

# Programmation Fonctionnelle... effective!!!

Pierre Bourreau

<sup>1</sup>IUT Bordeaux 1 - Licence Pro

February 24, 2010

## Introduction

Introduction

## Le $\lambda$ -calcul simplement typé

Le  $\lambda$ -calcul  
simplement typé

En (très) bref

En (très) bref

Syntaxe

Syntaxe

Typage simple

Typage simple

## CaML

CaML

Un langage interprété

Un langage interprété

Types

Types

Opérations

Opérations

Structure d'un programme

Structure d'un programme

Types Complexes

Types Complexes

## Conclusion

Conclusion

## Introduction

### Le $\lambda$ -calcul simplement typé

- En (très) bref
- Syntaxe
- Typage simple

### CaML

- Un langage interprété
- Types
- Opérations
- Structure d'un programme
- Types Complexes

## Conclusion

## Introduction

### Le $\lambda$ -calcul simplement typé

- En (très) bref
- Syntaxe
- Typage simple

### CaML

- Un langage interprété
- Types
- Opérations
- Structure d'un programme
- Types Complexes

## Conclusion

# Langages Fonctionnels vs Langages Impératifs

CaML

Pierre Bourreau

## Impératif

```
int factorielle (int n)
{
    res = 0;
    for (int i=1; i<=n; i++)
        res=res*i;
    return res;
}
```

## Fonctionnel

```
let rec factorielle n = function
  0 -> 1
  | n -> n * factorielle (n-1);;
```

Introduction

Le  $\lambda$ -calcul  
simplement typé

En (très) bref

Syntaxe

Typage simple

CaML

Un langage interprété

Types

Opérations

Structure d'un programme

Types Complexes

Conclusion

## Impératif

- ▶ langage riche (nombreux mots clés . . . )
- ▶ typage à la main
- ▶ allocation mémoire (old school -> C, C++)
- ▶ programmes de tailles conséquentes

## Fonctionnel

- ▶ langage pauvre
- ▶ typage automatique
- ▶ pas d'allocation mémoire (*garbage collector*)
- ▶ programmes petits

Introduction

Le  $\lambda$ -calcul  
simplement typé

En (très) bref

Syntaxe

Typage simple

CaML

Un langage interprété

Types

Opérations

Structure d'un programme

Types Complexes

Conclusion

## Présentation

- ▶ (Objective) Categorical Abstract Machine Language (87)
- ▶ Héritier de ML
- ▶ Langage fonctionnel
- ▶ Garbage collector
- ▶ **Typage fort et inféré**
- ▶ Extension pour commandes impératives + objet
- ▶ Utilisé en entreprise: Airbus, Microsoft, Intel, ...

### Introduction

Le  $\lambda$ -calcul  
simplement typé

En (très) bref

Syntaxe

Typage simple

### CaML

Un langage interprété

Types

Opérations

Structure d'un programme

Types Complexes

### Conclusion

## Introduction

### **Le $\lambda$ -calcul simplement typé**

En (très) bref

Syntaxe

Typage simple

### **CaML**

Un langage interprété

Types

Opérations

Structure d'un programme

Types Complexes

### **Conclusion**

Introduction

**Le  $\lambda$ -calcul  
simplement typé**

En (très) bref

Syntaxe

Typage simple

**CaML**

Un langage interprété

Types

Opérations

Structure d'un programme

Types Complexes

Conclusion

- ▶ **1930's** Alonzo Church
  - ▶ Théorie de la calculabilité (le  $\lambda$ -calcul est un modèle de la calculabilité)
- ▶ **1958** Lisp
- ▶ **1970's** Scheme
- ▶ **1980's** ML
- ▶ **1987** CaML - **1996** OCaml

## Définition

Soit  $\mathcal{V}$  un ensemble énumérable de variables. On définit l'ensemble  $\Lambda$  des termes construits sur  $\mathcal{V}$ :

