

Programmation Logique et Par Contraintes Avancée

Cours 2 – Le modèle d'exécution d'Oz

Ralf Treinen

Université Paris Cité
UFR Informatique
Institut de Recherche en Informatique Fondamentale



treinen@irif.fr

15 janvier 2025

Procédures et Fonctions

- ▶ Au cours 1 on a vu seulement les procédures.
- ▶ Les fonctions existent aussi dans la syntaxe du langage ; elles sont simplement un raccourci pour les procédures.
- ▶ On peut définir une fonction et l'utiliser comme une procédure, ou définir une procédure et l'utiliser comme une fonction.

Contenu cours 2

Compléments Oz de base

Procédures et Fonctions

Déclarations de variables

Lambda-expressions

Langage noyau : mini-Oz

La machine abstraite

Les règles d'exécution

Définition d'une fonction

- ▶ Syntaxe définition d'une fonction :

```
fun {F X1 ... Xn} i1 ... im expr end
```

Dans le corps il y a une séquence éventuellement vide d'instructions $i_1 \dots i_m$, suivie d'une expression expr qui calcule la valeur de retour.

- ▶ C'est est une abréviation pour

```
proc {F X1 ... Xn R} i1 ... im R=expr end
```

Exemple : Définition d'une fonction

La définition de la fonction

```
fun {Max X Y} if X > Y then X else Y end end
```

est une abréviation pour

```
proc {Max X Y R} if X>Y then R=X else R=Y end end
```

Appeler une procédure comme une fonction

- ▶ Quand P est une procédure à $n + 1$ argument, on peut l'utiliser comme une fonction à n arguments dans un contexte où une expression est attendue :

```
context[{P X1 ... Xn}]
```

- ▶ C'est une abréviation pour

```
local R in  
  {P X1 ... Xn R}  
  context[R]  
end
```

Exemples (fonctions2.oz)

```
% use a fonction as a procedure  
local F Z in  
  fun {F X Y} X+Y end  
  {F 1 2 Z}  
  {Browse Z}  
end
```

Exemples (fonctions1.oz)

```
% use a procedure as a function  
local P in  
  proc {P X Y Z} Z=f(X Y) end  
  {Browse {P a b}}  
end
```

Procédures et Fonctions

- ▶ Une fonction est une abréviation pour une procédure.
- ▶ Un appel de procédure est une instruction, un appel de fonction est une expression.
- ▶ Si la procédure P prend $n + 1$ arguments alors la fonction P prend n arguments.
- ▶ Pas d'application partielle.

Déclaration avec Binding

- ▶ On peut mettre certaines instructions entre `local` et `in`, ou après un `declare` : Des équations et des définitions de procédures (ou fonctions).
- ▶ Effet : sont déclarées toutes les variables sur les côtés gauches des équations, et la variable directement après `proc/fun`.
- ▶ C'est un idiome fréquent en Oz, en particulier pour définir des fonctions/procédures locales.

Exemples (fonctions3.oz) I

```
% l'exemple d'insertion dans les arbres de recherche
% maintenant comme fonction
declare FunInsert
fun {FunInsert Key Value TreeIn}
  case TreeIn
  of nil then tree(Key Value nil nil)
  [] tree(K1 V1 T1 T2) then
    if Key == K1 then tree(Key Value T1 T2)
    elseif Key < K1 then
      tree(K1 V1 {FunInsert Key Value T1} T2)
    else
      tree(K1 V1 T1 {FunInsert Key Value T2})
    end
  end
end
end
```

Exemples (binding1.oz) I

```
% proc performs a binding
local
  proc {P Y} {Browse Y+Y} end
  X = 5
in
  {P X}
end

% raccourci pour :
local
  P X
in
  proc {P Y} {Browse Y+Y} end
  X = 5
  {P X}
end
```

Exemples (binding2.oz) I

```
% several variables on the left hand side
local
  f(X Y) = f(1 2)
in
  {Browse g(X Y)}
end

% raccourci pour
local
  X Y
in
  f(X Y) = f(1 2)
  {Browse g(X Y)}
end
```

