Programmation Fonctionnelle
Introduction

Luigi Santocanale
LIF, Aix-Marseille Université
Marseille, FRANCE

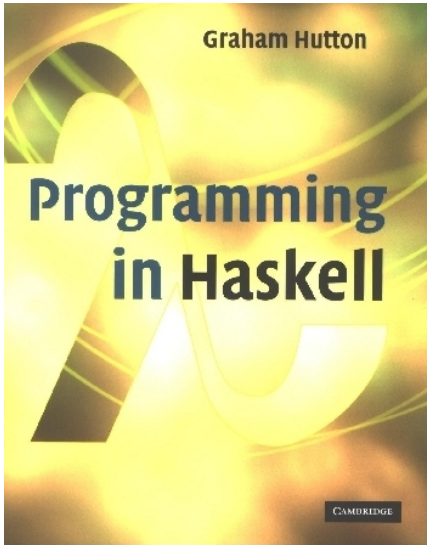
29 août 2017

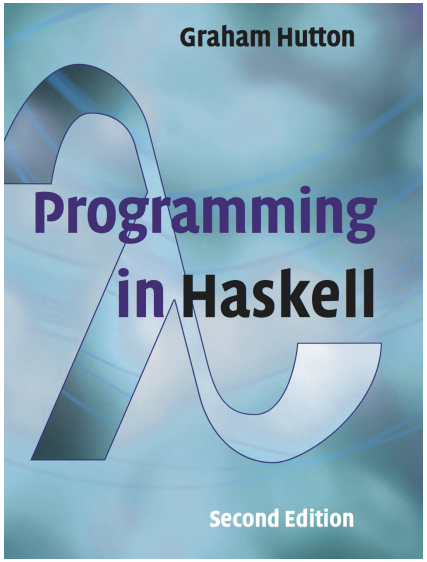
Plan

Le cours

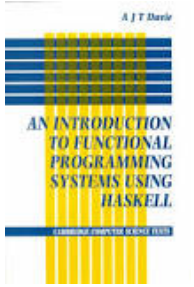
Cours partiellement basé sur le livre :

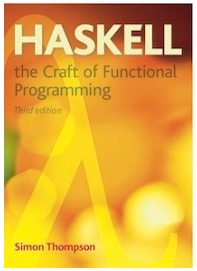
- Graham Hutton. *Programming in Haskell*. Cambridge University Press, 2007.

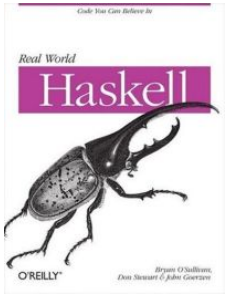




Autres lectures :







Voir la page web du cours.

Bureaucratie

Page web du cours :

`pageperso.lif.univ-mrs.fr/~luigi.santocanale/teaching/PF/`

Calcul de la note finale (I session) :

$$\mathtt{NF} = 0,7 * \mathtt{Examen} + 0,3 * \mathtt{Projet}$$

Attention : on ne rattrappe pas le projet.

Calcul de la note finale (II session) :

$$\mathtt{NF} = \max(\mathtt{Examen}; 0,7 * \mathtt{Examen} + 0,3 * \mathtt{Projet})$$

Bureaucratie

Page web du cours :

`pageperso.lif.univ-mrs.fr/~luigi.santocanale/teaching/PF/`

Calcul de la note finale (I session) :

$$\mathtt{NF} = 0,7 * \mathtt{Examen} + 0,3 * \mathtt{Projet}$$

Attention : on ne rattrappe pas le projet.

Calcul de la note finale (II session) :

$$\mathtt{NF} = \max(\mathtt{Examen}; 0,7 * \mathtt{Examen} + 0,3 * \mathtt{Projet})$$

Bureaucratie

Page web du cours :

`pageperso.lif.univ-mrs.fr/~luigi.santocanale/teaching/PF/`

Calcul de la note finale (I session) :

$$\mathtt{NF} = 0,7 * \mathtt{Examen} + 0,3 * \mathtt{Projet}$$

Attention : on ne rattrappe pas le projet.

Calcul de la note finale (II session) :

$$\mathtt{NF} = \mathtt{max}(\mathtt{Examen}; 0,7 * \mathtt{Examen} + 0,3 * \mathtt{Projet})$$

Le projet

- assigné à le deuxième moitié d'octobre ;
- à rendre en début de décembre ;
- soutenance quelques jours après rendu.

Les débuts (1980 environ) : 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 ?

Théoriser les langages de programmation

Deviser des langages de programmation qui :

- permettent que les programmes soient écrits clairement, à un haut niveau d'abstraction ;
- encouragent le recours à la vérification formelle ;
- supportent des composantes logicielles réutilisables ;
- 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.

