Programmation Logique et Par Contraintes Avancée
Cours 5 – Contraintes de domaine fini en Oz

Ralf Treinen

Université Paris Diderot
UFR Informatique
IRIF, équipe PPS

treinen@irif.fr

4 février 2019

Contraintes sur un domaine fini

- On peut donner pour certaines variables un *choix fini* de leurs valeurs.
- Ces valeurs possibles sont des entiers non négatives.
- Utiles pour modéliser des problèmes où il faut trouver une affectation à des variables sous des contraintes.
- Exemples : ordonnancement, 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, on y reviendra.

Domaines

- Un *domaine* D est une fonction partielle qui associe à certaines variables un ensemble fini (éventuellement vide) d'entiers non négatives. $vars(D)$ = l'ensemble des variables pour lesquelles D est défini.
- Sur l'exemple : on a un domaine initial (défini par l'énoncé du problème)

$$\begin{array}{rcl} D(X) & = & \{1, \ldots, 9\} \\ D(Y) & = & \{1, \ldots, 9\} \end{array}$$

- Ici on a donc même des intervalles, mais ce n'est pas nécessairement le cas.
- Notation Oz pour imposer un domaine fini à une variable : X::1#9 Y::1#9

Domaines (suite)

- Nouveau type de valeurs dans la mémoire : *ensemble fini* (finite domain).
- 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).

Exemples (domaines1.oz) I

```
% bind a variable to a finite domain
declare X
X::1#9
{Browse X}

% bind a variable to the maximal finite domain
declare X
{FD.decl X}
{Browse X}
```


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 quelques 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 $X \leq Y$

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

Exemple

- $D_1(X) = \{7, \ldots, 12\}, D_1(Y) = \{5, \ldots, 10\}$
- Soit p le propagateur du transparent précédent.
- $p(D_1) = D_2$ t.q. $D_2(X) = \{7, \ldots, 10\}, D_2(Y) = \{7, \ldots, 10\}$
- 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 =: >: >=: <: =<: \=: qui réalisent une propagation de *bornes* !
- Exemple :

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

- Attention : ne pas oublier le « : » quand on veut écrire un propagateur.
- L'application d'un propagateur peut déclencher l'application d'un autre propagateur !

Exemples (propagators1.oz) I

```
declare X Y Z
{Browse [X Y Z]}        % [X Y Z]
X :: 1#13               % [X[1#13] Y Z]
Y :: 0#27               % [X[1#13] Y[0#27] Z]
Z :: 1#12               % [X[1#13] Y[0#27] Z[1#12]]

2*Y =: Z                % [X[1#13] Y[1#6]  Z[2#12]]

X <: Y                  % [X[1#5]  Y[2#6]  Z[4#12]]

Z <: 7                  % [X[1#2]  Y[2#3]  Z[4#6]]

X \=: 1                 % [2 3 6]
```


Plusieurs propagateurs pour la même contrainte

- On peut avoir plusieurs propagateurs pour la même contrainte.
- Du coup, qu'est-ce que ça veut dire qu'un propagateur "est pour" une contrainte ? Voir la semaine prochaine.
- Ces propagateurs peuvent se distinguer par leur "puissance" de propagation.
- Un propagateur plus fort peut être aussi plus coûteux à exécuter.

Exemples (propagators2.oz) I

```
% FD.distinct : n'echoue pas sur cet exemple
declare X Y Z
[X Y Z]:::1#2
{Browse [X Y Z]}
{FD.distinct [X Y Z]}

% FD.distinctD : full domain propagation
declare X Y Z
[X Y Z]:::1#2
{Browse [X Y Z]}
{FD.distinctD [X Y Z]}
```


Exemples (linear.oz) I

```
declare
proc {Script S}
   X Y Z
in
   [X Y Z]:::0#9
   S=[X Y Z]
   100*X + 10*Y + Z =: 342
end

% solution found directly by constraint simplification –
{Browse {SearchAll Script}}

% no search tree constructed
{ExploreAll Script}
```

Exemples (square.oz) I

```
declare
proc {Script S}
   X Y Z
in
   {FD.decl X}
   S=X
   X*X =: 77
end

% finding a solution requires real search
% what we get here is only a domain, not an integer value
% we need a way to construct a search tree: distribution
{Browse {SearchAll Script}}

{ExploreAll Script}
```


Exemples (square2.oz) I

```
declare
proc {Script S}
   X Y Z
in
   {FD.decl X}
   S=X
   X*X =: 81
   {FD.distribute naive [S]}
end

% finding a solution requires real search: distribute!
{Browse {SearchAll Script}}

{ExploreAll Script}
```


Résoudre un problème de contraintes

L'algorithme vu de loin

- 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 crées.
- 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), on découpant le domaine d'une variable en deux
- Descente dans une branche de l'arbre de recherche : réitérer !