Exemples (lambda.oz)

```
declare
fun {Map F L}
  case L of
    nil then nil
    [] H|R then {F H} | {Map F R}
  end
end

{Browse {Map fun {$ X} X+X end [1 2 3 4]}}
```

Procédures et fonctions anonymes

- ▶ On peut utiliser le symbole \$ à la première position de l'entête d'une définition de procédure/fonction.
- ▶ Effet : on obtient une *expression* dont la valeur est la procédure/fonction.
- ▶ Cela correspond à des lambda-expressions :

function x -> x+x	OCaml
lambda x: x+x	Python
$\lambda x.x + x$	Math (lambda-calcul)
fun {\$ X} X+X end	Oz

Syntaxe de *mini-Oz*

Langage noyau, utilisé seulement pour définir la sémantique.

Une instruction $\langle s \rangle$ peut être :

- ▶ skip
- ▶ $\langle s \rangle_1 \langle s \rangle_2$
- ▶ local $\langle x \rangle$ in $\langle s \rangle$ end
- ▶ $\langle x \rangle = \langle t \rangle$ ($\langle t \rangle$ peut être une procédure)
- ▶ if $\langle x \rangle$ then $\langle s \rangle_1$ else $\langle s \rangle_2$ end
- ▶ case $\langle x \rangle$ of $\langle p \rangle$ then $\langle s \rangle_1$ else $\langle s \rangle_2$ end
- ▶ { $\langle x \rangle \langle y \rangle_1 \dots \langle y \rangle_n$ }

Syntaxe de mini-Oz

Un terme peut être :

- ▶ un identificateur x
- ▶ un terme composé $f(<t>_1 \dots <t>_n)$
- ▶ une procédure `proc { $\$ y_1 \dots y_n$ } <s> end`

Remarques :

- ▶ Ici : seulement termes classiques, pas d'enregistrements (mais l'algorithme d'unification se généralise facilement aux enregistrements)
- ▶ Les fonctions sont une abréviation pour les procédures.

Raccourcis syntaxiques (2)

`declare`

`declare X in <i>`

peut être traduit (pour un programme complet) en

`local X in <i> end`

Raccourcis syntaxiques (1)

Le langage Oz complet peut être traduit en mini-Oz :

`local` avec plusieurs variables

`local X Y Z in <i> end`

peut être traduit en

```
local X in
  local Y in
    local Z in
      <i>
    end
  end
end
```

Raccourcis syntaxiques (3)

`local` avec affectation d'une valeur

`local proc {P X} <i1> end in <i2> end`

peut être traduit en

```
local P in
  P = proc {$ X} <i1> end
  <i2>
end
```

Raccourcis syntaxiques (4)

utiliser des termes à la place de variables

```
if <t> then <i1> else <i2> end
```

peut être traduit en

```
local X in  
  X = <t>  
  if X then <i1> else <i2> end  
end
```

(où X est un nouvel identificateur)

Modèles de mémoire

1. à *valeur* : toute variable a une valeur, et cette valeur ne change pas pendant l'exécution du programme (langages fonctionnels purs : OCaml etc.)
2. à *cellule* : la valeur d'une variable peut changer pendant l'exécution d'un programme (langages impératifs : C, Pascal, C++, etc.)
3. à *affectation unique* : Une variable peut être non liée, ou liée à une valeur. Une fois créée, la liaison d'une variable ne change plus (langages logiques et/ou à contraintes : Prolog, Oz, etc.)

Raccourcis syntaxiques (5)

Les instructions d'unification

```
s=t
```

peut être traduit en

```
local X in  
  X = s  
  X = t  
end
```

Valeurs partielles

Si on a à la fois

- ▶ une mémoire à affectation unique
- ▶ des valeurs hiérarchiques (arbres, etc.)

alors on obtient un langage à affectation unique avec des valeurs *partielles* : une variable peut être liée à une valeur qui contient des variables, par exemple $X = f(Y, g(a, Z))$.

