Ralf Treinen

Universite Paris Diderot
UFR Informatique
Laboratoire Preuves, Programmes et Systemes
treinen@pps.jussieu.fr

Programmation Logique et Par Contraintes Avancee Cours 5 { Contraintes de domaine ni en Oz

Programmation Logique et Par Contraintes Avancee Cours 5 { Contraintes de domaine ni en Oz

Programmation Logique et Par Contraintes Avancee Cours 5 { Contraintes de domaine ni en Oz

- On a pour toute variable un *choix fini* de ses valeurs.
- Utiles pour modeliser des problemes ou il fait trouver une a ectation a des variables sous des contraintes.
- Exemples : ordonnancement, emploi du temps, routage, etc.
- Beaucoup d'applications industrielles.

Trouver les côtes d'un rectangle a dimensions entieres tel que

- surface = 24 unites
- perimetre = 20 unites

- Toutes les valeurs sont des entiers.
- Il y a deux variables a determiner (largeur X et hauteur Y)
- Pour chacune des deux variables il y a un choix ni entre 1 et 9 (pourquoi ?)
- On peut supposer que X Y. (pourquoi ?)
  Il s'agit d'une *symétrie* du probleme

- Un *domaine* D est une fonction partielle qui associe a une variable un ensemble ni (eventuellement vide) d'entiers. $vars(D)$ = l'ensemble des variables pour lesquelles D est de ni.
- Sur l'exemple : on a un domaine initial (de nie par l'enonce du probleme) qui associe a X l'ensemble $f1, \ldots, 9g$, et a Y le même ensemble $f1, \ldots, 9g$. Ici on a donc même des intervalles, mais ce n'est pas necessairement le cas.
- Notation Oz : `X::1#9 Y::1#9`

- Nouveau type de valeur dans la memoire : *ensemble fini* (nite domain).
- Un domaine D a *échoué* (is failed) : il existe une variable X telle que $D(X) = ;$.
- 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 $D_1(X)$ $D_2(X)$ pour tout X 2 $vars(D_1)$.
- Le but du jeu est de reduire le domaine, a n d'obtenir soit un domaine echoue (il n'y a pas de solution), ou d'obtenir un domaine qui xe toutes les variables du probleme (solution trouvee).

- Une *contrainte* est une formule logique (equation, inegalite, disequation, etc.).
- La contrainte est la formulation *mathématique* du probleme qu'on souhaite resoudre.
- Sur l'exemple : $X + Y = 10$ ^ X $Y = 24$
- On peut parfois ajouter des contraintes supplementaires a n d'exclure des solutions symetriques. Dans notre exemple :

$$X + Y = 10 \text{ ^ } X \quad Y = 24 \text{ ^ } X \quad Y$$

- Plus sur les symetries plus tard.

- Un *propagateur* est un l d'execution (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 !

pour X Y

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

- $D_1(X) = \{7, \ldots, 12\}, D_1(Y) = \{5, \ldots, 10\}$
- Soit p le propagateur du transparent precedent.
- $p(D_1) = D_2$ 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 ?).

Il y a des propagateurs `=:` `>:` `>=:` `<:` `=<:` `\=:` qui realisent une propagation de *bornes* !

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

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

- On applique d'abord tous les propagateurs tant que possible. C'est le calcul d'un point xe de l'ensemble de *tous* les propagateurs qui ont ete crees.
- Si pas d'echec et s'il y a une variable X qui n'est pas xee : creer une alternative (n ud dans l'arbre de recherche), on decoupant le domaine d'une variable en deux
- Descente dans une branche de l'arbre de recherche : reiterer !

- Arbre de recherche similaire a Prolog.
- Mecanisme plus puissant car utilisation des propagateurs avant la creation d'un choix.

- S'il y a plusieurs variables qui ne sont pas xees, laquelle choisir ?
- Si le domaine de X n'est pas xe, comment le couper en deux ?
- Comment organiser le calcul quand on a plusieurs alternatives :
  - recherche en profondeur d'abord (strategie de Prolog) ?
  - recherche en largeur d'abord ?
  - autres ?
- Reponses : voir des cours ulterieurs.

