Programmation Logique et Par Contraintes Avancee
Cours 4 { Programmation logique en Oz

Ralf Treinen

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

Contenu cours 4

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 utiliser si on en a besoin.
- Permet d'exprimer plusieurs possibilites de continuer le calcul (non-determinisme).

Deux types de non-determinisme

- Non-determinisme ≪ don't care ≫ :
  - Toutes les alternatives sont bonnes et mênent a un resultat equivalent.
  - Il su t de laisser la machine faire son choix.
  - Exemple : Ordonnancement des threads on Oz.
  - Aussi appele *indeterminisme*
- Non-determinisme ≪ don't know ≫
  - On ne sait pas quel est le bon choix parmi les alternatives o ertes.
  - La machine doit exploiter tous les choix possibles et construire un arbre de recherche.
  - Exemple : programmation logique.

Arbre de recherche

- Operation `choice ... end` pour indiquer un *choix* entre plusieurs possibilites de continuer.
- L'execution d'un `choice` cree un *choice point* : On essaye de continuer sur la premiere alternative, si cela echoue on revient au *choice point*.
- Donne lieu a un *arbre de recherche*, les *choice points* sont les n uds dans cet arbre.

Recherche encapsulee

- On peut imaginer que, quand on entre dans une branche qui est ls d'un *choice point*, qu'on garde une copie de l'etat de memoire pour être capable de revenir.
- L'implementation peut être beaucoup plus e cace. Technique de la *Warren Abstract Machine* : defaire les modi cations de la memoire faites depuis le *choice point*.
- La recherche est *encapsulée* : On ne peut pas modi er la ≪ copie de sauvegarde ≫ de la memoire associe au *choice point*. (pas de `cut`).

`fail` et `Search`

- `fail` : fait echouer 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 echec de l'operation Tell (ajout des equations a la memoire) execute egalement un `fail`.

Obtenir les solutions

- `{SearchOne P}`, ou P est une procedure a un argument, donne la valeur de X dans la premiere solution de l'arbre de recherche pour `{P X}`.
- `{SearchAll P}`, ou P est une procedure a un argument, donne la liste des valeurs de X dans toutes les solutions de l'arbre de recherche pour `{P X}`.

On ecrit souvent la procedure P comme une fonction (a 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

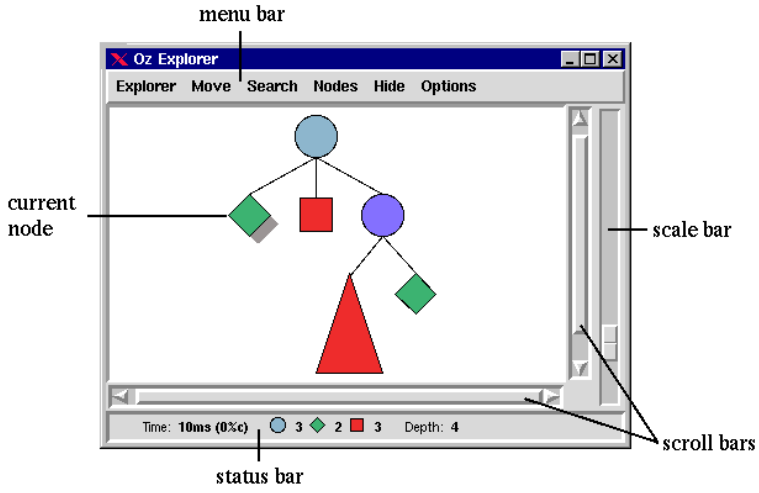
Chercher tous les nombres a quatre chi res qui sont des palindromes, et qui sont produit de deux nombres a deux chi res.

```
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 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.

Interface graphique



Continuation de l'exemple

```
{ExploreOne Digit}

{ExploreAll Digit}
```


Explorer un arbre de recherche

- `{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}`.

Append avec resultat en troisieme argument

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


Append avec resultat 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
```

```
{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 en
```

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