Exemples (valeurs.oz) I

```
declare X Y Z in
X = f(Y Z)
{Browse X}

Y=g(4 5 6)

% effect of adding an equation
declare X Y Z in
{Browse [X Y Z]}

X = f(Y Z)

X = f(a g(b))

% infinite trees / graphes cycliques
local X in X=f(X) {Browse X} end
```

Exemples (lexical.oz)

```
% liaison statique
local X P in
  X = 17
  proc {P} {Browse X} end
  local X in
    X = 42
    {P}
  end
end
```

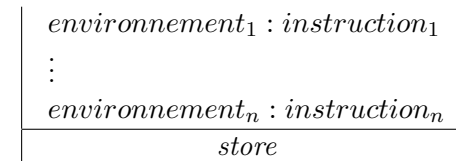
Composantes de la machine abstraite

1. Une mémoire (angl. : *store*). En général, la mémoire contient une contrainte en forme résolue : pour l'instant des liaisons de variables vers des termes (voir le cours 3).
2. Une pile de paires (environnement, instruction).

Précisions :

- ▶ Un environnement lie des *identificateurs* (qui paraissent dans le programme) à des *variables* (qui existent dans la mémoire).
L'environnement est nécessaire pour la gestion des identificateurs locales et de la liaison statique.
- ▶ Les procédures sont représentées dans la mémoire comme des clôtures (liaison statique!).

Représentation graphique



Exécution d'un programme

- ▶ État initial pour l'exécution d'un programme s :

$\emptyset : s$
\emptyset

- ▶ État terminal :

σ

Exécution : le cas du skip

$E : \text{skip}$ $reste$
σ

 \Rightarrow

$reste$
σ

La mémoire

Dans un premier temps, la mémoire est un système d'équations résolues :

$$\begin{aligned}x_1 &= t_1 \\ &\vdots \\ x_n &= t_n\end{aligned}$$

où toutes $x_i \neq x_j$ pour $i \neq j$. On peut le voir comme

- ▶ liaison des variables à des valeurs partielles
- ▶ les cycles sont permises!

Exécution : le cas d'une composition

$E : s_1 \ s_2$ $reste$
σ

 \Rightarrow

$E : s_1$ $E : s_2$ $reste$
σ

Exécution : le cas d'une portée locale

$$\frac{\left| \begin{array}{l} E : \text{local } X \text{ in } s \text{ end} \\ \text{reste} \end{array} \right|}{\sigma} \Rightarrow \frac{\left| \begin{array}{l} E \cup \{X \mapsto x\} : s \\ \text{reste} \end{array} \right|}{\sigma}$$

- ▶ où x est une nouvelle variable.
- ▶ $E \cup F$ est la mise à jour de l'environnement E par l'environnement F . Si l'identificateur X est lié à la fois par E et par F alors $E \cup F$ lie X à $F(X)$.
- ▶ La mémoire reste inchangée car il n'y a pas encore de liaison pour x .

Exécution : le cas d'une équation $X = t$ (2)

$$\frac{\left| \begin{array}{l} E : X = t \\ \text{reste} \end{array} \right|}{\sigma} \Rightarrow \perp$$

- ▶ Soit $E(X = t)$ obtenu par remplaçant tout identificateur Y par $E(Y)$.
- ▶ Si $\sigma \cup \{E(X = t)\}$ n'a pas de solution.

Exécution : le cas d'une équation $X = t$ (1)

$$\frac{\left| \begin{array}{l} E : X = t \\ \text{reste} \end{array} \right|}{\sigma} \Rightarrow \frac{\left| \begin{array}{l} \text{reste} \\ \sigma' \end{array} \right|}{\sigma}$$

- ▶ Soit $E(X = t)$ obtenu par remplaçant tout identificateur Y par $E(Y)$.
- ▶ Si σ' est la forme résolue de $\sigma \cup \{E(X = t)\}$.
- ▶ σ' peut lier des variables à des nouvelles clôtures (avec environnement E).

