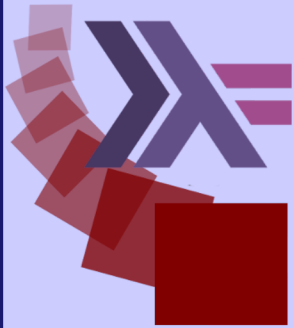
PROGRAMMATION FONCTIONNELLE

via le langage Haskell

OUALI Abdelkader

L2 au département Mathématique-Informatique
UFR des sciences
Université de Caen Normandie



Organisation

Organisation :

- 10 CM de 2H
- 10 TP de 3H
- L2 MATH (2 groupes)
- L2 INFO (8 groupes : 4 groupes A ++ 4 groupes B)
- Début des TPs (2ème semaine du semestre, i.e. à partir du lundi 09 janvier'23)

Évaluation en CC sous réserve de validation :

- session 1 : 50% CC et 50% Examen et
- en cas de session 2 : 50% CC + 50 % Examen en session 2

Infos support CM + TPs sous eCampus

Références bibliographiques

- Programming in Haskell - Graham Hutton
- Notes on Functional Programming with Haskell, H. Conrad Cunningham.
- The craft of Functional Programming, Simon Thompson. 3ème édition.

Le contenu du cours portera sur une synthèse des différents chapitres de ces références.

Paradigmes de programmation

UNIVERSITÉ CAEN NORMANDIE

En programmation, vous avez vu jusqu'à présent les bases de la **programmation impérative (procédurale)**

- Variables et affectation (état)
- Structures de contrôle
- Structures de données
- Fonctions/Procédures et arguments
- Pointeur et gestion de mémoire

Un programme est une **suite d'instructions** qui définit un schéma de résolution d'un problème donné.

Paradigmes de programmation

UNIVERSITÉ CAEN NORMANDIE

Autres paradigmes : différents **styles** d'écriture de programme. un informaticien jongle avec ces styles selon les **besoins**.

- Machine d'état
- Instruction à la machine
- Langages : C, Fortran, Pascal et beaucoup de langages modernes.

Effet de bord

- Modification d'une zone de **mémoire partagée**
- Source de **nombreux bugs** :
  - Utilisation **imprévu** d'une variable,
  - **Mauvaise allocation** de la mémoire,
  - **État imprévu** dans le programme
  - . . .
- **Réduction** des interactions
  - Programmation **structurée**,
  - Variable **locale**,
  - **Encapsulation**

Paradigmes de programmation

UNIVERSITÉ CAEN NORMANDIE

En programmation, vous avez vu aussi les bases de la **programmation orientée-objet**

- Le concept de classe et objet
- Encapsulation
- Héritage
- Abstraction
- Polymorphisme

Un programme est une **suite d'instructions** qui définit un schéma de résolution d'un problème donné.

Paradigmes de programmation

UNIVERSITÉ CAEN NORMANDIE

Autres paradigmes : différents **styles** d'écriture de programme. un informaticien jongle avec ces styles selon les **besoins**.

Programmation **orientée-objet**

- Modularité, Abstraction, Productivité et Sûreté
- Envoi de messages
- Langages : JAVA, C++, Python, etc.

Paradigmes de programmation

UNIVERSITÉ CAEN NORMANDIE

Autres paradigmes : différents **styles** d'écriture de programme. un informaticien jongle avec ces styles selon les **besoins**.

- C'est tout ? bien évidement non !
- Nous avons vu aussi SQL, il est situé où ?
  - ➠ Programmation déclarative : définir seulement le problème le solveur s'occupe du reste
- D'autres modèles de calcul ?

Programmation fonctionnelle

- Application des fonctions aux arguments
- Principe de récurrence et composition de fonction
- Langages spécialisés : Haskell, Elixir, etc.

Exemple de code

Effectuer une somme des entiers entre 1 et 10

Code en C :

```c
int sum = 0;
for (int i=1; i<=10; i++){
        sum += i;
}
```

donner des instructions
le résultat est l' affectation des variables (i et sum)
(changement d'état)

Code en Haskell :

```haskell
sum [1..10]
```

sum et .. sont des fonctions !
Le résultat est l'application de la définition des fonctions

Lambda-calcul

UNIVERSITÉ CAEN NORMANDIE

- Alonzo Church (1930), quelle est la notion d'une fonction d'un point de vue calcul ?
  - λ-calcul
  - Théorie simple mais puissante de fonctions
- une fonction est une **boite noire** sans état interne :

$$\lambda x.x + 1 \text{ ou } \lambda x.\lambda y.x + y$$

  - Variables
  - Construction de la fonction (notation λ)
  - Application par substitution des variables par les valeurs en entrée

