

# Programmation Logique et Par Contraintes Avancée

## Cours 5 – Contraintes de domaine fini en Oz

Ralf Treinen

Université Paris Diderot  
UFR Informatique  
Laboratoire Preuves, Programmes et Systèmes  
treinen@pps.jussieu.fr

## Contents

Introduction : Domaines, contraintes et propagateurs

Définir les domaines des variables

Quelques propagateurs en Oz

## Contraintes sur un domaine fini

- ▶ On a pour toute variable un *choix fini* de ses valeurs.
- ▶ Utilisée pour modéliser des problèmes avec des choix.
- ▶ Ordonnement, emploi du temps, routage, etc.
- ▶ Beaucoup d'applications industrielles.

## Exemple

Trouver les côtés d'un rectangle à dimensions entières tel que

- ▶ surface = 24 unités
- ▶ périmètre = 20 unités

## Premières observations

- ▶ Toutes les valeurs sont des entiers.
- ▶ Il y a deux variables à déterminer (largeur  $X$  et hauteur  $Y$ )
- ▶ Pour chacune des deux variables il y a un choix fini entre 1 et 9 (pourquoi?)
- ▶ On peut supposer que  $X \leq Y$ . (pourquoi?)  
Il s'agit d'une *symétrie* du problème

## Domaines

- ▶ Un *domaine*  $D$  est une fonction partielle qui associe à une variable un ensemble fini (éventuellement vide) d'entiers.  $vars(D) =$  l'ensemble des variables pour lesquelles  $D$  est défini.
- ▶ Sur l'exemple : on a un domaine initial (définie par l'énoncé du problème) qui associe à  $X$  l'ensemble  $\{1, \dots, 9\}$ , et à  $Y$  le même ensemble  $\{1, \dots, 9\}$ . Ici on a donc même des intervalles, mais ce n'est pas nécessairement le cas.
- ▶ Notation Oz :  $X :: 1\#9$   $Y :: 1\#9$

## Domaines (suite)

- ▶ Un domaine  $D$  a *échoué* (is failed) : il existe une variable  $X$  telle que  $D(X) = \emptyset$ .
- ▶ Un domaine  $D$  fixe une variable  $X$  si  $card(D(X)) = 1$ .
- ▶ Un domaine  $D_1$  est *plus fort* qu'un domaine  $D_2$  si  $vars(D_1) = vars(D_2)$ , et  $D_1(X) \subseteq D_2(X)$  pour tout  $X \in vars(D_1)$ .
- ▶ Le but du jeu est de réduire le domaine, afin d'obtenir soit un domaine échoué (il n'y a pas de solution), ou d'obtenir un domaine qui fixe toutes les variables du problème (solution trouvée).

## Contraintes

- ▶ Une *contrainte* est une formule logique (équation, inégalité, diséquation, etc.).
- ▶ La contrainte est la formulation *mathématique* du problème qu'on souhaite résoudre.
- ▶ Sur l'exemple :  $X + Y = 10 \wedge X * Y = 24$
- ▶ On peut parfois ajouter des contraintes supplémentaires afin d'exclure des solutions symétriques. Dans notre exemple :  
$$X + Y = 10 \wedge X * Y = 24 \wedge X \leq Y$$
- ▶ Plus sur les symétries dans 2 semaines.

## Propagateurs

- ▶ Un *propagateur* est un fil d'exécution (thread) qui peut renforcer le domaine.
- ▶ On peut voir un (ou des) propagateur(s) comme la *réalisation opérationnelle* d'une contrainte.
- ▶ Formellement, un propagateur  $p$  est une fonction qui envoie un domaine  $D$  vers un nouveau domaine  $p(D)$ .
- ▶ Plus sur les propagateurs la semaine prochaine !