Exécution : le cas du if (1)

$$\frac{\left| \begin{array}{l} E : \text{if } X \text{ then } s_1 \text{ else } s_2 \text{ end} \\ \text{reste} \end{array} \right|}{\sigma} \Rightarrow \frac{\left| \begin{array}{l} E : s_1 \\ \text{reste} \end{array} \right|}{\sigma}$$

- ▶ Si $\sigma(E(X)) = \text{true}$

Exécution : le cas du `if` (2)

$$\frac{\begin{array}{|l} E : \text{if } X \text{ then } s_1 \text{ else } s_2 \text{ end} \\ \text{reste} \\ \hline \sigma \end{array}}{\Rightarrow \frac{\begin{array}{|l} E : s_2 \\ \text{reste} \\ \hline \sigma \end{array}}$$

- ▶ Si $\sigma(E(X)) = \text{false}$

Exécution : le cas du `if` (4)

- ▶ Pourquoi un 4ème cas??
- ▶ $\sigma(E(X))$ peut être une variable!
- ▶ Pas de règle de transformation pour ce cas : suspension du fil d'exécution!

Exécution : le cas du `if` (3)

$$\frac{\begin{array}{|l} E : \text{if } X \text{ then } s_1 \text{ else } s_2 \text{ end} \\ \text{reste} \\ \hline \sigma \end{array}}{\Rightarrow \perp}$$

- ▶ Si $\sigma(E(X))$ est une valeur $\notin \{\text{true}, \text{false}\}$

Exemples (if.oz)

```
% suspension du if
declare X Y
if X then Y=17 else Y=42 end
{Browse Y}

X=false
```

Exécution : le cas du case

- ▶ Similaire au `if`, mais peut étendre l'environnement (par les identificateurs du motif) et la mémoire (pour les variables liées aux identificateurs du motif).
- ▶ Le calcul avance si on peut conclure que *toutes* les « valeurs possibles » de $E(X)$ sont filtrées par le motif (cas `then`), ou si on peut conclure qu'*aucune* « valeur possible » de $E(X)$ est filtrée par p (cas `else`).
- ▶ Sinon : suspension du fil.
- ▶ Voir la semaine prochaine !

Exécution : Appel d'une procédure (1)

$$\begin{array}{|l}
 E : \{X \ Y_1 \ \dots \ Y_n\} \\
 \hline
 \text{reste} \\
 \hline
 \sigma
 \end{array}
 \Rightarrow
 \begin{array}{|l}
 F \cup \{Z_1 \mapsto E(Y_1), \dots, Z_n \mapsto E(Y_n)\} : s \\
 \hline
 \text{reste} \\
 \hline
 \sigma
 \end{array}$$

- ▶ Si $\sigma(E(X))$ est la clôture (F, s) avec les paramètres Z_1, \dots, Z_n

Exemples (case.oz)

```

% suspension du case
declare X Y Z
case X of
  f (g (V)) then Z=17
else Z=42
end

{Browse Z}

X=f (Y)

local Y1 in Y=g (Y1) end
    
```

Exécution : Appel d'une procédure (2)

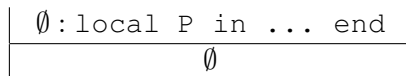
$$\begin{array}{|l}
 E : \{X \ Y_1 \ \dots \ Y_n\} \\
 \hline
 \text{reste} \\
 \hline
 \sigma
 \end{array}
 \Rightarrow \perp$$

- ▶ Si $\sigma(E(X))$ est une valeur qui n'est pas une clôture à n arguments

Exécution : Appel d'une procédure (3)

- ▶ Un appel de procédure $\{ X Y_1 \dots Y_n \}$ suspend quand $\sigma(E(X))$ est une variable.

Configuration Initiale



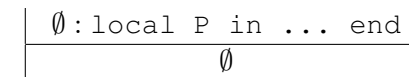
- ▶ sur la pile : le programme complet dans un environnement vide
- ▶ mémoire vide

Exemples (example.oz)

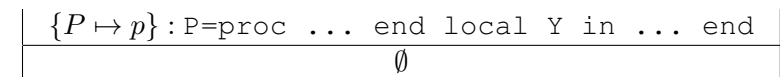
```
% L'exemple pour l'exécution
```

```
local P in  
  P = proc { $ X } X=1 end  
  local Y in  
    { P Y }  
    Y=2  
  end  
end
```

Exécution du local P



⇓



- ▶ p est une nouvelle variable

Décomposition de l'instruction composée

$$\frac{\{P \mapsto p\} : P = \text{proc } \dots \text{ end local } Y \text{ in } \dots \text{ end}}{\emptyset}$$

