Algorithmique - Programmation 1

Cours 9

Université Henri Poincaré

CESS Epinal

Automne 2008

Plan

Au sujet du cours

- Dans ce cours, nous avons étudié un **paradigme** de programmation : la **programmation fonctionnelle**
  - programmation où les fonctions sont des valeurs comme les autres
    $\longrightarrow$ pas de **boucles**, mais des fonctions récursives
    $\longrightarrow$ pas de données **mutables**, mais des liaisons entre noms et valeurs
  - différent de la programmation **impérative** qui utilise une suite d'instructions

Au sujet du cours (suite)

- La programmation fonctionnelle utilise généralement des **algorithmes** manipulant des **données** plus ou moins **complexes**
- Exemples :
  - produits cartésiens
  - listes (voire listes de listes, *etc*)
  - types sommes
- Style de programmation généralement moins verbeux que la **programmation impérative**, car manipulant peu de noms (les données sont encapsulées dans des structures complexes)

Au sujet du cours (suite)

- Cependant, il y a quelques **invariants** entre les différents paradigmes de programmation :
  - **modélisation** de données
  - **algorithmes** de résolution de problèmes
  - **vérification** au moyen de tests
- Notion d'interpréteur commun avec d'autres langages :

  shell scripts (bash, csh ...), **Prolog**, **Haskell**, **Perl**, **Oz**, *etc*

Au sujet du cours (suite)

- Implantation des algorithmes au moyen du langage `Caml`
- Langage de programmation typé :
  - **typage statique**
    $\longrightarrow$ le compilateur vérifie le type des données manipulées avant d'exécuter le code
    $\longrightarrow$ beaucoup d'erreurs sont détectées statiquement ($\neq$ dynamiquement)
- Compilateur efficace
  (*cf* The Great Computer Language Shootout – `http://shootout.alioth.debian.org/`)

Plan

Rappels

- Notions abordées dans ce cours :

Rappels

- Notions abordées dans ce cours :
  - Typage d'expressions

Rappels

- Notions abordées dans ce cours :
  - Typage d'expressions
  - Polymorphisme

Rappels

- Notions abordées dans ce cours :
  - Typage d'expressions
  - Polymorphisme
  - $\left\{ \begin{array}{l} \text{Fonctions à } n \text{ paramètres} \\ \text{Application partielle} \end{array} \right.$

Rappels

- Notions abordées dans ce cours :
  - Typage d'expressions
  - Polymorphisme
  - $\left\{ \begin{array}{l} \text{Fonctions à } n \text{ paramètres} \\ \text{Application partielle} \end{array} \right.$
  - Fonctions récursives

Rappels

- Notions abordées dans ce cours :
  - Typage d'expressions
  - Polymorphisme
  - $\left\{ \begin{array}{l} \text{Fonctions à } n \text{ paramètres} \\ \text{Application partielle} \end{array} \right.$
  - Fonctions récursives
  - Produits cartésiens

Rappels

- Notions abordées dans ce cours :
  - Typage d'expressions
  - Polymorphisme
  - $\left\{ \begin{array}{l} \text{Fonctions à } n \text{ paramètres} \\ \text{Application partielle} \end{array} \right.$
  - Fonctions récursives
  - Produits cartésiens
  - Listes

Rappels

- Notions abordées dans ce cours :
  - Typage d'expressions
  - Polymorphisme
  - $\left\{ \begin{array}{l} \text{Fonctions à } n \text{ paramètres} \\ \text{Application partielle} \end{array} \right.$
  - Fonctions récursives
  - Produits cartésiens
  - Listes
  - Enregistrements

Rappels

- Notions abordées dans ce cours :
  - Typage d'expressions
  - Polymorphisme
  - $\left\{ \begin{array}{l} \text{Fonctions à } n \text{ paramètres} \\ \text{Application partielle} \end{array} \right.$
  - Fonctions récursives
  - Produits cartésiens
  - Listes
  - Enregistrements
  - Types sommes

Rappels : Typage d'expressions

- Donnez le type des expressions suivantes :

```
# function 0 -> 42 | x -> x + 1;;
```

Rappels : Typage d'expressions

- Donnez le type des expressions suivantes :

```
# function 0 -> 42 | x -> x + 1;;
- :  int -> int = <fun>
```

Rappels : Typage d'expressions

- Donnez le type des expressions suivantes :

```
# function 0 -> 42 | x -> x + 1;;
- :  int -> int = <fun>

# function f -> 3 + (f 4);;
```

Rappels : Typage d'expressions

- Donnez le type des expressions suivantes :

```
# function 0 -> 42 | x -> x + 1;;
- :  int -> int = <fun>

# function f -> 3 + (f 4);;
- :  (int -> int) -> int = <fun>
```

Rappels : Typage d'expressions

- Donnez le type des expressions suivantes :

```
# function 0 -> 42 | x -> x + 1;;
- :  int -> int = <fun>

# function f -> 3 + (f 4);;
- :  (int -> int) -> int = <fun>

# function x -> function y -> (x > 4) & y;;
```

Rappels : Typage d'expressions

- Donnez le type des expressions suivantes :

```
# function 0 -> 42 | x -> x + 1;;
- :  int -> int = <fun>

# function f -> 3 + (f 4);;
- :  (int -> int) -> int = <fun>

# function x -> function y -> (x > 4) & y;;
- :  int -> bool -> bool = <fun>
```

