Approches Fonctionnelles de la Programmation

Introduction

Didier Verna

EPITA / LRDE

didier@lrde.epita.fr

lrde/~didier

@didierverna

didier.verna

in/didierverna

Plan

Plan

1930 : Lambda-Calcul (Alonzo Church)

- Système formel de calcul *(Top-Down)*
- Règles
  - Variable : x
  - Abstraction : $(\lambda x.M)$
  - Application : (MN)
- Opérateurs
  - α-conversion : $(\lambda x.M[x]) \rightarrow (\lambda y.M[y])$
  - β-reduction : $((\lambda x.M)E) \rightarrow (M[x := E])$
- 1937 : Preuve de la Turing-complétude
  - [Turing, 1937]
- 1920 / 1930 : Logique Combinatoire, équivalent théorique
  - [Curry, 1958]

Princeton

1958 : Lisp (John McCarthy)

- Origine (LISP) [McCarthy, 1960]
  - MIT AI Lab, IBM 700/7000
  - 2e plus ancien langage de haut niveau
  - Premier language fonctionnel
  - lambda-calcul + recursion
  - Multi-paradigme / Homoïconique
- Dialectes / Chronologie
  - 1975 : Scheme / Lisp Machines
  - 1980 : Common Lisp (ANSI Standard 1994)
  - 1985 : Emacs Lisp
  - 2005 : Clojure
  - 2010 : Racket

Futura Science

Autres Langages

- Historiques
  - IPL (1956) *Niveau assembleur, très impératif*
  - APL (Iverson, 1960), FP (Backus, 1977) [Backus, 1978]
  - ML (Milner, 1973), Standard ML, Caml / OCaml (Leroy, 1996)
  - Miranda (Turner, 1985), Haskell (1990)
- Plus récents *Mix Objet / Fonctionnel*
  - Scala (Odersky, 2004)
  - F# (M$, 2005)
- Dans le mix
  - Python, Ruby, JavaScript, Julia, *etc.*
- Retour vers le fonctionnel
  - Lambda-expressions dans C++11 / Java 8

Plan

Un Paradigme de Programmation

- **Rappels sur la notion de paradigme**
  - Affecte l'*expressivité* d'un langage
  - Affecte la manière de *penser* dans un langage
  - Porosité
- **Le paradigme fonctionnel**
  - Expression (*vs.* Instruction)
  - Évaluation (*vs.* Exécution)

« Quoi faire » plutôt que « Comment faire »

De l'Impératif au Fonctionnel

« La somme des carrés des entiers entre 1 et N »

C (impératif)

```
int ssq (int n)
{
  int i = 1, a = 0;
 loop:
  a += i*i; i += 1;
  if (i <= n+1)
    goto loop;

  return a;
}
```

C (récursif)

```
int ssq (int n)
{
  if (n == 1)
    return 1;
  else
    return n*n + ssq (n-1);
}
```

Lisp

```
(defun ssq (n)
  (if (= n 1)
      1
      (+ (* n n) (ssq (1- n)))))
```

Haskell

```
ssq :: Int -> Int
ssq 1 = 1
ssq n = n*n + ssq (n-1)
```

- Clarté
- Concision

De l'Impératif à l'Envers

« La racine carrée de la somme des carrés de a et de b »

C (impératif)

```c
float hypo (float a, float b)
{
  float a2 = a*a;
  float b2 = b*b;
  float s = a2 + b2;

  return sqrt (s);
}
```

Haskell

```haskell
hypo :: Float -> Float -> Float
hypo a b = sqrt (a*a + b*b)
```

Lisp

```lisp
(defun hypo (a b)
  (sqrt (+ (* a a) (* b b))))
```

C (moins impératif)

```c
float hypo (float a, float b)
{
  return sqrt (a*a + b*b);
}
```

Mais...

Haskell (100% préfixe)

```haskell
hypo :: Float -> Float -> Float
hypo a b = sqrt ((+) ((*) a a) ((*) b b))
```

Plan

Fonctions d'Ordre Supérieur

Aka 1er ordre, 1re classe

- **Christopher Strachey** [Strachey, 72]
  - nommage (variables)
  - agrégation (structures)
  - argument de fonction
  - retour de fonction
  - manipulation anonyme
  - construction dynamique
  - ...
- **Intérêt :** plus d'expressivité (clarté, concision, *etc.*)
- Exemple : « mapping »

Lisp

```
(mapcar #'sqrt '(1 2 3 4 5))
```

Haskell

```
map sqrt [1..5]
```

Principes d'Évaluation

- **Question :** quand calculer la valeur d'une expression ?
- **Réponses**
  1. Au préalable (Lisp, *évaluation stricte*)
  2. À la demande (Haskell, *évaluation paresseuse*)
- Exemple :

Lisp

```
(defun intlist (s)                    ;; KO
  (cons s (intlist (1+ s))))
```

Haskell

```
intlist :: Int -> [ Int ]             -- OK
intlist s = s : intlist (s + 1)
```

- **Intérêt :** plus d'expressivité (clarté, concision, *etc.*)
- **Contrainte :** fonctionnel pur uniquement

Programmation Fonctionnelle Pure

La fonction au sens mathématique

$$ssq(x) = \begin{cases} 1 & \text{si } x = 1, \\ x^2 + ssq(x-1) & \text{sinon.} \end{cases}$$