Contraintes et choix

- Arbre de recherche similaire à Prolog.
- Mécanisme plus puissant car utilisation des propagateurs avant la création d'un choix.

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 ?
- Réponses : voir des cours ultérieurs.

Imposer des domaines de variables en Oz

- Deux formes équivalentes pour imposer le domaine de la variable D comme étant l'intervalle $[Lower \ldots Upper]$:

```
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.decl D}
```

- Imposer un domaine de D par énumération des valeurs :

```
D::[17 42 73]
```

Programmation Logique et Par Contraintes Avancée Cours 5 – Contraintes de domaine fini en Oz
└Définir les domaines des variables

Imposer des domaines de variables en Oz

- Imposer que L est une liste de N variables de domaine fini :

```
{FD.list N Lower#Upper L}
```

- Imposer que T est un n-uplet de variables de domaine fini, de longueur N et label L :

```
{FD.tuple L N 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}
```

Programmation Logique et Par Contraintes Avancée Cours 5 – Contraintes de domaine fini en Oz
└Définir les domaines des variables

Exemples (domaines2.oz) I

```
% bind L to a list of 5 finite domains 10..20
declare L
{FD.list 5 10#20 L}
{Browse L}

% bind T to a 5-tuple of finite domains 10..20, and label 5
declare T
{FD.tuple f 5 10#20 T}
{Browse T}

% bind R to a record with keys a,b,c, all values are finite
% domains 10..20, and label f
declare R
{FD.record f [a b c] 10#20 R}
{Browse R}
```

Programmation Logique et Par Contraintes Avancée Cours 5 – Contraintes de domaine fini en Oz
└Définir les domaines des variables

Imposer des domaines d'un vecteur

- *Vecteur* : un enregistrement (n-uplet inclus), ou une liste.
- Restreindre les domaines de toutes les variables d'un vecteur V (tous les enfants *directs* d'un enregistrements, ou tous les éléments d'une liste) :

```
V:::Lower#Upper
{FD.dom Lower#Upper V}
```

Exemples (domaines3.oz) I

```
% restreindre toutes les domaines de toutes les variables d'un vecteur
declare V A B C
V=f(A B C)
V:::10#20
{Browse V}

declare V A B C
V=[A B C]
V:::10#20
{Browse V}

% doesn't work on nested tuples
declare V A B C
V=f(A g(B C))
V:::10#20 % error
{Browse V}
```


Propagateurs en Oz

- Propagation de bornes pour des contraintes arithmétiques :

  =:, \=:, <:, >:, =<:, >=:

- Propagation pour la distance entre deux variables :

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

  où Rel est un des atomes donnés au-dessus.
- 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.

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}
   % declarer des variables
in
   % mettre les domaines initiales
   % propagateurs pour la constrainte
   % 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)
```


Continuation de l'exemple

On peut interagir avec l'arbre de recherche à l'aide de l'explorateur vu au dernier cours :

```
{ExploreOne Script}

{ExploreAll Script}
```


Exemples (rectangle.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

{Browse {SearchAll Rectangle}}

{ExploreOne Rectangle}
```


Quelque mots sur *distribute*

- Il y a des stratégies (de choix de variable, du découpage du domaine d'une variable) prédéfinies, par exemple *naive*, ou *ff*. On y reviendra.
- La variable pour laquelle Oz va créer une alternative (choice point) sera choissie par Oz à chaque moment parmi les *fils directs* du dernier argument de *distribute*, ici `Sol`. Attention dans les cas où les variables paraissent plus profondement dans `Sol`.

Send More Money

Un problème classique :

$$
\begin{array}{cccccc}
 & & S & E & N & D \\
+ & & M & O & R & E \\
\hline
= & M & O & N & E & Y
\end{array}
$$

Trouver des valeurs pour les lettres tel que

- des lettres différentes dénotent des chiffres différents ;
- aucune des lignes commence sur 0 ;
- la somme est correcte.

Exemples (sendmoremoney.oz) II

```
{Browse {SearchAll SendMoreMoney}}

{ExploreAll SendMoreMoney}
% try also with strategy ff
```


Exemples (sendmoremoney.oz) I

```
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}
      {FD.distribute naive Sol}
   end
end
```