Approche fonctionnelles de la programmation

Didier Verna

ING1 2014

Sources disponibles sur http://ing1.nemunai.re/ ou ing1@nemunai.re

Table des matières

Chapitre 1

Introduction

Lisp : fanboy du lisp ! C'est sans doute l'un des langage qui inclus le plus de paradigme possible : homoiconicité (un autre exemple serait Prolog).

Tous les dialectes de lisp sont composés de 7 opérateurs.

Dans les dialectes sympa, il y a Scheme qui est simple et minimaliste. Common Lisp est un standard et possède même des bibliothèques standards. Emacs Lisp est un mélange du Common Lisp et du Mac Lisp.

Ruby est inspiré de Lisp.

1.1 Paradigme

Un *paradigme* est une conception qui permet d'exprimer un certain nombre de choses : fonctionnel, impérative, procédural, ...

Cela permet de faciliter l'écriture de certain concepts : dans certain paradigme, il est plus simple de faire certaines choses. Tout est possible partout, plus ou moins simplement :p

Le paradigme affecte donc l'expressivité d'un langage, mais aussi la manière de penser dans un langage.

Généralement, un langage n'a pas de paradigme ferme.

1.2 1er ordre

Inventé par Christopher Strackey.

Un objet de première classe peut être stocké dans une variable, agrégé (structure), passer en argument à une fonction, retourner d'une fonction, manipuler de manière anonyme (utiliser un entier sans l'avoir assigné à une variable) ou faire de la construction dynamique (un entier calculé à partir de deux autres entiers).

Dans une langage fonctionnel, la fonction est un objet de première classe.

1.2.1 Lisp 1 vs 2

Common Lisp est un Lisp 2, Scheme un Lisp 1. Déréférence les espaces de noms.

1.2.2 Mapping et folding

Mapping Consiste à appliquer une fonction sur tous les éléments d'une fonction.

Folding Consiste à combiner 2 par 2 tous les éléments d'une liste pour n'avoir plus qu'une valeur unique à la fin.

Retour fonctionnel Capacité à créer de nouvelles fonctions à la volée.

1.3 Pureté

Une fonction mathématique, sans effet de bord. En lui passant les mêmes arguments, elle doit toujours renvoyé la même chose quelque soit le contexte.

Au niveau des variables, elles n'existent plus ! il n'y a que des constantes.

Étant donné qu'écrire dans un terminal est un effet de bord, l'implémentation des fonctions interagissant avec des fichiers sont confinées dans des bunkers.

1.3.1 Intérêt

La pureté est essentielle pour le parallélisme. En effet, avec la pureté, on a pas de problème de concurence d'accès, ...

Par exemple Erlang est parfaitement adapté à ça. Il est d'ailleurs parfaitement adapté à la distribution qui est prévu dans les primitives du langage.

Sëmantique locale aux fonctions Étant donné que les fonctions pures n'ont pas d'effet de bord, il est très facile de les tester/déboguer.

Preuve de programme

1.4 Évaluation

1.4.1 Strict

Dans un évaluateur strict (tel que Lisp), il va d'abord évaluer les arguments de gauche à droite, puis il applique la fonction avec les arguments évalués.

1.4.2 Lazy

ou évaluation paresseuse (Haskell).
Les expressions ne sont évaluées que quand on en a besoin.

1.5 Typage

En Haskell, toutes les variables sont typées et la vérification est faite à l'exécution.

Chapitre 2

2.1 Élements de syntaxe

2.1.1 Syntaxe

Haskell respecte la règle de l'offside-rule (comme Python, initié par Peter J. Landin). Il existe un séparateur implicite ; Pour le nommage, c'est comme le C, avec en plus l'apostrophe.

En Lisp, il n'y a pas de syntaxe ou presque. Aucun mot n'est réservé.

2.1.2 Opérateurs et fonctions

En Lisp, il n'y a pas de distinction, la notation est exclusivement préfixe.

En Haskell, il y a une notation infixe pour les principaux opérateurs. On peut d'ailleurs définir ses propres opérateurs infixes, c'est génial.

2.2 Notion d'expressions

Chapitre 3

Bibliographie

Apprendre Scheme
Le livre Practical Common Lisp (http://www.gigamonkeys.com/book/)