Programmation Logique et Par Contraintes Avancée
Cours 4 – Programmation logique en Oz

Ralf Treinen

Université Paris Diderot
UFR Informatique
IRIF, équipe PPS

treinen@irif.fr

28 janvier 2019

Contenu cours 4

Recherche

- Construction d'un *arbre de recherche*
- Exploitation d'un arbre de recherche, av ec retour en arrière (*backtracking*) si nécessaire.
- Recherche est *le* paradigme de calcul en Prolog.
- En Oz, la recherche est une construction du langage qu'on peut utiliser quand on en a besoin.
- Permet d'exprimer plusieurs possibilités de continuer le calcul (non-déterminisme).

Deux types de non-déterminisme

- Non-déterminisme « don't care » :
  - Toutes les alternatives sont bonnes et mènent à un résultat équivalent.
  - Il suffit de laisser la machine faire son choix.
  - Exemple : Ordonnancement des threads on Oz.
  - Aussi appelé *indeterminisme*
- Non-déterminisme « don't know »
  - On ne sait pas quel est le bon choix parmi les alternatives offertes.
  - La machine doit exploiter tous les choix possibles et construire un arbre de recherche.
  - Exemple : programmation logique.

Arbre de recherche

- Opération **choice** ... **end** pour indiquer un *choix* entre plusieurs possibilités de continuer.
- L'exécution d'un **choice** crée un *choice point* avec plusieurs alternatives.
- Donne lieu à un *arbre de recherche*, les *choice points* sont les nœuds dans cet arbre.
- On a besoin d'une *machine de recherche* pour forcer l'exploitatation d'un tel arbre de recherche.

Machines de Recherche

- Il y a plusieurs machines de recherche en Oz : pour la recherche en profondeur d'abord (*depth-first*), la largueur d'abord (*breadth-first*), l'optimisation (*branch-and-bound*), ...
- En plus, on peut paramétrer ces machines pour controller la stratégie de recherche finement.
- On en parlera dans le contexte des contraintes.

`fail` et `Search`

- **fail** : fait échouer la branche courante de l'arbre de recherche.
- Le calcul reprend avec l'alternative suivante du *choice point* le plus proche (recherche en profondeur d'abord).
- Un échec de l'opération Tell (ajout des équations à la mémoire) exécute également un **fail**.

Obtenir les solutions

- {SearchOne P}, où P est une procédure à un argument, donne une liste qui est soit vide (aucune solution trouvée), soit une liste avec un seul élément X qui est la première solution de l'arbre de recherche pour {P X} (recherche en profondeur d'abord).
- {SearchAll P}, où P est une procédure à un argument, donne la liste des valeurs de X dans toutes les solutions de l'arbre de recherche pour {P X}.
- On écrit souvent la procédure P comme une fonction (à zero arguments).

Exemples (digit.oz) I

```
declare
fun {Digit}
   choice
      0 [] 1 [] 2 [] 3 [] 4
    [] 5 [] 6 [] 7 [] 8 [] 9
   end
end

% n'affiche rien car {Digit} suspend
{Browse {Digit}}

% premiere solution
{Browse {SearchOne Digit}}

% toutes les solutions
{Browse {SearchAll Digit}}
```


Palindromes

Chercher tous les nombres à quatre chiffres qui sont des palindromes, et qui sont le produit de deux nombres à deux chiffres.

```
declare
proc {Palindrome X}
   X=(10*{Digit}+{Digit})*(10*{Digit}+{Digit})
   (X>=1000)=true
   (X div 1000) mod 10 = (X div 1) mod 10
   (X div 100) mod 10 = (X div 10) mod 10
end

{Browse {SearchAll Palindrome}}
```