- ▶ si  $x \in \mathcal{V}$ , alors  $x \in \Lambda$
- ▶ si  $M \in \Lambda$  et  $x \in \mathcal{V}$ , alors  $\lambda x.M \in \Lambda$
- ▶ si  $M, N \in \Lambda$ , alors  $MN \in \Lambda$

## Exemples

- ▶  $x, x_1, y, z \in \mathcal{V}$
- ▶  $\lambda x.x$
- ▶  $yx$
- ▶  $\lambda x.yx$

- ▶ comprendre les variables comme des fonctions
- ▶ écrire  $y x_1 \dots x_n$  signifie que  $y$  est une fonction  $n$ -aire
- ▶ le terme  $\lambda x_1 \dots x_n. x_i$ , est une fonction  $n$ -aire qui renvoie son  $i$ -ème argument.
- ▶  $\lambda x_1 \dots x_n. M$  est une fonction à  $n$  arguments
- ▶ c'est aussi une fonction à  $n - 1$  arguments,  $n - 2$  arguments, etc ...

## Exemples

- ▶  $\lambda x. \text{factorielle } x$  est une fonction à un argument

- ▶ associer un type à chaque variable
- ▶ une variable ne peut avoir qu'un seul type (*est-ce le cas pour tous les langages impératifs?*)

## Exemple

- ▶  $x$
- ▶  $\lambda x.fx$
- ▶  $\lambda x.f(\lambda y.xy)$

- ▶ associer un type à chaque variable
- ▶ une variable ne peut avoir qu'un seul type (*est-ce le cas pour tous les langages impératifs?*)

## Exemple

- ▶  $x : a$
- ▶  $\lambda x^a.f^{a \rightarrow b} x^a : a \rightarrow b$
- ▶  $\lambda x.f(\lambda y.xy)????$

- ▶ associer un type à chaque variable
- ▶ une variable ne peut avoir qu'un seul type (*est-ce le cas pour tous les langages impératifs?*)

## Exemple

- ▶  $x : a$
- ▶  $\lambda x^{a \rightarrow b}. f^{a \rightarrow b} x^a : a \rightarrow b$
- ▶  $\lambda x^{a \rightarrow b}. f^{(a \rightarrow b) \rightarrow c} (\lambda y^a. x^{a \rightarrow b} y^a) : a \rightarrow c$

- ▶ Vous faites de l'**inférence de type**.
- ▶ On pourrait n'utiliser qu'un type  $a$ .
- ▶ Mais ne donne pas le type le plus général.
- ▶ On ne veut donner aucune contrainte sur les types à utiliser.
- ▶ **En CaML, le système de typage est fort: certaines opérations ont un type contraint (+, -, ...)**

# Sommaire

## Introduction

### Le $\lambda$ -calcul simplement typé

- En (très) bref
- Syntaxe
- Typage simple

### CaML

- Un langage interprété
- Types
- Opérations
- Structure d'un programme
- Types Complexes

## Conclusion

CaML

Pierre Bourreau

Introduction

Le  $\lambda$ -calcul  
simplement typé

En (très) bref

Syntaxe

Typage simple

CaML

Un langage interprété

Types

Opérations

Structure d'un programme

Types Complexes

Conclusion

- ▶ repose sur le  $\lambda$ -calcul simplement typé:
- ▶ système d'inférence de type **statique**
- ▶ permet de vérifier qu'une fonction est bien construite dès compilation - interprétation
- ▶ permet aussi de définir des fonctions d'ordre supérieur

- ▶ depuis la console, lancer `ocaml`
- ▶ l'interprète se lance; écrire une expression CaML
  - ▶ **# let** `x = 2 + 6 ;;`
- ▶ l'interprète analyse l'expression; si correcte, infère son type:
  - ▶ `val x : int = 7`

## Types

- ▶ `int`: entier signé (4, 3445, -234)
- ▶ `float`: nombre à virgule flottante double- (≈ type double du C) (4., 34.45, -.234)
- ▶ `bool`: booléen (`true`, `false`)
- ▶ `char`: caractère sur 8 bits ('a', 'v')
- ▶ `string`: chaîne de caractères ("a", "3445", "hello")
- ▶ `unit` valeur unique notée ()

