Programmation fonctionnelle / Haskell

Clément Delafargue

15 octobre 2012

Section 1

Contexte

Une profusion de langages

Il existe beaucoup de langages de programmation

Des célèbres (C, Java)

D'autres moins (J)

Pourquoi cette profusion ?

Différents paradigmes

Quelques paradigmes de programmation :

- Impératif
- Fonctionnel
- Logique

Le paradigme impératif

Remonte à la machine de Turing

Manutention d'espace mémoire

Proche du fonctionnment des processeurs

Le paradigme fonctionnel

Remonte au Lambda Calcul

Définition de fonctions et de leur combinaisons

Proche des mathématiques

Le paradigme fonctionnel

Pas d'instructions, mais des données et leurs relations

Pourquoi le fonctionnel ?

- Raisonnement simplifié
- Débogage aisé
- Tests unitaires plus simples
- Concurrence
- Garanties fortes

Section 2

La programmation fonctionnelle

L'essence de la programmation fonctionnelle

Des fonctions

Leur composition

L'immutabilité

Définition mathématique d'une variable

Immutabilité

La “transparence référentielle”

Une expression dénote toujours la meme chose, indépendamment du contexte.

Terme pas toujours employé correctement.

En réalité : “Absence d’effets de bord”

La récursion

Fondamental

Comment représenter une boucle sans la mutabilité ?

Approche mathématique

```
x^0 = 1
x^n = x * x^(n-1)
```

Insiste sur la décomposition du problème

La programmation fonctionnelle typée

Chaque valeur a un type (Entier, Booléen, Liste d'entiers, Fonction des entiers vers les Booléens)

Systèmes de types souvent sophistiqués

Correspondance de Curry Howard

Correspondance preuve - programme.

```
type <=> proposition
programme bien typé <=> preuve de la proposition
```

Fondamental

Un langage fonctionnel

Permet la manipulation de fonctions

Encourage l'immutabilité

Saines lectures

Pour mieux comprendre pourquoi la FP est intéressante

- FP for the rest of us
- Why FP matters

La référence :

Structure and Interpretation of Computer Programs

Section 3

Un langage fonctionnel, Haskell

Un langage fonctionnel, Haskell

Haskell (nommé en hommage à Haskell Curry)

Première version en 1990 (un an avant python) “Design by committee” (entre autres : Simon Peyton Jones, John Hughes, Philip Wadler)

Les caractéristiques de Haskell

Haskell est un langage :

- Fonctionnel
- Statiquement typé
- “Fortement” typé
- À inférence de type
- Pur
- Paresseux
- Utilisé dans la vraie vie (un peu)

En détail :

- Typage statique : typage déterminé en amont
- Typage “fort” : à voir
- À inférence de type : capable de déterminer le type d’une variable
- Pur : pas d’effets de bord
- Paresseux : ne calcule un terme que quand (**et si**) il en a besoin

Les avantages

Haskell est un langage :

- Fonctionnel : composablité
- Statiquement typé : sûreté
- “Fortement” typé : sûreté
- À inférence de type : concision
- Pur : composabilité
- Paresseux : composabilité

Pourquoi apprendre Haskell ?

- Pour le fun
- Pour voir ce qu'est un beau langage
- Pour les garanties qu'il apporte
- Pour la rapidité de développement

Qui utilise Haskell ?

- Moi
  - Cette présentation a été générée par un programme en Haskell
  - Elle est affichée dans un système de fenêtrage en Haskell
- Des banques, en interne
- Facebook
- Des agences de développement

Quelques projets en Haskell

- pandoc: conversion de documents
- xmonad: gestionnaire de fenêtres
- Darcs: gestionnaire de versions
- Yesod: framework web

Quelques exemples

La somme des n premiers nombres premiers

```
primes = nubBy (\x y -> (gcd x y) > 1) [2..]
sumOfPrimes n = sum $ take n primes
```

La suite de Fibonacci

```
fibs = fix ((0:) . scanl (+) 1)
fibs' = 0 : 1 : zipWith (+) fibs' (tail fibs')
```

La syntaxe Haskell

Annotation de type (facultatif)

```
a :: String
f :: String -> String -> Boolean
(+) :: (Num a) => a -> a -> a
```

La syntaxe Haskell

Déclaration de fonction

```
add2 a = a + 2

isTwo 2 = True
isTwo _ = False

headOr a [] = a
headOr _ (x:_) = x
```

La syntaxe Haskell

Application de fonction

```
add2 4
isTwo (add2 0)
3 + 3
```

Application partielle

```
(+3)
(==0)
```

La syntaxe Haskell

Composition de fonction

```
isTwo (add2 0)
isTwo . add2 0
```

La syntaxe Haskell

Déclaration de variable

```
let x = 0, y = 2 in x + y
```

Déclaration de fonction interne

```
foo x = bar (x + 2) where
    bar y = y + 5
```

La syntaxe Haskell

List comprehension

```
[ 2 * x | x <- [1..5]]
```

Exercice

Calculer la somme d'une liste d'entiers.

La version courte

```
sum = fold (+) 0
```

Comment aboutir à la solution

Pour un algorithme récursif

- Exprimer un cas d'arret
- Exprimer un pas de réduction

Quelques petits exercices

- Calculer la somme des entiers naturels multiples de 3 ou 5 inférieurs à 1000
- Calculer l'inverse d'une liste
- Le Fizz Buzz

Saines lectures

- Learn You a Haskell
- Real World Haskell
- `#haskell` sur freenode