- Deux formes equivalentes pour imposer le domaine de la variable D comme etant l'intervalle [*Lower* . . . *Upper*] :

```
D::Lower#Upper
{FD.int Lower#Upper D}
```

- Imposer que D est une variable de domaine ni entre 0 et le maximum des domaines nis :

```
{FD.decl D}
```


- Imposer que L est une liste de I variables de domaine ni :

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

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

```
{FD.tuple L I Lower#Upper T}
```

- Imposer que R est un enregistrement de variables de domaine ni, avec liste de features F et label L :

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

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

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


- propagation de bornes pour des contraintes arithmetiques : `=:`, `\=:`, `<:`, `>:`, `=<:`, `>=:`
- propagation pour la distance entre deux variables :

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

Exemple :

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

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

- Propager que toutes les variables dans un vecteur ont des valeurs di erentes (peut creer des trous dans les domaines) :

```
FD.distinct V
```

- Propager que toutes les variables dans un vecteur V sont di erentes par rapport a un vecteur de decalage 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.


```
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]
```

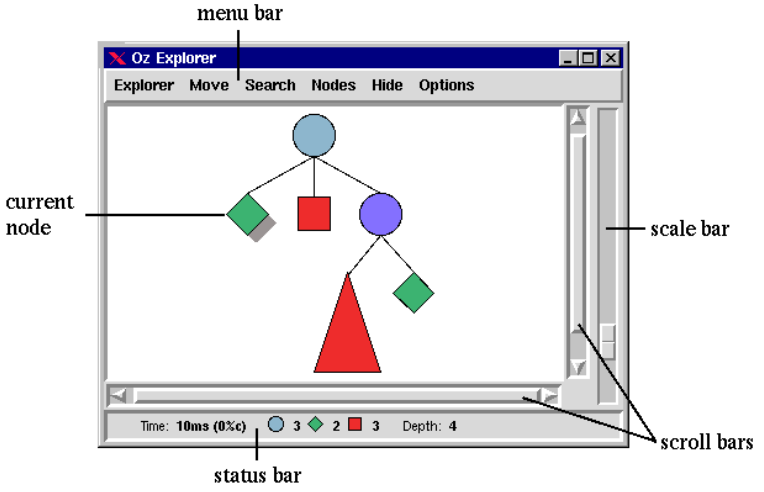
- Ecrire une *script* pour un probleme donne.
- Un script est une procedure a un seul argument (appele sa *racine*)
- Le rôle du script est de lier sa racine a une solution du probleme *quand l'arbre de recherche est construit*.
- Quand on construit l'arbre de recherche complet :
  - Dans toute feuille, la racine doit être liee a une solution de la contrainte (correction du script)
  - Toute solution de la contrainte doit se trouver comme valeur de la racine dans une feuille (completude du script). Parfois completude modulo des symetries du probleme.


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

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

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


- L'explorateur est un outil interactif pour etudier l'arbre de recherche
- Il utilise des couleurs di erents pour indiquer l'etat d'un n ud :
  - vert : succes
  - rouge :echec
  - bleu : n ud de choix qui n'a pas echoue
- Cliquer sur un n ud : a che les valeurs des variables locales dans une fenêtre *Inspector*.
- On peut demander de continuer la recherche.



```
{ExploreOne Script}

{ExploreAll Script}
```

Programmation Logique et Par Contraintes Avancee Cours 5 { Contraintes de domaine ni en Oz
└L'explorateur et l'inspecteur

- `{ExploreOne F}`, ou F est une fonction a un argument, a che l'arbre de recherche pour `{F X}` jusqu'a la premiere solution.
- `{ExploreAll F}`, ou F est une fonction a un argument, a che l'arbre de recherche complet pour `{F X}`.

Programmation Logique et Par Contraintes Avancee Cours 5 { Contraintes de domaine ni en Oz
└L'explorateur et l'inspecteur

- f`SearchAll Script`g donne la liste des toutes les solutions
- f`SearchOne Script`g donne soit une liste avec un seul element qui est la premiere solution trouvee, soit la liste vide quand il n'y a pas de solution
- f`ExploreAll Script`g construit l'arbre de recherche complet et le presente dans la fenêtre de l'explorateur
- f`ExploreOne Script`g construit un debut de l'arbre de recherche jusqu'a la premiere feuille reussie, et le presente dans la fenêtre de l'explorateur.

Programmation Logique et Par Contraintes Avancee Cours 5 { Contraintes de domaine ni en Oz
└L'explorateur et l'inspecteur

```
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
```

```
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
```

Programmation Logique et Par Contraintes Avancee Cours 5 { Contraintes de domaine ni en Oz
└L'explorateur et l'inspecteur

```
{Browse {SearchAll SendMoreMoney}}

{ExploreAll SendMoreMoney}
```