8.2. **Notions sur la programmation avec contraintes** *(VIDEO-PPT 4.3)*

Le but de cette section est d'aborder succinctement quelques notions de la programmation avec contraintes en extension de la programmation logique. Il s'agit d'un domaine en soi portant une méthodologie propre, les « *Constraint Satisfactory Problems* » (CSP). Cette section se bornera à montrer l'intérêt des contraintes au regard des problèmes traités dans la section précédente avec l'approche générer-tester.

Définition : Une *contrainte* est une relation entre des variables, chacune prenant ses valeurs dans un domaine donné. Elle restreint donc les valeurs possibles que les variables peuvent prendre.

Dans la résolution d'un problème tant que les contraintes sur les variables ne peuvent pas être calculées (parce que des variables ne sont pas instanciées) ou peuvent être calculées **et satisfaites**, la résolution du problème se poursuit. Par contre, dès qu'une contrainte est insatisfaite, la résolution échoue.

Les contraintes du résolveur de Gprolog portent sur des domaines finis d'entiers. Elles se caractérisent par le signe `# : #=, #\=, #<, #=<, #>, #>=`.

Le prédicat prédéfini `fd_domain(Lvars, Binf, Bsup)` permet de poser le système de contraintes tel que toutes les valeurs possibles des variables de `Lvars` soient comprises entre `Binf` et `Bsup`.

Le prédicat `fd_labeling(Lvars)` affecte (génère) des valeurs possibles à chaque variable de `Lvars` compte tenu de leur domaine.

- *Exemple 1* : Une équation

```
| ?- fd_domain([X, Y], 1, 5), X + Y #= 8 , fd_labeling([X, Y]).

X = 3
Y = 5 ? a

X = 4
Y = 4

X = 5
Y = 3
```

☞ *La programmation par essais successifs intégrant des contraintes amène à un schéma du type « contraindre-générer » plus efficace que la méthode « générer-tester ». En effet, les contraintes sur les variables du problème devant être satisfaites sont d'abord posées puis les valeurs possibles sur ces variables sont générées mais dès qu'une des contraintes est non satisfaite, le problème échoue. Dans l'approche générer-tester, les tests ne sont faits que lorsque toutes les variables ont été instanciées.*

- *Exemple 2* : Coloriage d'une carte

Colorier la carte proposée en exemple de la section précédente avec 4 couleurs et rester dans le domaine des entiers.

En programmation par contraintes le problème s'écrit en codant chaque couleur par un entier entre 1 et 4:

```
1. coloriage(L) :- L = [X1, X2, X3, X4, X5], fd_domain(L, 1, 4) ,
                                    X1 #\= X2, X1 #\= X4, X1 #\= X5,
                                    X2 #\= X3, X2 #\= X5, X2 #\= X6,
                                    X3 #\= X6,
                                    X4 #\= X5,
                                    X5 #\= X6,
                                    fd_labeling(L).
```

➢

➢ *Exemple 3 :* Cryptarithme

Il s'agit de résoudre l'équation *SEND + MORE = MONEY* où chaque caractère est associé à un chiffre (entier entre 0 et 9) et où tous les caractères sont associés à des valeurs différentes.

```
1. cryptarithme(L) :- L = [S, E, N, D, M, O, R, Y],  fd_domain(L, 0, 9),
                fd_all_different(L), contraintesEquations(L),
                fd_labeling(L).
2. contraintesEquations([S,E, N, D, M, O, R, Y]):-
                Send #= 1000*S + 100*E + 10 *N + D ,
                More #= 1000*M + 100*O + 10*R + E,
                Money #= 10000*M + 1000*O + 100*N + 10*E + Y,
                Money #= Send + More.
```

Le prédicat prédéfini `fd_all_différent(Lvars)` contraint toutes les variables de la liste `Lvars` à prendre des valeurs différentes.

MiniTP : un coloriage a 3 couleurs !

Soit la carte suivante décrivant les frontières entre quatre villes (V1, ..., V4). On voudrait colorier la carte en utilisant seulement les couleurs rouge, bleu et vert, de sorte que :

- *V1 soit en rouge ou en vert*
- *V2 et V3 soient en bleu ou en vert*
- *V4 soit en vert.*