Lambda-calcul

UNIVERSITÉ CAEN NORMANDIE

- Peut encoder tout calcul (code en λ-calcul $\leftrightarrows$ code en machine de Turing (1936))
  - les termes lambda peuvent émuler les machines de Turing et inversement,
  - Une architecture Von-Neumann peut exécuter les deux (à l'exception des restrictions techniques)
  - Lambda-calcul est moins efficace sur une architecture Von-Neumann en raison son nature sans état et le niveau d'abstraction
- La base de la programmation fonctionnelle
- Présent dans la plupart de langages (JAVA, VisualBasic, C#, etc.)

Historique des langages fonctionnels

UNIVERSITÉ CAEN NORMANDIE

- 1950, John McCarthy a développé Lisp ("LISt Processor"),
- 1960, Peter Landin a développé ISWIM ("If you See WhatMean"),
- 1970, John Backus développe la FP («Functional Programming »), Robin Milner and others developed ML (“Meta-Language”)
- 1980, Robin Milner et al. Proposition des langages fonctionnels paresseux (Système Miranda)
- 1987, initiation du développement de of **Haskell** (Haskell Curry)
- 1990, Phil Wadler et al. Classes types et Monades (effet de bords)
- 2003, 2010 Haskell Report, version stable de Haskell (conception longue mais fructueuse)
- Aujourd’hui, plus d’accès avec des bibliothèques, supports, nouvelles caractéristiques, influence dans d’autres langages...

Langages fonctionnels

UNIVERSITÉ CAEN NORMANDIE

Intérêts : expressivité, découplage du code, réduction des sources d'erreurs,...

- Langages fonctionnels de type **Lisp** :
  - expressions symboliques
  - homoïconicité
  - typage dynamique
- Langages fonctionnels de type **ML** :
  - types algébriques
  - typage statique et inférence de type
  - filtrage par motifs

Programmation fonctionnelle : objectif

UNIVERSITÉ CAEN NORMANDIE

- **Réfuter** la notion d'instruction et la mutation d'une variable
- **Fonction** comme une base du modèle de calcul
- Programmer à base de fonctions **pures** (la sortie d'une fonction est **uniquement** déterminée par l'entrée)
- **Composition** de fonctions (prend entrée une fonction et retourne une fonction)
- Programmation **orientée-expression** (déclarative)

Langage Haskell

- Langage créé en 1990 par un comité dédié avec les caractéristiques suivantes :
  - Récursivité, filtrage par motifs, curry, fonctions d'ordre supérieur, map, filtre reduce (fold)
  - Évaluation paresseuse/stricte, sémantique non-stricte/stricte.
  - Inférence de type, suite, fermeture, monades,...
- Normes : Haskell 98, Haskell 2010
- Langage **fonctionnel**, **pur** et **paresseux** ayant un **typage statique**
- Compilateur : GHC
- Outils de gestion de projet : Cabal, Stack
- Utilisations : recherche académique, industrie (surtout en finance), enseignement

Langage fonctionnel, pur et paresseux ayant un typage statique

- Haskell **prend en charge et encourage** le style de programmation fonctionnelle où la méthode de calcul de base est l'**application de fonctions à des arguments**

```
int sum = 0;
for (int i=1; i<=10; i++){
        sum += i;
}
```

⟶

```
sum [1..10]
```

- Toutes les valeurs, y compris les fonctions, jouent le même rôle
  - utilisées comme des arguments
  - retournées comme résultats
  - placées dans une structure de données

```
>f x = x <= 5
>filter f [1..10]
[1,2,3,4,5]
>g = f
>filter g [1..10]
[1,2,3,4,5]
```

Langage fonctionnel, pur et paresseux ayant un typage statique

UNIVERSITÉ CAEN NORMANDIE

- Comme en mathématiques, aucune possibilité de modifier l'environnement extérieur à la fonction sans aucun **effets de bord**

  - **Données immuables :** au lieu de modifier les valeurs existantes, des copies modifiées sont créées et l'original est préservé,

    ➠ il n'y a donc pas d'affectation destructive

  - **Transparence référentielle :** les expressions donnent la même valeur chaque fois qu'elles sont invoqués ; aide au raisonnement.