## Un propagateur pour la contrainte $X \leq Y$

$$\begin{aligned} p(D(X)) &= D(X) \cap \{n \mid n \leq \max(D(Y))\} \\ p(D(Y)) &= D(Y) \cap \{n \mid n \geq \min(D(X))\} \\ p(D(Z)) &= D(Z) \quad \text{si } Z \text{ variable différente de } X, Y \end{aligned}$$

## Exemple

- ▶  $D_1(X) = \{7, \dots, 12\}, D_2(Y) = \{5, \dots, 10\}$
- ▶ Soit  $p$  le propagateur du transparent précédent.
- ▶  $p(D_1) = D_2$  t.q.  $D_2(X) = \{7, \dots, 10\}, D_2(Y) = \{7, \dots, 10\}$
- ▶ Dans cet exemple on a même  $p(D_2) = D_2$
- ▶ Ce propagateur  $p$  est même *idempotent* :  $p(p(D)) = p(D)$  pour *tout* domaine  $D$  (pourquoi ?).

## Propagateurs en Oz

Il y a des propagateurs  $=$   $>$   $>=$   $<$   $=<$   $\backslash=$  qui réalisent une propagation de *bornes*!

```
X*Y=:24  
X+Y=:10  
X=<:Y
```

Attention : ne pas oublier le  $\ll$  :  $\gg$  quand on veut écrire un propagateur.  
L'application d'un propagateur peut déclencher l'application d'un autre propagateur !

## Chercher des solutions

- ▶ On applique d'abord tous les propagateurs tant que possible. C'est le calcul d'un point fixe de l'ensemble de *tous* les propagateurs qui ont été créés.
- ▶ Si pas d'échec et s'il y a une variable  $X$  qui n'est pas fixée : créer une alternative (nœud dans l'arbre de recherche).
- ▶ Similaire aux choix de la programmation logique en Oz, mais le mécanisme des contraintes est beaucoup plus puissant.

## Questions de stratégie

- ▶ S'il y a plusieurs variables qui ne sont pas fixées, laquelle choisir ?
- ▶ Si le domaine de  $X$  n'est pas fixé, comment le couper en deux ?
- ▶ Comment organiser le calcul quand on a plusieurs alternatives :
  - ▶ recherche en profondeur d'abord (stratégie de Prolog) ?
  - ▶ recherche en largeur d'abord ?
  - ▶ autres ?

## Imposer des domaines de variables en Oz

- ▶ Imposer le domaine de la variable  $D$  comme étant l'intervalle  $[Lower \dots Upper]$  :  

```
D :: Lower#Upper
{FD.int Lower#Upper D}
```
- ▶ Imposer que  $D$  est une variable de domaine fini entre 0 et le maximum des domaines finis :  

```
{FD.dec1 D}
```

## Imposer des domaines de variables en Oz

- ▶ Imposer que  $L$  est une liste de  $I$  variables de domaine fini :  

```
{FD.list I Lower#Upper L}
```
- ▶ Imposer que  $T$  est un  $n$ -uplet de variables de domaine fini, de longueur  $I$  et label  $L$  :  

```
{FD.tuple L I Lower#Upper T}
```
- ▶ Imposer que  $R$  est un enregistrement de variables de domaine fini, avec liste de features  $F$  et label  $L$  :  

```
{FD.record L F Lower#Upper R}
```

## Imposer des domaines d'un vecteur

- ▶ *Vecteur* : un enregistrement, ou un n-uplet qui n'est pas une liste
- ▶ Restreindre les domaines de toutes les variables d'un vecteur  $V$  :  
 $V::\text{Lower}\#\text{Upper}$   
{FD.dom Lower#Upper V}

## L'exemple Rectangle en Oz

```
declare
proc {Rectangle Sol}
  sol(X Y)=Sol
in
  X::1#9
  Y::1#9
  X*Y=:24
  X+Y=:10
  X<:Y
  {FD.distribute naive Sol}
end
```

