PLC – Cours 8

Thi-Bich-Hanh Dao

Université d'Orléans

M1 Informatique

Plan

Programmation avec contraintes en Gnu-Prolog

1. Variables de domaines finis
2. Contraintes arithmétiques
3. Contraintes booléennes et contraintes réifiées
4. Contraintes symboliques
5. Labelling
6. Contraintes d'optimisation

Variables sur les domaines finis

Une variable de domaines

- Toute variable dans des contraintes sont des entiers entre 0 et une valeur maximum (donnée par le prédicat `fd_max_integer/1`)
- Le domaine d'une variable de domaine fini (variable FD) est réduit (filtré) petit à petit par les contraintes posée sur la variable. Une fois une valeur est supprimée du domaine d'une variable X, elle ne pourra plus être réintégrée dans le domaine de X dans la suite de la résolution.
- Deux représentations de domaine de variable : représentation par intervalle et représentation "clairsemée".

Variables sur les domaines finis

Déclaration de domaine par intervalle

- `fd_domain/3` contrainte le domaine d'une variable ou une liste de variables, à être l'ensemble compris entre deux valeurs données
- Schéma d'appel :
  `fd_domain(+liste_de_variables_FD, +entier_min, +entier_max)`
  `fd_domain(?variables_FD, +entier_min, +entier_max)`
- Exemples :
  `fd_domain([X1,X2,X3,X4],1,4)`
  `fd_domain(X1,1,4)`

Variables sur les domaines finis

Déclaration de domaine par liste "clairsemée"

- `fd_domain/2` contrainte une variable ou une liste de variables, à prendre des valeurs dans une liste d'entiers donnée
- Schéma d'appel :
  `fd_domain(+liste_de_variables_FD, +liste_d_entiers)`
  `fd_domain( ?variables_FD, +liste_d_entiers)`
- Exemples :
  `fd_domain([X1,X2],[2,4,6])`
  `fd_domain(X1,[2,4,6])`
- La longueur de la liste d'entiers est limitée par `vector_max`

Contraintes arithmétiques

- On peut poser des contraintes d'égalité ($=$), de différence ($\neq$) ou d'inégalité ($<, <=, >, >=$) sur des expressions arithmétiques sur les domaines finis.
- Une expression arithmétique sur les domaines finis est écrite avec des opérateurs classiques ($+, -, *, /, \ldots$) dont les variables sont sur les domaines finis.
- Deux façons de déclarer des contraintes arithmétiques :
  - Si l'on veut que le solveur établisse sur les contraintes la consistance de borne, alors les contraintes sont déclarées par :
    `#=, #\=, #<, #=<, #>, #>=`
  - Si l'on veut que le solveur établisse sur les contraintes la consistance d'arc :
    `#=#, #\=#, #<#, #=<#, #>#, #>=#`

Contraintes arithmétiques

Exemples

```
| ?- fd_domain(X,1,8), fd_domain(Y,2,7), X#= 2*Y.

X = _#3(4..8)
Y = _#25(2..4)

yes
| ?- fd_domain(X,1,8), fd_domain(Y,2,7), X#=# 2*Y.

X = _#3(4:6:8)
Y = _#25(2..4)

yes
```


Distinction entre =, is et #=#

```
| ?- fd_domain(X,1,5), Y = 2+X.

X = _#3(1..5)
Y = 2+_#3(1..5)

(1 ms) yes
| ?- fd_domain(X,1,5), Y is 2+X.
uncaught exception: error(type_error(evaluable,
                                     _#787741(1..5)),(is)/2)
| ?- fd_domain(X,1,5), Y #=# 2+X.

X = _#3(1..5)
Y = _#25(3..7)

yes
```

Exemples de consistances

```
| ?- fd_domain(X,1,10), fd_domain(Y,[1,3,5,7]), X #=# Y+1.

X = _#3(2:4:6:8)
Y = _#24(1:3:5:7)

| ?- fd_domain(X,1,10), fd_domain(Y,[1,3,5,7]), X #= Y+1.

X = _#3(2..8)
Y = _#24(1:3:5:7)

| ?- fd_domain([X,Y,Z],1,10), X #< Y, Y #=< Z, Z #=< 2.

X = 1
Y = 2
Z = 2
```


Les chats et les oiseaux

```
| ?- co_iso(2,3,T,P).
P = 14
T = 5

| ?- co_iso(C,V,5,14).
uncaught exception: error(instantiation_error,(is)/2)

| ?- chatoiseau_ac(2,3,T,P).
P = 14
T = 5

| ?- chatoiseau_ac(C,V,5,14).
C = 2
V = 3
```