    ➠ Une expression peut être remplacée par sa valeur et le comportement résultant est le même qu'avant le changement

    `y = f X et g = h y y`
    `puis, en remplaçant la définition de g par g = h (f x) (f x)`
    `obtiendra le même résultat (valeur).`

Langage fonctionnel, pur et paresseux ayant un typage statique

UNIVERSITÉ CAEN NORMANDIE

- Les expression ne sont pas évaluée tant que leur valeur n'est pas requise
  - Éviter les calculs inutiles
  - Terminer le programme dès que possible

```
Prelude> oublier x = 0

Prelude> tjrs_continuer x = tjrs_continuer (x+1)

Prelude> oublier(tjrs_continuer 1)
?
```

Langage fonctionnel, pur et paresseux ayant un typage statique

UNIVERSITÉ CAEN NORMANDIE

- Types sont **déterminés** à la compilation (**inférence de types**) avec la possibilité d'en **indiquer quelques-uns** pour définir les fonctions

- Définir ses propres types (voir dans les prochains CMs)

```haskell
oublier :: Int -> Int
-- types polymorphiques et classes types
tjrs_continuer :: Num t1 => t1 -> t2
```

Pourquoi apprendre Haskell ?

UNIVERSITÉ CAEN NORMANDIE

- Haskell est plus adapté par rapport à d'autres langages fonctionnels (Elixir, etc.) en :
  - Prototypage rapide
  - Passage à l'échelle (code native beaucoup plus rapide qu'un script)
  - Concurrence
- Un langage que vous pourriez utiliser pour mieux programmer dans votre langage préféré
  - De nombreux langages intègrent maintenant des concepts empruntés à la programmation fonctionnelle

https://wiki.haskell.org/Haskell_in_practice

Exemple de code en Haskell

```haskell
f [] = []
f (x:xs) = f ys ++ [x] ++ f zs
                        where
                                ys = [a | a <- xs, a <= x]
                                zs = [b | b <- xs, b > x]
main =
        print (f [3,2,4,1,5])
```

Exemple de code en C

```c
#include<stdio.h>
int partition(int xs[], int first, int last){
        int k = xs[first];
        int i = first-1;
        int j = last+1;
        int temp;
        do {
                do { j--; } while (k<xs[j]);
                do { i++; } while (k>xs[i]);
                if (i<j) { temp=xs[i]; xs[i]=xs[j]; xs[j]=temp; }
        } while (i<j);
        return j;
}
void f(int xs[], int first, int last){
        int mid;
        if (first <last) {
                mid = partition(xs, first, last);
                f(xs, first, mid);
                f(xs, mid+1, last);
        }
        return;
}
int main(){
        int xs[5] = {3,2,4,1,5};
        f(xs,0,5);
        for(int i=0;i<5;i++)
                printf("%d ", xs[i]);
        printf("\n");
        return 0;
}
```

GHC et GHCi

- À partir d'un terminal, le compilateur est appelé en tant que

```
> ghc monfichier.hs
> runghc monfichier.hs (sans créer le fichier binaire)
> ghci (ou comme >ghc --interactive)
```

- GHCi opère sur une boucle évaluer-afficher

```
Prelude>sqrt(3^2 + 4^2)
5.0
Prelude>
```

- Possibilité d'invoquer un éditeur dans le terminal

GHCi

Commandes de bases utiles dans GHCi :

- > :? – Help ! Afficher toutes les commandes
- > :load test – Ouvrir le fichier test.hs
- > :reload – Recharge le fichier précédemment chargé
- > :main a1 a2 – Invoque main avec les arguments de ligne de commande a1 a2
- > :! – Exécuter une commande shell
- > :edit nom – Modifier le script nom.hs
- > :edit – modifier le script actuel
- > :type expr – Afficher le type d'une expression
- > :quit –Quitter GHCi

Les commandes peuvent être **abrégées**. Par exemple, :r pour ;reload
Au démarrage, les définitions du "**Standard Prelude**" sont chargées.

GHCi

Commandes de bases utiles dans GHCi :

- > :? – Help ! Afficher toutes les commandes
- > :load test – Ouvrir le fichier test.hs
- > :reload – Recharge le fichier précédemment chargé
- > :main a1 a2 – Invoque main avec les arguments de ligne de commande a1 a2
- > :! – Exécuter une commande shell
- > :edit nom – Modifier le script nom.hs
- > :edit – modifier le script actuel
- > :type expr – Afficher le type d'une expression
- > :quit –Quitter GHCi

Les commandes peuvent être **abrégées**. Par exemple, :r pour ;reload
Au démarrage, les définitions du "**Standard Prelude**" sont chargées.