↓

$$\frac{\begin{array}{l} \{P \mapsto p\} : P = \text{proc } \dots \text{ end} \\ \{P \mapsto p\} : \text{local } Y \text{ in } \dots \text{ end} \end{array}}{\emptyset}$$

Traiter l'équation pour P

$$\frac{\begin{array}{l} \{P \mapsto p\} : P = \text{proc } \dots \text{ end} \\ \{P \mapsto p\} : \text{local } Y \text{ in } \dots \text{ end} \end{array}}{\emptyset}$$

↓

$$\frac{\begin{array}{l} \{P \mapsto p\} : \text{local } Y \text{ in } \dots \text{ end} \\ p = \langle P \mapsto p; \text{proc } \dots \text{ end} \rangle \end{array}}$$

- ▶ La valeur de la variable p est une clôture

Exécution du local Y

$$\frac{\{P \mapsto p\} : \text{local } Y \text{ in } \dots \text{ end}}{p = \langle P \mapsto p; \text{proc } \dots \text{ end} \rangle}$$

↓

$$\frac{\begin{array}{l} \{P \mapsto p, Y \mapsto y\} : \{P \ Y\} \ Y=2 \\ p = \langle P \mapsto p; \text{proc } \dots \text{ end} \rangle \end{array}}$$

- ▶ y est une nouvelle variable

Décomposition de l'instruction composée

$$\frac{\begin{array}{l} \{P \mapsto p, Y \mapsto y\} : \{P \ Y\} \ Y=2 \\ p = \langle P \mapsto p; \text{proc } \dots \text{ end} \rangle \end{array}}$$

↓

$$\frac{\begin{array}{l} \{P \mapsto p, Y \mapsto y\} : \{P \ Y\} \\ \{P \mapsto p, Y \mapsto y\} : Y=2 \\ p = \langle P \mapsto p; \text{proc } \dots \text{ end} \rangle \end{array}}$$

Appel de la procédure P

$$\frac{\begin{array}{l} \{P \mapsto p, Y \mapsto y\} : \{P \ Y\} \\ \{P \mapsto p, Y \mapsto y\} : Y=2 \end{array}}{p = \langle P \mapsto p; \text{proc } \dots \text{ end} \rangle} \Downarrow \frac{\begin{array}{l} \{P \mapsto p, X \mapsto y\} : X=1 \\ \{P \mapsto p, Y \mapsto y\} : Y=2 \end{array}}{p = \langle P \mapsto p; \text{proc } \dots \text{ end} \rangle}$$

- ▶ L'environnement lie P à p
- ▶ La mémoire donne pour p une clôture
- ▶ On remplace l'appel par le corps de la procédure
- ▶ Environnement du corps : environnemnt de la clôture plus liaison

Traiter l'équation

$$\frac{\begin{array}{l} \{P \mapsto p, Y \mapsto y\} : Y=2 \\ p = \langle P \mapsto p; \text{proc } \dots \text{ end} \rangle \\ y = 1 \end{array}}{\perp}$$

- ▶ l'environnement lie Y à la variable y
- ▶ $y = 1 \wedge y = 2$ est contradictoire

Traiter l'équation

$$\frac{\begin{array}{l} \{P \mapsto p, X \mapsto y\} : X=1 \\ \{P \mapsto p, Y \mapsto y\} : Y=2 \end{array}}{p = \langle P \mapsto p; \text{proc } \dots \text{ end} \rangle} \Downarrow \frac{\begin{array}{l} \{P \mapsto p, Y \mapsto y\} : Y=2 \\ p = \langle P \mapsto p; \text{proc } \dots \text{ end} \rangle \\ y = 1 \end{array}}$$

Ce qui reste à comprendre

- ▶ Mécanisme de résolution (lors d'une tentative d'ajout d'une équation à la mémoire)
- ▶ Mécanisme de suspension : quand peut-on conclure qu'on a suffisamment d'information dans la mémoire pour aller dans une branche d'une conditionnelle ou d'une distinction de cas ?
- ▶ Le modèle de concurrence.
- ▶ Voir la semaine prochaine !