
  Yes (0.01s cpu, solution 2, maybe more)
  X = 3
  Y = 3
  Yes (0.01s cpu, solution 3, maybe more)
  ...
X #:: -10..10, vabsIC(X, Y).
  X = -10
  Y = 10
  Yes (0.00s cpu, solution 1, maybe more)
```

```
  X = -9
  Y = 9
  Yes (0.00s cpu, solution 2, maybe more)
  X = -8
  Y = 8
  Yes (0.00s cpu, solution 3, maybe more)
  ...

/**
 * Question 1.13
 * faitListeV1(?ListVar, ?Taille, +Min, +Max)
 * faitListeV1(ListVar, 2, 1, 3)
 */
faitListeV1([], 0, _, _).
faitListeV1([First|Rest], Taille, Min, Max):-
  First #:: Min..Max,
  Taille #> 0,
  Taille1 #= Taille - 1,
  faitListeV1(Rest, Taille1, Min, Max).

/**
 * faitListe(?ListVar, ?Taille, +Min, +Max)
 */
faitListe(ListVar, Taille, Min, Max):-
  length(ListVar, Taille),
  ListVar #:: Min..Max.

/**
 * Question 1.14
 * suite(?ListVar)
 */
suite([Xi, Xi1, Xi2]):-
  checkRelation(Xi, Xi1, Xi2).
suite([Xi, Xi1, Xi2|Rest]):-
  checkRelation(Xi, Xi1, Xi2),
  suite([Xi1, Xi2|Rest]).

/**
 * checkRelation(?Xi, ?Xi1, ?Xi2)
 */
checkRelation(Xi, Xi1, Xi2):-
  vabs(Xi1, VabsXi1),
  Xi2 #= VabsXi1 - Xi.

/**
 * Question 1.15
 * checkPeriode(+ListVar).
 */
faitListe(ListVar, 11, -1000, 1000), suite(ListVar), ListVar = [X1, X2,
    X3, X4, X5, X6, X7, X8, X9, X10, X11|Rest], X1 #\= X10 or X2 #\=
    X11, labeling(ListVar).
  No (133.20s cpu)


/**
 * Tests
 */
vabs(5, 5). => Yes
```

```
vabs(5, -5). => No
vabs(-5, 5). => Yes
vabs(X, 5).
vabs(X, AbsX).
vabsIC(5, 5). => Yes
vabsIC(5, -5). => No
vabsIC(-5, 5). => Yes
vabsIC(X, 5).
vabsIC(X, AbsX).

faitListe(ListVar, 5, 1, 3).
  ListVar = [_236{1 .. 3}, _254{1 .. 3}, _272{1 .. 3}, _290{1 .. 3},
     _308{1 .. 3}]
  Yes (0.00s cpu)
faitListe(ListVar, _, 1, 3).
  ListVar = []
  Yes (0.00s cpu, solution 1, maybe more)
  ListVar = [_217{1 .. 3}]
  Yes (0.00s cpu, solution 2, maybe more)
  ListVar = [_222{1 .. 3}, _240{1 .. 3}]
  Yes (0.00s cpu, solution 3, maybe more)
  ...
faitListe([_, _, _, _, _], Taille, 1, 3).
  Taille = 5
  Yes (0.00s cpu)
faitListe(ListVar, 18, -9, 9), suite(ListVar).
  ListVar = [-2, 1, 3, 2, -1, -1, 2, 3, 1, -2, 1, 3, 2, -1, -1, 2, 3, 1]
  Yes (0.00s cpu, solution 1, maybe more)
  ListVar = [-3, 1, 4, 3, -1, -2, 3, 5, 2, -3, 1, 4, 3, -1, -2, 3, 5, 2]
  Yes (0.00s cpu, solution 2, maybe more)
  ListVar = [-4, 1, 5, 4, -1, -3, 4, 7, 3, -4, 1, 5, 4, -1, -3, 4, 7, 3]
  Yes (0.00s cpu, solution 3, maybe more)
  ... (99 solutions)
```

Question 1.12

Le calcul de la valeur absolue avec utilisation de l'opérateur de disjonction *or* de *ic* semble s'exécuter beaucoup plus rapidement que l'autre version. Cela semble être dû au fait qu'une plus grande utilisation de contraintes permet une meilleure réduction.

Nous pouvons aussi remarquer que les résultats n'apparaissent pas dans le même ordre malgré que l'ordre des contraintes soit le même.