## Send More Money en Oz

```
declare
proc {SendMoreMoney Sol}
  local
    S E N D M O R Y
  in
    Sol=sol(s:S e:E n:N d:D m:M o:O r:R y:Y)
    Sol::0#9
    {FD.distinct Sol}
    S\=:0
    M\=:0
    1000*S + 100*E + 10*N + D
    + 1000*M + 100*O + 10*R + E
    =: 10000*M + 1000*O + 100*N + 10*E + Y
    {FD.distribute ff Sol}
  end
end
```

## Resoudre

```
{Browse {SearchAll {Queens 8}}}
{ExploreOne {Queens 8}}
{ExploreAll {Queens 8}}
```

## Propagateurs en Oz

- ▶ propagation de bornes pour des contraintes arithmétiques :  
=:, \=:, <:, >:, =<:, =>:

- ▶ propagation pour la distance entre deux variables :

```
{FD.distance X Y Rel Z}
```

Exemple :

```
{FD.distance X Y '>:' 8}
```

impose que la différence entre X et Y est strictement plus grande que 8.

## Propagateurs en Oz

- ▶ Propager que toutes les variables dans un vecteur ont des valeurs différentes (peut créer des trous dans les domaines) :  
`FD.distinct V`
- ▶ Propager que toutes les variables dans un vecteur  $V$  sont différentes par rapport à un vecteur de décalage  $D$  : pour tous  $i, j$  :  $V.i + D.i \neq V.j + D.j$  :  
`{FD.distinctOffset V D}`
- ▶ Voir `System Modules` de la documentation Oz, chapitre 5.

## Exemple propagateur

```
declare X Y Z
{Browse [X Y Z]}
X :: 1#13
Y :: 0#27
Z :: 1#12
2*Y =: Z
X <: Y
Z <: 7
X \=: 1
```

```
% [X Y Z]
% [X[1#13] Y Z]
% [X[1#13] Y[0#27] Z]
% [X[1#13] Y[0#27] Z[1#12]]
% [X[1#13] Y[1#6] Z[2#12]]
% [X[1#5] Y[2#6] Z[4#12]]
% [X[1#2] Y[2#3] Z[4#6]]
% [2 3 6]
```

## Rappel : Chercher des solutions

- ▶ On applique d'abord tous les propagateurs tant que possible. C'est le calcul d'un point fixe de l'ensemble de *tous* les propagateurs qui ont été créés.
- ▶ Si pas d'échec et s'il y a une variable  $X$  qui n'est pas fixée : créer une alternative (nœud dans l'arbre de recherche).
- ▶ Similaire aux `choice` de la programmation logique en Oz, mais le mécanisme des contraintes est beaucoup plus puissant.

## Résoudre des contraintes de domaine fini

- ▶ Écrire une *script* pour un problème donné.
- ▶ Un script est une procédure à un seul argument (appelé sa *racine*)
- ▶ Le rôle du script est de lier sa racine à une solution du problème *quand l'arbre de recherche est construit*.
- ▶ Quand on construit l'arbre de recherche complet :
  - ▶ Dans toute feuille la racine doit être liée à une solution de la contrainte (correction du script)
  - ▶ Toute solution de la contrainte doit se trouver comme valeur de la racine dans une feuille (complétude du script). Parfois complétude modulo des symétries du problème.

## Schéma d'un script

```
proc {Script Root}
    % déclarer des variables
in
    % propageurs pour la contrainte
    % spécifier la stratégie de distribution
end
```

Quand la solution consiste en plusieurs composantes en doit lier Root à une structure (liste, n-uplet, enregistrement). Par exemple :

```
Root = solution(x:X y:Y z:Z)
```

## Construire l'arbre de recherche

- ▶ {SearchAll Script} donne la liste des toutes les solutions
- ▶ {SearchOne Script} donne soit une liste avec un seul élément qui est la première solution trouvée, soit la liste vide quand il n'y a pas de solution
- ▶ {ExploreAll Script} construit l'arbre de recherche complet et le présente dans la fenêtre de l'explorateur
- ▶ {ExploreOne Script} construit un début de l'arbre de recherche jusqu'à la première feuille réussie, et le présente dans la fenêtre de l'explorateur.