Programmation Logique et Par Contraintes Avancee
Cours 7 { Blocages de Propagateurs et Recherche Multidimensionelle

Ralf Treinen

Université Paris Diderot
UFR Informatique
Laboratoire Preuves, Programmes et Systèmes
treinen@pps.univ-paris-diderot.fr

Contenu cours 7

Propagateurs pour la disegalite

- `{FD.distinct *Dv}`
  All elements in Dv are pairwise distinct.
- `{FD.distinctOffset *Dv +Iv}`
  All sums $D_i + I_i$ are pairwise distinct.
- `{FD.atMost *D *Dv +I}`
  `{FD.atLeast *D *Dv +I}`
  `{FD.exactly *D *Dv +I}`
  At most, at least, exactly D elements of Dv are equal to I.

Pour en savoir plus : Reference Manual , chapitre 5 , section 5.8 .

Exemple : N dames

```
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
end
```

Resoudre

L'appel {Queens 8} donne le script pour 8 dames.

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

{ExploreOne {Queens 8}}

{ExploreAll {Queens 8}}
```


Un piege : utiliser le mauvais 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
```


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}}

% pareil
```


Ca bloque ou 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 : , Section
:

- le propagateur pour + : `{FD.plus $D1 $D2 $D3}`
- le propagateur pour : `{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 (cher !)
  - utiliser des propagateur pour la multiplication

Enumerer pour eviter 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
```


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 !
```


Solution alternative

```
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
      {FD.times Z P1 P2} % P2=Z*P1
      P2 + Y =: X
      Y <: Z
      {FD.distribute ff [X Y Z]}
   end
end
```

Exemple : Emploi du temps d'un colloque

- On veut faire l'emploi du temps pour un colloque
- Donné : un certain nombre de sessions.
- 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.
- On cherche une solution qui occupe un nombre minimal de créneaux
- Contrainte : nombre maximal de sessions en parallèle
- Contrainte possible pour une session : doit être avant une autre session , ou ne doit pas être en parallèle avec une autre session .

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}`

Colloque : les donnees

```
Data = 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]) ] )
```


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.
- On occurrence : `FD.int` suspend quand les bornes inférieurs et supérieures ne sont pas des entiers !