Contraintes booléennes et contraintes réifiées

Contraintes booléennes

- Déclaration de variables booléennes : domaine `[0,1]` ou `fd_domain_bool/1`
- Une contrainte booléenne sur les domaines finis est composée par des opérateurs booléens suivants :
  - l'équivalence : `E1 #<=> E2`
  - la non équivalence : `E1 #\<=> E2`
  - l'implication : `E1 #==> E2`
  - le non logique : `#\E1`
  - le ou logique : `E1 #\/ E2`
  - le ou exclusif : `E1 ## E2`
  - le et logique : `E1 #/\E2`

Remarque

```
| ?- X #= 3, X #= 3 #==> X #\= 4.

X = 3
yes
| ?- X #= 3, X #= 3 #==> X #= 4.

no
| ?- X #= 3, X #= 3 #<=> X #= 4.

no
| ?- X #= 2, X #= 3 #<=> X #= 4.

X = 2
```

Exemples

- Si Alice et Marianne chantent toutes les deux, alors les voisins se plaignent.
- Si Paul fait de la musique et Alice ne chante pas, tout le monde danse.
- Si tout le monde danse, les voisins se plaignent.
- Si Paul ne fait pas de musique, alors il fait beau.
- S'il ne fait pas beau, Marianne chante.
- Il ne fait pas beau.

On doit pouvoir en déduire quelque chose sur les voisins : à votre avis, se plaignent-ils ?

Modélisation

```
| ?- A #/\ M #<=> V, P#/\ #\A #==> D, D #==> V, #\P #==> B,   #\B #==> M, #\B.
A = _#19(0..1)
B = 0
D = _#143(0..1)
M = 1
P = 1
V = _#0(0..1)

| ?- A #/\ M #<=> V, P#/\ #\A #==> D, D #==> V, #\P #==> B,  #\B #==> M, #\B,
     fd_labeling([A,D]).
A = 1
B = 0
D = 0
M = 1
P = 1
V = 1 ? ;

A = 1
B = 0
D = 1
M = 1
P = 1
V = 1
```


Contraintes symboliques

`fd_all_different/1`

- Contrainte les variables d'une liste à prendre des valeurs deux à deux différentes :

  `fd_all_different(+liste_de_vars)`
- Exemples :

  `fd_all_different([X1,X2,X3,X4])`
- Lorsque l'une des variables soit instanciée par une valeur, cette valeur est supprimée des domaines des autres variables.
- Contrainte se basant sur l'arc-consistance, n'assure pas la constance globale (ce n'est pas une contrainte globale)

Exemple

```
| ?-fd_domain([X,Y,Z],0,1), fd_all_different([X,Y,Z]).
X = _#154(0..1)
Y = _#173(0..1)
Z = _#192(0..1)

| ?-fd_domain([X,Y,Z],0,1), fd_all_different([X,Y,Z]),
     fd_labeling([X,Y,Z]).

no
| ?-fd_domain([X,Y,Z],0,1),fd_all_different([X,Y,Z]),X#=0.

no

| ?-fd_domain([X,Y,Z],0,1),fd_all_different([X,Y,Z]),X#=1.

no
```

Contraintes symboliques

Quelques autre prédicats

- `fd_element(?I,+L,?X)`,
  `fd_element_var(?I,+L,?X)` :
  contraignent X à être le I-ème élément de la liste de valeurs ou de variables L
- `fd_atmost(+N,+L,+V)`,
  `fd_atleast(+N,+L,+V)`,
  `fd_exactly(+N,+L,+V)` :
  exprime qu'au plus, qu'au moins ou qu'exactement N variables de la liste L prennent la valeur V.

Résolution de CSP en Gnu-Prolog

- Après avoir défini un CSP
  - en déclarant les variables,
  - en posant les contraintes sur des variables

  on peut demander Gnu-Prolog à le résoudre.
- Gnu-Prolog résoud un CSP
  - en énumérant les différentes affectations possibles de valeurs aux variables jusqu'à en trouver des solutions,
  - tout en utilisant les contraintes pour réduire des domaines de variables.
- Pour lancer une énumération : `fd_labeling`

Résolution de CSP en Gnu-Prolog

`fd_labeling`

`fd_labeling(+V)` ou `fd_labeling(+LV)` :

- affecte une valeur à la variable V ou à chaque variable de la liste LV de sorte que toutes les contraintes portant sur ces variables soient satisfaites.
- Lorsque Prolog "backtrack" sur ce prédicat, il cherche à chaque fois une solution différente.

