Programmation fonctionnelle (Le langage CAML)

Dr Mouhamadou GAYE

UFR Sciences Et Technologies
Département d'Informatique
Licence 3 en Informatique Option Génie Logiciel

28 septembre 2020

Contenu du module

- Chapitre 1 : Bases du langage CAML
- Chapitre 2 : Fonctions
- Chapitre 3 : Listes

Évaluation

- Projet : **50%**
- Examen final : **50%**

Objectifs du cours

- Comprendre les principes de la programmation fonctionnelle
- Connaitre les éléments de base du langage CAML

Chapitre 1 : Bases du langage CAML

Paradigmes de programmation

- **Programmation impérative**
  Le but est de définir une liste d'instructions qui déterminent les modifications successives des états de la mémoire pour produire le résultat demandé.
- **Programmation déclarative**
  C'est un style de programmation où l'on décrit les relations qui existent entre les données et les résultats.
  - **Programmation logique** : les relations sont des formules logiques et le traitement est effectué par inférence.
  - **Programmation fonctionnelle** : les relations sont assimilées à des fonctions et le traitement est une combinaison de fonctions intermédiaires.

CAML

CAML (Categorical Abstract Machine Language) est un langage de programmation fonctionnelle.

- Il est fortement typé : le typage est automatique et ne nécessite pas d'annotation.
- Les instructions se terminent par ;; et le système interactif affiche le résultat après chaque instruction.

Types de base et opérateurs

- **int** : +, -, *, /, mod
- **float** : +., -., *., /.
- **bool** : &&, ||, not
- **char** :
- **string** : =, <=, >=, <>, <, >, ^ (concaténation)

Définitions

Une définition permet d'associer une valeur à un nom pour pouvoir la réutiliser.

- **Définition globale** :

  let nom = expression | valeur ; ;

  Exemple :

  let n = 2 + 3 ; ;

  val n : int = 5

  NB : Une fois défini, un nom a toujours la même valeur, il n'est pas modifiable.

Définitions

- **Définition locale** :
  Une définition locale permet de définir des valeurs qui ne sont valides que pour le calcul en cours.

  let nom = expression | valeur in expression ; ;

  Exemple :

  let n = 3 in 7 - n ; ;

  - : int = 4

  NB : Une définition locale est toujours suivie d'une expression et une expression peut contenir une définition locale.

Définitions

- **Définition multiple** :
  Une définition multiple est une succession de définitions (globales ou locales) séparées par le mot-clé **and**.

  let nom_1 = expression_1 | valeur_1 and nom_2 = expression_2 | valeur_2 ; ;

  Exemple :

  ```
  let n = 3 and m = 5 ; ;
  val n : int = 3
  val m : int = 5
  let x = 2 and y = 4 in x + y ; ;
  - : int = 6
  ```

  NB : Les définitions multiples se font en même temps et ne sont utilisables qu'une fois toutes les définitions terminées.

Analyse par cas

- **Alternative**

  Une alternative est une expression qui permet de faire des calculs dépendant d'une condition.

  if condition then expression_1 else expression_2 ; ;

  Exemple :

  ```
  5 * (if true then 3 else 2) ; ;
  - : int = 15
  ```

  NB : expression_1 et expression_2 doivent être du même type.

Filtrage

Le filtrage consiste à mettre en correspondance des motifs et des expressions.

- **Filtrage explicite**

```
match identifiant with
        motif_1 → expression_1
      | motif_2 → expression_2
      |....
      | motif_n → expression_n
```

Exemple :

```
let f x = match x with
        0 → 0
      | 1 → 2 * x
      | 2 → 2 / x ; ;
```

NB : Les expressions doivent être du même type qui sera le type de l'identifiant.

Filtrage

- **Motif universel**

  Le motif universel noté "_" est un motif qui s'approprie à toute valeur. Il peut être considéré comme le cas par défaut si toutes les valeurs ont été décrites.

  match identifiant with
  $\text{motif}_1 \longrightarrow \text{expression}_1$
  $\mid \text{motif}_2 \longrightarrow \text{expression}_2$
  $\mid \ldots.$
  $\mid \_ \longrightarrow \text{expression}_n$

Filtrage

- **Motif conditionnel**

  Le motif conditionnel permet de faire un filtre gardé c'est-à-dire un filtre qui dépend d'un test.

  $$\text{motif when condition} \longrightarrow \text{expression}$$

  Exemple :

```
let f x = match x with
          0 -> 0
        | y when y > 0 -> 2 * y ; ;
```