

Programmation Logique et Par Contraintes Avancée

Cours 4 – Programmation logique en Oz

Ralf Treinen

Université Paris Diderot
UFR Informatique
Laboratoire Preuves, Programmes et Systèmes
treinen@pps.jussieu.fr

Contenu cours 4

Programmation logique en Oz

L'explorateur et l'inspecteur

Recherche

- ▶ Permet la programmation d'une *recherche*
- ▶ Recherche est *le* paradigme de base de Prolog, ici c'est une construction du langage qu'on peut éviter quand l'algorithme ne demande pas de recherche.
- ▶ Permet d'exprimer plusieurs possibilités de continuer le calcul. Si la première ne réussit pas alors on essaye la suivante.

Deux types de non-déterminisme

- ▶ Non-déterminisme « don't care » :
 - ▶ La façon comment le choix est fait (par la machine) n'est pas importante.
 - ▶ Il suffit de laisser la machine faire son choix
 - ▶ Exemple : Ordonnement des threads en Oz.
- ▶ 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 possible 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* : On essaye de continuer sur la première alternative, si cela échoue on revient au *choice point*.
- ▶ Donne lieu à un *arbre de recherche*, les *choice points* sont les nœuds dans cet arbre.

Recherche encapsulée

- ▶ On peut imaginer que, quand on entre dans une branche qui est fils 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* : On ne peut pas modifier la « copie de sauvegarde » de la mémoire associé au *choice point*. (pas de `cut`).

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 la valeur de X dans la première solution de l'arbre de recherche pour $\{P X\}$.
- ▶ `{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).

Exemple

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

{Browse {SearchOne Digit}}

{Browse {SearchAll Digit}}
```

Palindromes

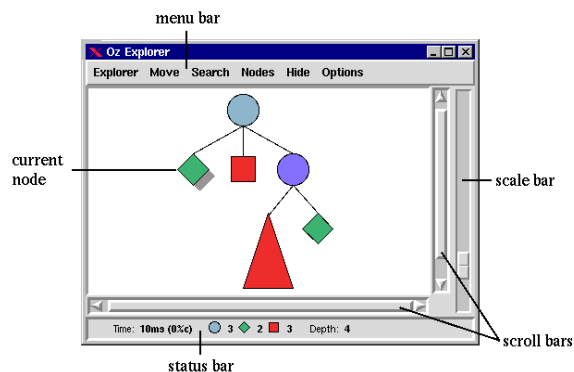
Chercher tous les nombres à quatre chiffres qui sont des palindromes, et qui sont produit de deux nombres à deux chiffres.

```
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
```

L'explorateur

- ▶ L'explorateur est un outil interactif pour étudier l'arbre de recherche
- ▶ Il utilise des couleurs différents pour indiquer l'état d'un nœud :
 - ▶ vert : succès
 - ▶ rouge : échec
 - ▶ bleu : nœud de choix qui n'a pas échoué
- ▶ Cliquer sur un nœud : affiche les valeurs des variables locales dans une fenêtre *Inspector*.
- ▶ On peut demander de continuer la recherche.

Interface graphique



Continuation de l'exemple

```
{ExploreOne Digit}
```

```
{ExploreAll Digit}
```

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}`.

Append avec résultat en troisième argument

```
declare  
fun {Append A B}  
  case A  
  of nil then B  
  [] X|As then X|{Append As B}  
  end  
end
```

Append avec résultat en premier argument

```
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
```

Append non-directionnel (comme en Prolog)

```
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
```

Application

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