Programmation Fonctionnelle
Introduction

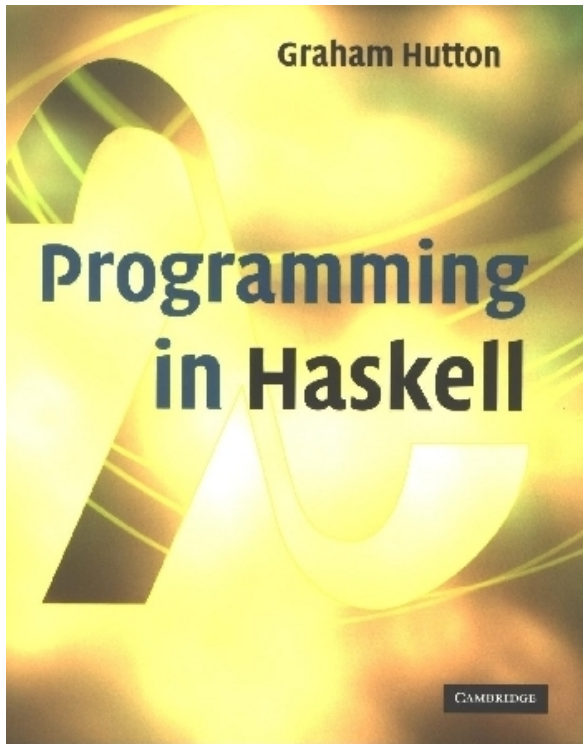
Luigi Santocanale
LIF, Aix-Marseille Université
Marseille, FRANCE

8 septembre 2013

Plan

Le cours

Cours basé sur le livre :



Bureaucratie

Calcul de la note finale :

$$\texttt{NF} = 0,6 * \texttt{ExamenTerminale} + 0,4 * \texttt{Projet}$$

Attention : le projet n'est pas rattrapable !!!

Page web du cours :

`http://www5.lif.univ-mrs.fr/ lsantoca/teaching/PF/`

La « crise » du logiciel

Comment :

- gérer la taille et la complexité des programmes modernes ?
- réduire le temps et le coût du développement logiciel ?
- accroître notre confiance qu'un programme fonctionne correctement ?

Langages de programmation

Une approche :

deviser des langages de programmation qui

- permettent que les programmes soient écrits clairement, à un haut niveau d'abstraction ;
- supportent des composantes logicielles réutilisables ;
- encouragent le recours à la vérification formelle ;
- permettent un prototypage rapide ;
- offrent des outils puissants pour résoudre les problèmes.

Langages de programmation fonctionnels :

boite à outils particulerièrement élégante permettant de réaliser ces objectifs.

Langages de programmation

Une approche :

deviser des langages de programmation qui

- permettent que les programmes soient écrits clairement, à un haut niveau d'abstraction ;
- supportent des composantes logicielles réutilisables ;
- encouragent le recours à la vérification formelle ;
- permettent un prototypage rapide ;
- offrent des outils puissants pour résoudre les problèmes.

Langages de programmation fonctionnels :

boite à outils particulerièrement élégante permettant de réaliser ces objectifs.

Une autre histoire . . .

- Langages fonctionnels développés dans le départements d'informatique théorique . . .
- (Au début) pas de succès en dehors du milieu académique
- Des idées se propagent en dehors du milieu (e.g. gestion de la mémoire)
- Récemment, intérêt renaissant pour ces langages, du à l'utilisation grandissante des langages des scripts (`python`, `ruby`).

Qu'est ce qu'un langage fonctionnel ?

Plusieurs avis, pas de définition précise, mais (en gros) :

- la programmation fonctionnelle est un

  style de programmation,

  où l'étape élémentaire du calcul est

  l'application d'une fonction à ses arguments.

- un langage est fonctionnel s'il supporte et encourage ce style fonctionnel.

Exemple

Sommation des entiers de 1 à 5 en Java :

```
total = 0;
for (i = 1; i <= 5; ++i)
    total = total+i;
```

Le calcul répose sur :

- *l'affectation des variables ;*
- *les boucles.*

Exemple

La fonction `somme` en Haskell :

```
somme :: [Int] -> Int
somme [] = 0
somme (x:xs) = x + somme xs
```

Sommation des entiers de 1 à 5 en Haskell :

```
somme (1:2:3:4:5:[]) =
1 + somme (2:3:4:5:[]) =
1 + 2 + somme (3:4:5:[]) =
1 + 2 + 3 + somme (4:5:[]) =
1 + 2 + 3 + 4 + somme (5:[]) =
1 + 2 + 3 + 4 + 5 + somme [] =
1 + 2 + 3 + 4 + 5 + 0 =
15
```

Exemple

La fonction `somme` en Haskell :

```
somme :: [Int] -> Int
somme [] = 0
somme (x:xs) = x + somme xs
```

Sommation des entiers de 1 à 5 en Haskell :

```
somme (1:2:3:4:5:[]) =
1 + somme (2:3:4:5:[]) =
1 + 2 + somme (3:4:5:[]) =
1 + 2 + 3 + somme (4:5:[]) =
1 + 2 + 3 + 4 + somme (5:[]) =
1 + 2 + 3 + 4 + 5 + somme [] =
1 + 2 + 3 + 4 + 5 + 0 =
15
```

La méthode de calcul répose sur :

- *l'application d'une fonction à ses arguments ;*
- *la récursion ;*
- *l'évaluation d'une expression vers un valeur.*

Variables et application

On peut lire l'expression Haskell

```
let
     x = 3 + 4
in
   x + 5
```

par :

appliquer le résultat de l'évaluation $3+5$
à la fonction f *définie par*

$$f(x) := x+5$$

Historique

1930s :

Alonzo Church développe le *lambda-calcul*, une théorie des fonctions, simple mais puissante.