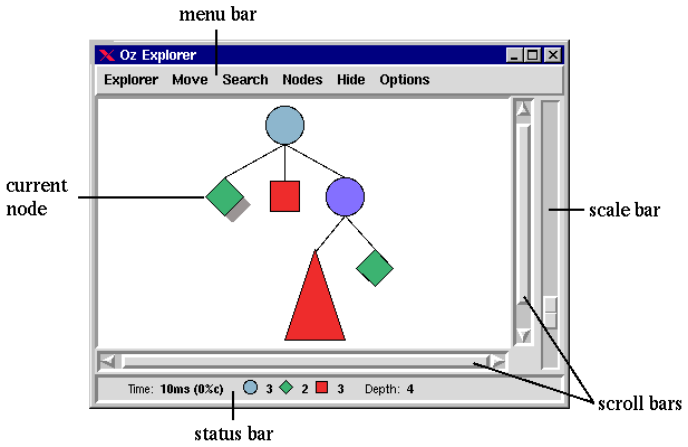

Recherche encapsulée

- On peut imaginer que, quand on entre dans une branche qui est enfant d'un *choice point*, qu'on garde une copie de l'état de mémoire pour être capable de revenir.
- L'implémentation peut être beaucoup plus efficace. Technique de la *Warren Abstract Machine* : défaire les modifications de la mémoire faites depuis le *choice point*.
- La recherche est *encapsulée* : un « calcul spéculatif » (dans une alternative) ne peut influencer les variables en-dehors de cette alternatives (propriétée importante pour le retour en arrière).
- Il n'y a pas de cut.
- Les *contraintes* que nous étudierons à partir de la semaine prochaine amènent également à la construction d'arbres de recherche (mais ne nécessitent pas de **choice** ... **end**).

L'explorateur

- L'explorateur est un outil interactif pour étudier l'arbre de recherche.
- Il utilise des couleurs différentes pour indiquer l'état d'un nœud :
  - vert : succès
  - rouge : échec
  - bleu : nœud de choix qui n'a pas échoué (bleu foncé si toutes les alternatives sont depliées)
- Double-clique sur un nœud : donne un numéro à ce nœud, et affiche les valeurs des variables locales dans une fenêtre *Inspector*.
- On peut demander de continuer la recherche.

Interface graphique



Explorer un arbre de recherche

- {ExploreOne F}, où F est une fonction à un argument, affiche l'arbre de recherche pour {F X} jusqu'à la première solution.
- {ExploreAll F}, où F est une fonction à un argument, affiche l'arbre de recherche complet pour {F X}.

Exemples (append1.oz) I

```
% deterministic: append of two lists
declare
fun {Append A B}
   case A
   of nil then B
   [] X|As then X|{Append As B}
   end
end

declare X in
{Append [1 2 3] [4 5 6] X}
{Browse X}

declare Y in
{Append Y [4 5] [1 2 3 4 5]}
{Browse Y}
```


Exemples (append2.oz) I

```
% deterministic: calculate prefix
declare
proc {Append2 A B C}
   if B==C then A=nil
   else
      case C of X|Cs
      then local As in
              A=X|As
              {Append2 As B Cs}
           end
      end
   end
end

declare X in
{Append2 [1 2 3] [4 5 6] X}
{Browse X}
```

Exemples (append2.oz) II

```
declare Y in
{Append2 Y [4 5] [1 2 3 4 5]}
{Browse Y}
```


Exemples (append3.oz) I

```
% Append Prolog-style
declare
proc {Append3 A B C}
   choice
      A=nil B=C
   [] local As Cs X in
         A=X|As
         C=X|Cs
         {Append3 As B Cs}
      end
   end
end

{Browse {SearchAll proc {$ X} {Append3 [1 2 3] [4 5 6] X} end}}

{ExploreOne proc {$ X} {Append3 [1 2 3] [4 5 6] X} end}
```


Exemples (append3.oz) II

```
{Browse {SearchAll proc {$ X} {Append3 X [4 5 6] [1 2 3 4 5 6]} end}}

{ExploreOne proc {$ X} {Append3 X [4 5 6] [1 2 3 4 5 6]} end}

{Browse {SearchAll
         proc {$ Z} local X Y in Z=X#Y {Append3 X Y [1 2 3 4]} end end} }

{ExploreOne
         proc {$ Z} local X Y in Z=X#Y {Append3 X Y [1 2 3 4]} end end}
```