Théoriser les langages de programmation

Deviser des langages de programmation qui :

- permettent que les programmes soient écrits clairement, à un haut niveau d'abstraction ;
- encouragent le recours à la vérification formelle ;
- supportent des composantes logicielles réutilisables ;
- 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.

Évolution . . .

- 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, fonctions d'ordre supérieure, . . .)
- Récemment : intérêt renaissant et grandissant pour ces langages . . .
- apparenté au succès des langages des scripts (`python`, `ruby`, `perl`) . . .
- . . . et à la puissance actuelle de l'hardware.

Utilisation des langages fonctionnels

- En janvier 2015, Haskell était le 15ème langage de programmation le plus utilisé (source `http://redmonk.com/sogrady/2015/01/14/language-rankings-1-15/`).
- Aujourd'hui (june 2017) Haskell est 19ème (source `http://redmonk.com/sogrady/2017/06/08/language-rankings-6-17/`).
- Un autre langage fonctionnel, Scala, est classé 12ème.
- Par rapport à Scala, Haskell est purement fonctionnel.

Qu'est ce qu'un langage fonctionnel ?

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

- la programmation fonctionnelle est un

  paradigme/style de programmation,

  où
  - paradigme de programmation de type **déclaratif** ;
  - considère le calcul en tant qu'**évaluation de fonctions mathématiques** ;
  - 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 ;*
- *évaluation d'une expression, ou non (mathématique)*

  vers un valeur.

Affectation des variables versus application

Considérez l'expression Haskell

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

On peut "lire" cette expression par :

> *appliquer le résultat de l'évaluation* $3+4$
> *à la fonction* f *définie par*
>
> $$f(x) := x+5$$

Dans les langages de programmation fonctionnels, la notion de variable, au sens usuel, "disparaît".

Les variables sont immutables

```
let
    x = 0
in
let
    f y = x + y
in
let
    x = 1
in
    f 0
```

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 appliqué à la liste vide est la liste vide,*
- *f appliqué à une liste non vide est composéé de troix morceaux (dans l'ordre) :*
  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 appliqué à la liste vide est la liste vide,*
- *f appliqué à une liste non vide est composéé de troix morceaux (dans l'ordre) :*
  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. . . .*

Historique

1930s :

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

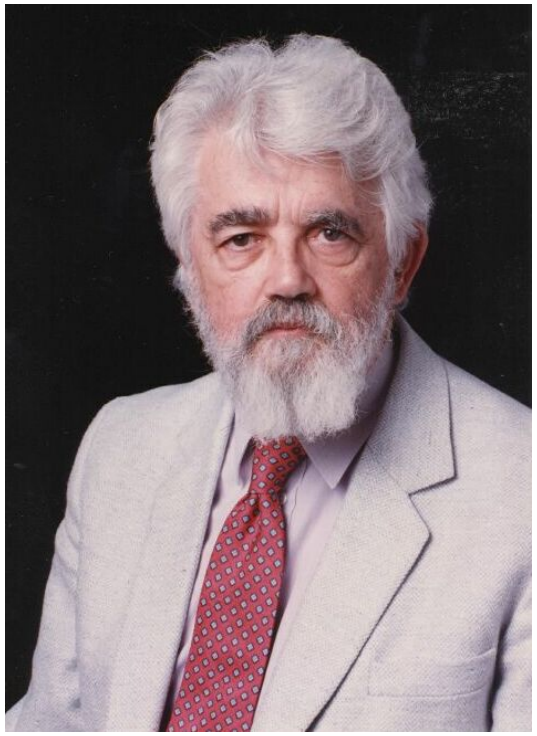
Historique

1930s :

Haskell B. Curry développe la *logique combinatoire* (variante du λ-calcul), qui deviendra le moteur des langages fonctionnels (paresseux).

Historique

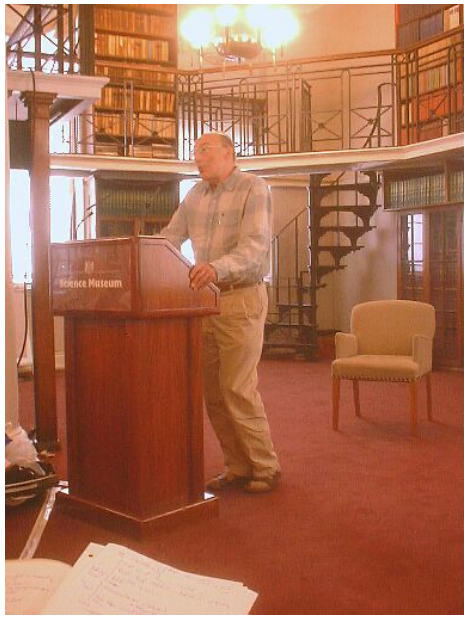
1950s :



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

Historique

1960s :