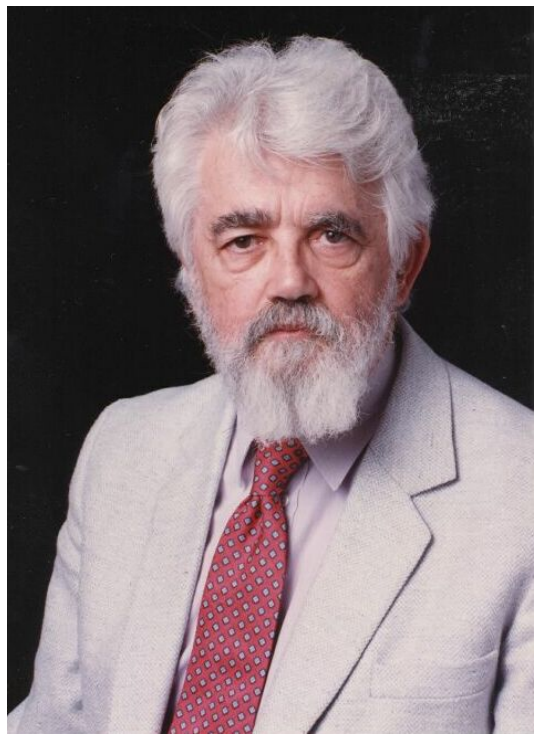
Historique

1930s :

Haskell B. Curry développe la *logique combinatoire*, qui deviendra le moteur des langages fonctionnels.

Historique

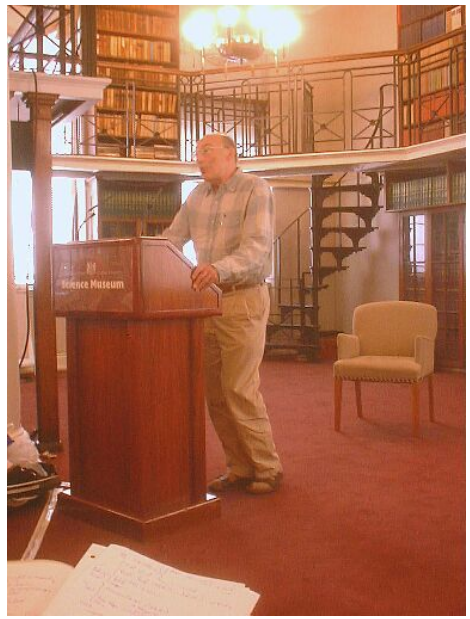
1950s :



John McCarthy développe Lisp, le premier langage fonctionnel, sous l'influence du lambda-calcul, mais en conservant l'affectation des variables.

Historique

1960s :



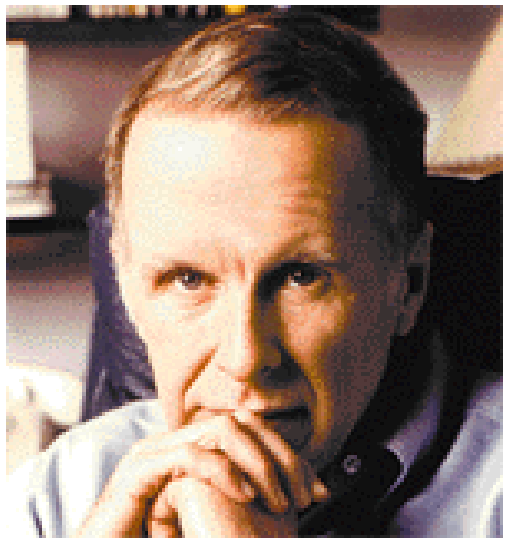
Peter Landin développe ISWIM [1] :

- le premier langage de programmation fonctionnel pur,
- fortement basé sur le lambda-calcul,
- sans affectation de variables.

1. De : « *If you See What I Mean* »

Historique

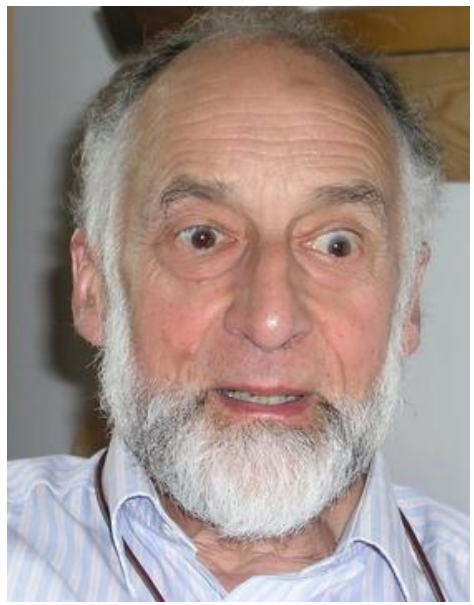
1970s :



John Backus développe FP, un langage de programmation fonctionnel qui pose l'accent sur les *fonctions d'ordre supérieur* et sur l'intégration avec le raisonnement sur les programmes.

