Programmation Logique et Par Contraintes Avancée Cours 7 – Blocages de Propagateurs et Recherche Multidimensionelle

Ralf Treinen

Université Paris Diderot
UFR Informatique
IRIF, équipe PPS

treinen@irif.fr

18 février 2019

Types

- Dans la documentation de Oz, les noms utilisés pour des arguments de procédures indiquent leur *type* et leur *mode* attendus.
- Le type est en vérité le type attendu de la valeur (car Oz n'est pas statiquement typé), notament

lettre	type	lettre	type
I	entier	B	booléen
R	record	P	procédure
X	n'importe		
s final	liste	v final	vecteur

- Exemple :

```
{List.length +Xs ?I}
```


Modes

- Le mode peut être `?`, `+`, `*`, `$`, ou rien
- Le mode indique le comportement de *synchronisation* de la procédure :
  - Le mode `?` indique qu'il s'agit d'un *résultat*
  - Le mode `+` indique que l'argument doit être suffisament déterminée, sinon la procédure suspend.
  - Le mode `*` indique que l'argument doit avoir un domaine fini (ou même être une valeur entière)
  - Tous les arguments avec mode `$`, sauf éventuellement un, doivent avoir un domaine fini.

Exemples (mod1.oz)

```
% Problème : pas de propagateur !
declare
proc {D L}
   X Y Z
in
   L=X#Y
   [X Y]:::1#9
   Z::2#4
   X mod Z = Y
   {FD.distribute ff [X Y Z]}
end

{Browse {SearchAll D}}
% donne simplement une paire de deux domaines
```

Analyse : mod1.oz

- On a donnée tous les domaines initiaux qu'il faut.
- On a mis le bon `FD.distribute`
- Le problème est l'appel à `mod`. Regardons ce qui dit la doc (Oz Base Environment, 4.2 Integers) :

  `{Int.'mod' +I1 +I2 ?I3}`
- Donc l'appel à `mod` suspend car les deux premiers arguments sont des domaines et pas des entiers.

Exemples (mod2.oz)

```
% Toujours Blocage!

declare
proc {D L}
   X Y Z
in
   L=X#Y
   [X Y]:::1#9
   Z::2#4
   {FD.modI X Z Y}
   {FD.distribute ff [X Y Z]}
end

{Browse {SearchAll D}}
```


Exemples (mod3.oz)

```
% Ca bloque où exactement ?
declare
proc {D L}
   X Y Z
in
   [X Y]:::1#9
   Z::2#4
   {FD.modI X Z Y}
   L=X#Y
   {FD.distribute ff [X Y Z]}
end

{Browse {SearchAll D}}
% donne une variable non-determinée
```


Propagateurs qui peuvent suspendre

Voir la documentation Oz : *System Modules*, Section *5.11 Miscellaneous Propagators* :

- le propagateur pour + : `{FD.plus $D1 $D2 $D3}`
- le propagateur pour *mod* : `{FD.modI $D1 +I $D2}` suspend quand le diviseur n'est pas une valeur déterminée !
- Solutions possibles :
  - énumérer avant de tester avec div et mod
  - utiliser des propagateur pour la multiplication pour exprimer la division avec reste.

Exemples (mod4.oz) I

```
% Énumerer pour éviter le blocage ?
declare
proc {D L}
   X Y Z
in
   L=X#Y
   [X Y]:::1#9
   Z::2#4
   {FD.distribute ff [Z]}
   X mod Z = Y
   {FD.distribute ff [X Y]}
end

{Browse {SearchAll D}}
% donne trois paires de domaines
```


Exemples (mod5.oz) I

```
% Il faut quand même un propagateur !
declare
proc {D L}
   X Y Z
in
   L=X#Y
   [X Y]:::1#9
   Z::2#4
   {FD.distribute ff [Z]}
   {FD.modI X Z Y}
   {FD.distribute ff [X Y]}
end

{Browse {SearchAll D}}
% marche !
```


Exemples (mod6.oz) I

```
% Solution alternative avec propagateurs pour addition et multiplication
declare
proc {D L}
   X Y Z
in
   L=X#Y [X Y]:::1#9 Z::2#4
   local P1 P2 in
      [P1 P2]:::0#9
      P1 * Z =: P2
      P2 + Y =: X
      Y <: Z
   end
   {FD.distribute ff [X Y Z]}
end

{Browse {SearchAll D}}
```


Exemples (domain.oz)

```
declare X Y
{Browse [X Y]}

X :: 1#10

Y :: X#10 % bloque, attend la valeur de X

X = 5     % maintenant Y a son domaine
```

Propagateurs pour la diségalité

- Reference Manual *System Modules*, chapitre 5 *Finite Domain Constraints*, section 5.8 *Symbolic Propagators*.
- `{FD.distinct *Dv}`
  All elements in Dv are pairwise distinct.
- `{FD.distinctOffset *Dv +Iv}`
  All sums $D_i + I_i$ are pairwise distinct.