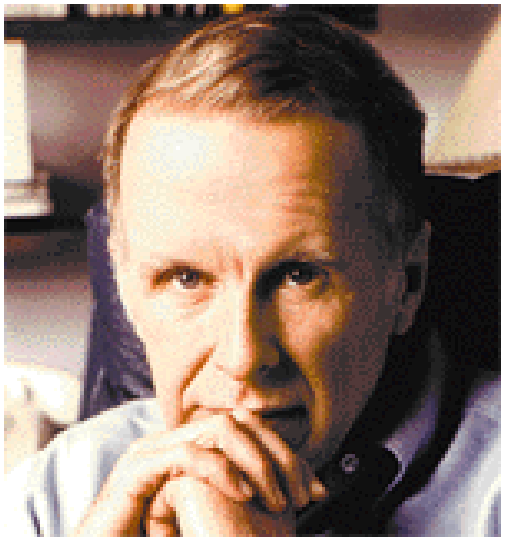
Peter Landin développe ISWIM [1] :

- 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

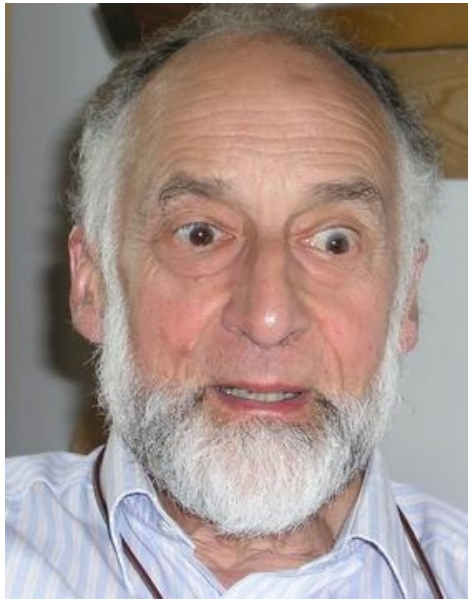
1970s :



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*.

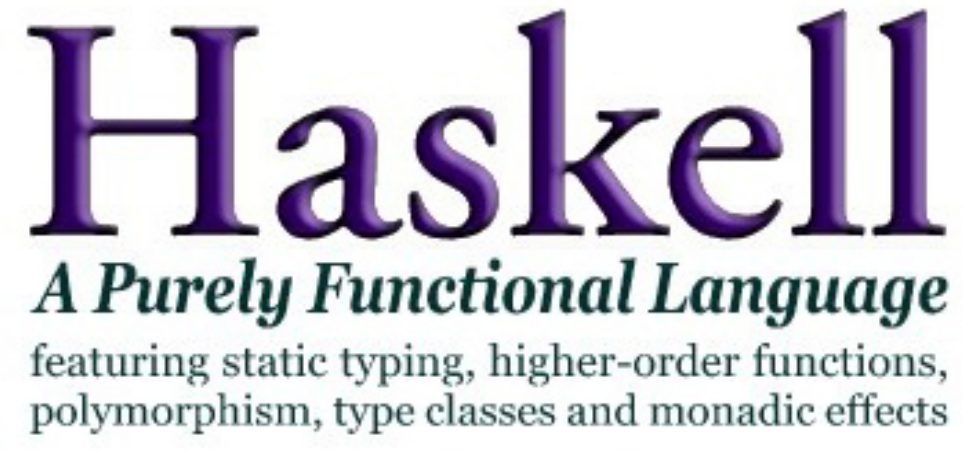
Historique

1970s - 1980s :

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

Historique

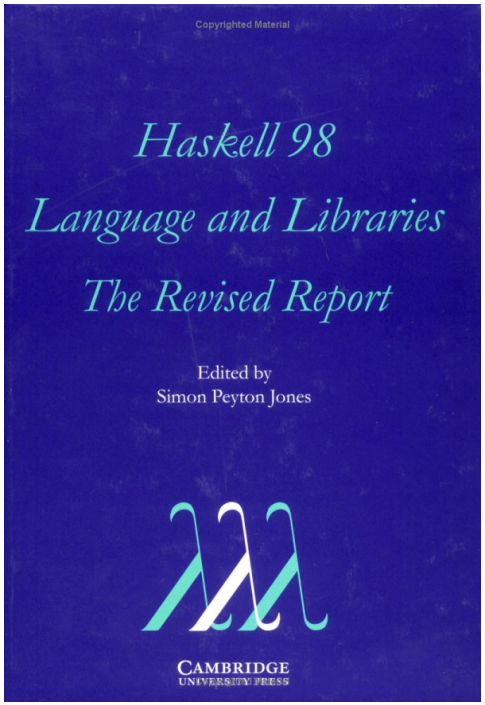
1987 :



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.

Pour terminer ce chapitre

	Typage forte	Typage statique	Inference de type
C	Non	Oui	Non
Java	Oui	Oui	Non
Python	Oui	Non	Non
Haskell	Oui	Oui	Oui