Historique

1970s :



Robin Milner et autres développent ML, le premier langage fonctionnel moderne, qui introduit l'*inférence de type* et les *types polymorphes*.

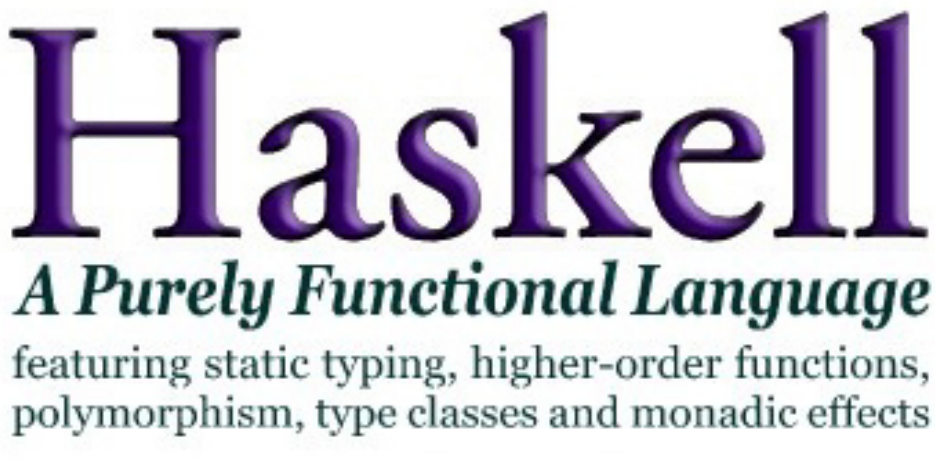
Historique

1970s - 1980s :

David Turner développe un nombre de langages fonctionnels *paresseux* (*lazy*), qui culminent dans le système Miranda (ancêtre de Haskell).

Historique

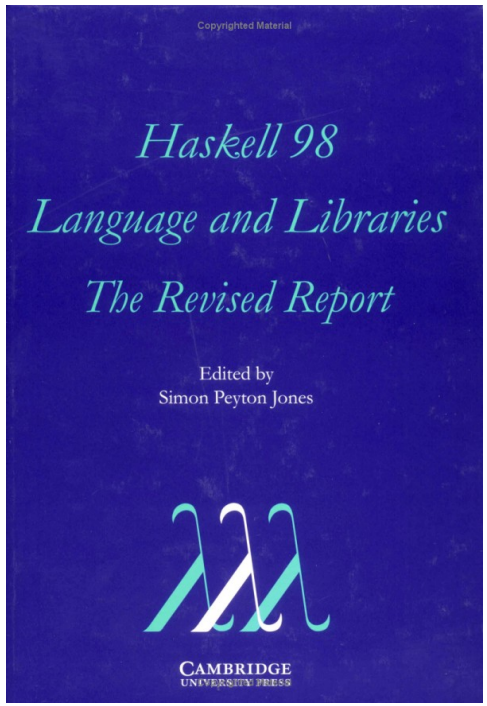
1987 :



Un comité international de chercheurs débute le développement de Haskell, un langage fonctionnel paresseux standard.

Historique

2003 :



Le comité publie le rapport Haskell 98, qui définit une version stable du langage.

Les héros nationaux

1985 :

Gérard Huet et son équipe à l'INRIA développent CAM, une version du langage ML destiné à s'intégrer avec le système Coq.
Autres intervenant dans le chemin CAM->CAML->OCAML : Xavier Leroy, Didier Rémy, Jérôme Vouillon.

Un avant-goût de Haskell

```
f []     = []
f (x:xs) = f ys ++ [x] ++ f zs
    where
      ys = [a | a <- xs, a <= x]
      zs = [b | b <- xs, b > x]
```

- *f de la liste vide est la liste vide,*
- *f d'une liste non vide est composéé de troix morceaux :*
  1. *f de ys,*
  2. *la tête de la liste,*
  3. *f de xs,*

  où

  1. *ys est la liste des a t.q. . . .*
  2. *zs est la liste des b t.q. . . .*

Un avant-goût de Haskell

```
f []     = []
f (x:xs) = f ys ++ [x] ++ f zs
    where
      ys = [a | a <- xs, a <= x]
      zs = [b | b <- xs, b > x]
```

- *f de la liste vide est la liste vide,*
- *f d'une liste non vide est composéé de troix morceaux :*
  1. *f de ys,*
  2. *la tête de la liste,*
  3. *f de xs,*

  où

  1. *ys est la liste des a t.q. . . .*
  2. *zs est la liste des b t.q. . . .*