Exemples (queens.oz) I

```
% N Queens

declare
fun {Queens N}
   proc {$ Row}
      L1N ={MakeTuple c N}
      LM1N={MakeTuple c N}
   in
      {FD.tuple queens N 1#N Row}
      {For 1 N 1 proc {$ I}
                    L1N.I=I LM1N.I=~I
                 end}
      {FD.distinct Row}
      {FD.distinctOffset Row LM1N}
      {FD.distinctOffset Row L1N}
      {FD.distribute generic(value:mid) Row}
   end
```


Exemples (queens.oz) II

```
end

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


AtMost, AtLeast, Exactly

- Reference Manual *System Modules*, chapitre 5 *Finite Domain Constraints*, section 5.8 *Symbolic Propagators*.
- ```
  {FD.atMost *D *Dv +I}
  {FD.atLeast *D *Dv +I}
  {FD.exactly *D *Dv +I}
  ```
- - le premier argument doit être un domaine fini (ou un entier) ;
  - le deuxième argument doit être un vecteur de domaines finis ;
  - le troisième argument doit être un entier.
- At most, at least, exactly D elements of Dv are equal to I.
  ```

Exemples (exactly.oz)

```
declare L N in
{FD.list 5 1#10 L}
N::0#20
{Browse L}
{Browse N}

{FD.exactly N L 8}

{Nth L 1} <: 5
{Nth L 2} <: 5
```


Exemples (atleast.oz)

```
% Propagators FD.atLeast

declare L in
{FD.list 5 1#10 L}
{Browse L}

{Nth L 1} <: 5
{Nth L 2} <: 5

{FD.atLeast 3 L 8}
```


Exemple : Emploi du temps d'un colloque

- On veut faire l'emploi du temps pour un colloque.
- L'emploi du temps va consister en plusieurs créneaux. Une session occupe toute la durée d'un créneau, mais on peut a priori avoir plusieurs sessions en parallèle.
- Donné : un nombre de sessions, nombre max de sessions en parallèle
- On cherche une solution qui occupe un nombre minimal de créneaux
- Contrainte possible pour une session n : doit être avant une autre session m, ou ne doit pas être en parallèle avec une autre session m.

Exemples (colloque1.oz) I

```
declare
Colloque =
  data(nbSessions:11  nbParSessions:3
       constraints: [ before(4 11)  before(5 10)
                      before(6 11)
                      disjoint(1 [2 3 5 7 8 10])
                      disjoint(2 [3 4 7 8 9 11])
                      disjoint(3 [5 6 8])
                      disjoint(4 [6 8 10])
                      disjoint(6 [7 10])
                      disjoint(7 [8 9])
                      disjoint(8 [10]) ] )
```

Solution pour le colloque

- Recherche en deux dimensions : nombre de créneaux d'abord, puis affectation des créneaux aux sessions
- Propagateur pour imposer qu'au plus M variables du vecteur V ont la valeur X :

```
{FD.atMost M V X}
```


L'exemple Colloque

Cet exemple montre plusieurs choses :

- Le script engendre les propagateurs à partir des données.
- Recherche multi-dimensionnelle. Faire attention à l'ordre dans lequel on distribue.
- En occurrence : `FD.int` suspend quand les bornes inférieurs et supérieures ne sont pas des entiers !

Exemples (colloque2.oz) I

```
declare
fun {Conference Data}
   NbSessions    = Data.nbSessions
   NbParSessions = Data.nbParSessions
   Constraints   = Data.constraints
   MinNbSlots    = NbSessions div NbParSessions
in
   proc {$ Plan}
      NbSlots  = {FD.int MinNbSlots#NbSessions}
   in
      {FD.distribute naive [NbSlots]}
      {Browse NbSlots}
      %% Plan: Session --> Slot
      {FD.tuple plan NbSessions 1#NbSlots Plan}
      %% at most NbParSessions per slot
      {For 1 NbSlots 1
       proc {$ Slot} {FD.atMost NbParSessions Plan Slot} end}
```


Exemples (colloque2.oz) II

```
      %% impose constraints
      {ForAll Constraints
       proc {$ C}
          case C
          of before(X Y) then
             Plan.X <: Plan.Y
          [] disjoint(X Ys) then
             {ForAll Ys proc {$ Y} Plan.X \=: Plan.Y end}
          end
       end}
      {FD.distribute ff Plan}
   end
end

{Browse {SearchOne {Conference Colloque}}}
```