Haskell

```
ssq :: Int -> Int
ssq 1 = 1
ssq n = n*n + ssq (n-1)
```

- **Fonction**
  - Impératif : (procédure) ensemble de calculs à *effets de bords* avec éventuellement retour d'une valeur
  - Fonctionnel Pur : calcul d'une valeur de sortie (retour) en fonction de valeurs d'entrée (arguments)
- **Variable**
  - Impératif : stockage d'information qui *varie* au cours du temps (mutation)
  - Fonctionnel Pur : valeur inconnue ou arbitraire (constante)

Intérêts de la Pureté

Pureté $\Longrightarrow$ (plus de) sureté

- **Parallélisme**
  - Cf. Erlang
- **Sémantique locale aux fonctions**
  - Tests locaux / Bugs locaux
- **Preuve de programme**

Preuves Formelles

Induction mathématique

« Prouvez-moi (s'il vous plaît) que $\forall N, ssq(N) > 0$ »

Fonctionnel pur

Haskell

```
ssq :: Int -> Int
ssq 1 = 1
ssq n = n*n + ssq (n-1)
```

- C'est vrai au rang 1
- Supposons que ce soit vrai au rang $N - 1$...

Impératif

C

```
int ssq (int n)
{
  int i = 1, a = 0;
 loop:
  a += i*i; i += 1;
  if (i <= n+1)
    goto loop;

  return a;
}
```

- Euh...

Limites du Formalisme Mathématique

Comment exprimer le concept de « racine carrée » ?

$$sqrt(x) = y | \begin{cases} y > 0 \\ y^2 = x \end{cases}$$

Lisp

```
(defun sqrtp (s x)
  (and (> s 0)
       (= (* s s) x)))
```

Lisp

```
(defun sqrt (x) ???)
```

Haskell

```
sqrtp :: Float -> Float -> Bool
sqrtp s x = s > 0 && s*s == x
```

Haskell

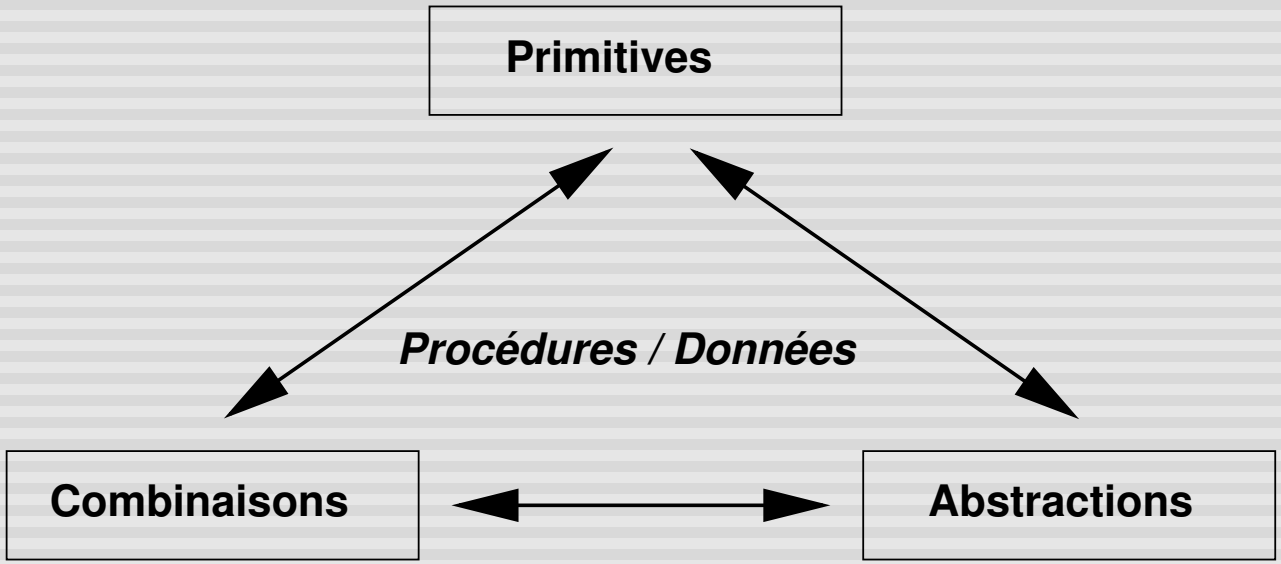
```
sqrt :: Float -> Float
sqrt x = ???
```

- Au final, il faut bien expliquer *comment faire*...
- Mais on repousse le problème : déclaratif ou impératif ?

Plan

Bénéfices de l'Approche Fonctionnelle

Les 3 caractéristiques des (bons) langages



- Moins de distinction entre procédures et données
- Plus de puissance dans la combinaison
- Plus de puissance dans l'abstraction

Plan

Bibliographie

Bibliographie

Alan M. Turing.

Computability and λ-definability.

Journal of Symbolic Logic. Cambridge University Press. 2 (4) : 153–163.

Haskell B. Curry, Robert Feys.

Combinatory Logic.

North-Holland Publishing Company.

, John McCarthy.

Recursive Functions of Symbolic Expressions and their Computation by Machine, part I.

Communications of the ACM. 3 : 184–195.

Bibliographie

John W. Backus.
Can Programming Be Liberated from the von Neumann Style ? A Functional Style and Its Algebra of Programs.
Communications of the ACM. 2(8) : 613-–641.

Joe E. Stoy and Christopher Strachey.
OS6 – An Experimental Operating System for a Small Computer.
The Computer Journal, 15(2) : 117–124.