Rappels : Typage d'expressions

- Donnez le type des expressions suivantes :

```
# function 0 -> 42 | x -> x + 1;;
- :  int -> int = <fun>

# function f -> 3 + (f 4);;
- :  (int -> int) -> int = <fun>

# function x -> function y -> (x > 4) & y;;
- :  int -> bool -> bool = <fun>

# (function f -> function g -> f + g) 3;;
```

Rappels : Typage d'expressions

- Donnez le type des expressions suivantes :

```
# function 0 -> 42 | x -> x + 1;;
- :  int -> int = <fun>

# function f -> 3 + (f 4);;
- :  (int -> int) -> int = <fun>

# function x -> function y -> (x > 4) & y;;
- :  int -> bool -> bool = <fun>

# (function f -> function g -> f + g) 3;;
- :  int -> int = <fun>
```

Rappels : Typage d'expressions

- Donnez le type des expressions suivantes :

```
# function 0 -> 42 | x -> x + 1;;
- :  int -> int = <fun>

# function f -> 3 + (f 4);;
- :  (int -> int) -> int = <fun>

# function x -> function y -> (x > 4) & y;;
- :  int -> bool -> bool = <fun>

# (function f -> function g -> f + g) 3;;
- :  int -> int = <fun>

# function y -> function f -> (string_of_float y)
^  (string_of_float (f y));;
```

Rappels : Typage d'expressions

- Donnez le type des expressions suivantes :

```
# function 0 -> 42 | x -> x + 1;;
- :  int -> int = <fun>

# function f -> 3 + (f 4);;
- :  (int -> int) -> int = <fun>

# function x -> function y -> (x > 4) & y;;
- :  int -> bool -> bool = <fun>

# (function f -> function g -> f + g) 3;;
- :  int -> int = <fun>

# function y -> function f -> (string_of_float y)
^  (string_of_float (f y));;
- :  float -> (float -> float) -> string = <fun>
```

Rappels : Fonctions à n paramètres

- Exemples :
  - fonction permettant de calculer la surface d'un rectangle à partir des coordonnées de 2 points du plan qui sont les coins inférieur gauche et supérieur droit du rectangle
  - fonction permettant de calculer la composé $f \ o \ f$ d'une fonction f

Rappels : Fonctions récursives

- Exemples :
  - fonction permettant de calculer les éléments d'une suite u_n telle que :
    $u_0 = 1$ et $u_n = u_{n-1}^2 + 3$
  - fonction permettant de calculer la sommes des i premiers éléments de u_n :
    $\sum_{k=0}^{k=i} u_k$
  - fonction permettant de calculer une valeur approchée (à ϵ près) du point fixe d'une fonction :

    $x = f(x) \Rightarrow f(x) - x \approx 0$

    (suite : $x_0, f(x_0), f(f(x_0)), \ldots$)

Rappels : Produits cartésiens

- Exemples :
  - fonction permettant de vérifier si une date (au format année, mois, jour) est antérieure à une autre
  - fonction permettant de vérifier si deux vecteurs de $\mathbb{R}^3$ sont orthogonaux

Rappels : Listes

- Exemples :
  - fonction permettant de calculer la moyennes des éléments d'une liste
  - fonction permettant de calculer le barycentre de points du plan (contenus dans une liste)
  - fonction permettant de concaténer les chaînes contenues dans une liste

Rappels : Enregistrements

- Exemples :
  - manipulation de coordonnées du plan (abscisse, ordonnée)
  - fonction prenant en entrée une liste de coordonnées correspondant aux sommets d'un polygone et retournant un couple contenant les coordonnées des coins gauche-bas et droite-haut d'un rectangle contenant le polygone

Rappels : Types sommes

- Exemples :
  - type permettant de modéliser une formule mathématique
  - fonction permettant de renommer toutes les occurrences d'une variable x en variable y
  - fonction permettant d'évaluer la formule pour une certaine valeur des variables
  - type permettant de modéliser une pile de données
  - fonctions permettant d'ajouter / retirer des éléments d'une pile

Rappels : Notions de complexité

- Exemples :
  - complexité du tri à bulles
  - complexité du tri fusion

Plan

Y'a-t-il une vie après AP1 (et Caml) ?

- Plusieurs paradigmes de programmation (et langages associés) :

  programmation **impérative** (perl, C), **fonctionnelle** (ML, Caml, Haskell), **orientée objets** (Smalltalk, Java, C++, Python), **logique** (Prolog, Oz), **spécialisés** (SQL, HTML, LaTeX, PHP, VRML)

- Liens entre langage de programmation et langage mathématique :

  `http://www.lix.polytechnique.fr/~dowek/Vulg/langagelangages.pdf`

Y'a-t-il une vie après AP1 (et Caml) ?

- Domaine vaste :
  - concepts de base (*e.g.* Algorithmique, Type de Données Abstraits)
  - spécialités (*e.g.* traitement d'images, réseaux, développement d'applications)
- Etudes en informatique : cursus universitaire (Masters), écoles d'ingénieurs
- Enseignements à venir (entre autres) : Environnements de programmation (linux), Programmation orientée objets (Java), bases de données (PostgreSQL), architecture des ordinateurs (processeurs, etc), structures de données (C), théorie des langages, projets, *etc*