## Opérations

- ▶ sur les entiers: `+`, `-`, `*`, `/`, `mod`
- ▶ sur les flottants: `+. .`, `-. .`, `*. .`, `/. .`, `sqrt`, `e`
  - ▶ conversion: `int_of_float`, `float_of_int`
  - ▶ `4.`
- ▶ sur les booléens: `&&`, `||`, `not`
- ▶ sur les chaînes: concaténation `^`
- ▶ une opération spéciale: `if val_bool then val1  
else val2;;`

## Globales

- ▶ `let x = 4;;`
- ▶ `let y = float_of_int(x) +. 6.;;`

## Locales

- ▶ `let z=3 in z+4;;`
  - ▶ `- : int = 7`
- ▶ `z` n'est plus accessible par la suite.

## Non-récurrentes

```
let func arg1 ...arg2 = corps;;
```

- ▶ `let norme x y =  
 sqrt (x*.x+.y*.y) ;;`
  - ▶ `norme 4. 6.;;`
- ▶ `let somme a b =  
 a + b;;`
  - ▶ `somme 1 8;;`

## Récurives

```
let rec func arg1 ...arg2 = corps;;
```

- ▶ 

```
let rec fact n =  
  if n<=1 then 1 else n * fact(n-1);;  
  ▶ fact 4;;
```
- ▶ 

```
let rec mult a b =  
  if b=0 then 0  
  else somme a (mult a (b-1));;  
  ▶ mult 2 3;;
```

[Introduction](#)[Le  \$\lambda\$ -calcul  
simplement typé](#)[En \(très\) bref](#)[Syntaxe](#)[Typage simple](#)[CaML](#)[Un langage interprété](#)[Types](#)[Opérations](#)[Structure d'un programme](#)[Types Complexes](#)[Conclusion](#)

## Les listes

- ▶ Prédéfinies (héritage de Lisp)
- ▶ # [1;4;7;3];;
- ▶ **Tous les éléments doivent être de même type.**
- ▶ Opérations:
  - ▶ insertion d'un élément en tête: `elt::liste;;`
  - ▶ récupérer élément de tête: `List.hd liste;;`
  - ▶ récupérer la queue d'une liste: `List.tl liste;;`

## Pattern Matching

- ▶ le type est défini par induction
  - ▶ Listes -> cas de base: []
  - ▶ Listes -> pas d'induction: `a::l`
- ▶ l'opérateur `match`:
  - ▶ `match` var `with`
    - | `cas1` -> `action1`
    - ...
    - | `casN` -> `actionN;;`
  - ▶ `# let rec` `taille l = match l with`
    - | [] -> 0
    - | `a::k` -> `taille k +1;;`

[Introduction](#)[Le  \$\lambda\$ -calcul  
simplement typé](#)[En \(très\) bref](#)[Syntaxe](#)[Typage simple](#)[CaML](#)[Un langage interprété](#)[Types](#)[Opérations](#)[Structure d'un programme](#)[Types Complexes](#)[Conclusion](#)

# Sommaire

## Introduction

### Le $\lambda$ -calcul simplement typé

- En (très) bref
- Syntaxe
- Typage simple

### CaML

- Un langage interprété
- Types
- Opérations
- Structure d'un programme
- Types Complexes

## Conclusion

CaML

Pierre Bourreau

Introduction

Le  $\lambda$ -calcul  
simplement typé

En (très) bref

Syntaxe

Typage simple

CaML

Un langage interprété

Types

Opérations

Structure d'un programme

Types Complexes

Conclusion

## Introduction

Le  $\lambda$ -calcul  
simplement typé

En (très) bref

Syntaxe

Typage simple

## CaML

Un langage interprété

Types

Opérations

Structure d'un programme

Types Complexes

## Conclusion

- ▶ langage très puissant.
- ▶ programme très concis
- ▶ langages fonctionnels < langages impératifs?
- ▶ s'appuie sur une théorie bien fondée, et qui évolue (système F, types intersections, ...)
- ▶ Important: les langages changent, la théorie reste...