Exemple

```
| ?- fd_domain([X,Y,Z],1,5), X#>2*Y, Z#=X+1.

X = _#3(3..4)
Y = 1
Z = _#47(4..5)

| ?- fd_domain([X,Y,Z],1,5), X#>2*Y, Z#=X+1, fd_labeling([X]).

X = 3
Y = 1
Z = 4 ? ;

X = 4
Y = 1
Z = 5
```

Contraintes d'optimisation

- `fd_minimize(Goal,X)` : exécute `Goal` d'une façon répétitive pour trouver une valeur qui minimise la variable `X`
- A chaque fois qu'une valeur V pour X est trouvée, `Goal` est ré-exécuté avec une nouvelle contrainte `X#<V`. La dernière valeur trouvée est la valeur optimale.
- `fd_maximize(Goal,X)` : ici la valeur de X est maximisée

Contraintes d'optimisation

Exemple

```
| ?- fd_domain([W,P,C],0,9),
     4*W + 3*P + 2*C #=< 9,
     15*W + 10*P + 7*C #= Max,
     fd_labeling([W,P,C,Max]).

...
| ?- fd_domain([W,P,C],0,9),
     4*W + 3*P + 2*C #=< 9,
     15*W + 10*P + 7*C #= Max,
     fd_maximize(fd_labeling([W,P,C,Max]),Max).

W = 1
P = 1
C = 1
Max = 32
```


Suite magique

- Une suite magique $x_0, x_1, \ldots, x_{n-1}$ est telle que x_i est le nombre d'occurrences de i dans la suite
- Exemples :
  - avec $n = 4$, deux suites magiques $[2, 0, 2, 0]$ et $[1, 2, 1, 0]$
  - avec $n = 7$, une suite magique $[3, 2, 1, 1, 0, 0, 0]$
- Formulation des conditions : pour tout i de 0 à $n - 1$

$$x_i = \sum_{j=0}^{n-1} b(i,j) \quad \text{avec} \quad b(i,j) = \begin{cases} 1 & \text{si } x_j = i \\ 0 & \text{sinon} \end{cases}$$

Suite magique : modélisation

- Variables :
  - une liste L de n variables : $L = [X_0, \ldots, X_{n-1}]$
  - domaine de chaque variables $X_i \in [0, n]$
- Contraintes : pour chaque $i \in [0, n-1]$
  - pour chaque $j \in [0, n-1]$

$$B_{ij} \Leftrightarrow X_j = i$$

  - X_i est le nombre d'occurrences de i

$$X_i = \sum_{j=0}^{n-1} B_{ij}$$

- Programme en Gnu-Prolog

Suite magique : contraintes redondantes

- Contraintes redondantes pour plus de réduction de domaine

$$n = \sum_{i=0}^{n-1} x_i$$

et

$$n = \sum_{i=0}^{n-1} i * x_i$$

- Temps de calcul pour $n = 100$:
  - 346785ms (sans contraintes redondantes)
  - 163ms (avec contraintes redondantes)

Un autre exemple

Trois amis aiment différents types de musique. A partir des indices suivants, pouvez-vous les identifier, dire leur musique préférée et dire qui a 26 ans ?

1. Rob est plus âgé que Queen, qui aime la musique classique.
2. Celui qui aime le pop n'est pas Prince et n'a pas 24 ans.
3. Leon, qui n'est pas King, a 25 ans.
4. Mark ne préfère pas le jazz.

Modélisation

- Variables : *Rob*, *Leon*, *Mark*, *Queen*, *King*, *Prince*, *Classique*, *Pop*, *Jazz*
- Domaine : $\{24, 25, 26\}$
- Contraintes :
  - Rob, Leon et Mark sont de différents âges

$$alldifferent(Rob, Leon, Mark)$$

  - Queen, King et Prince sont des personnes différentes (donc de différents âges)

$$alldifferent(Queen, King, Prince)$$

  - Classique, Pop, Jazz sont aimés par des personnes différentes (donc de différents âges)

$$alldifferent(Classique, Pop, Jazz)$$

Modélisation : autres contraintes

- Rob est plus âgé que Queen, qui aime la musique classique.

$$Rob > Queen$$
$$Queen = Classique$$

- Celui qui aime le pop n'est pas Prince et n'a pas 24 ans.

$$Pop \neq Prince$$
$$Pop \neq 24$$

- Leon, qui n'est pas King, a 25 ans.

$$Leon \neq King$$
$$Leon = 25$$

- Mark ne préfère pas le jazz.

$$Mark \neq Jazz$$