Algorithmique - Programmation 1

Cours 2

Université Henri Poincaré

CESS Epinal

Automne 2008

Plan

Rappel: Caml en mode interactif

Rappel: les types de base

Fonctions

Rappel: Caml en mode interactif

- Ocaml est un logiciel fonctionnant sous 2 modes : **compilateur** et **interpréteur** (mode interactif)
- En mode compilateur, Ocaml **traduit** un programme écrit en langage Caml en un code en langage assembleur (langage du processeur)
- En mode interpréteur, Ocaml exécute des instructions en langage Caml interactivement (**toplevel** de Caml)
- Le **prompt** (invite) de l'interpréteur Ocaml est le caractère #
- Pour chaque **expression** Caml fournie, l'interpréteur l'**évalue** et retourne **sa valeur et son type**

Rappel: Caml en mode interactif (suite)

- Exemples :

```
# 2 + 5;;
- : int = 7

# let x = 2 + 5;;
val x : int = 7

# x + 1;;
- : int = 8
```

- NB: mémoire de l'interpréteur

Rappel: Caml en mode interactif (suite)

- Erreurs les plus courantes :
  - Erreur de syntaxe (expression mal formée)
  - Erreur de type (données incompatibles)
  - Erreur d'algorithme (procédé de résolution erroné)

Plan

Rappel: Caml en mode interactif

Rappel: les types de base

Fonctions

Rappel: les types de base

Type **entier** (int)

- Valeurs comprises entre -2^{30} et $2^{30} - 1$
- Opérations : `+ - / * mod`
- Quelques fonctions prédéfinies :
  - `abs: int -> int = <fun>`
  - `int_of_float : float -> int = <fun>`
  - `string_of_int : int -> string = <fun>`
  - `max : 'a -> 'a -> 'a = <fun>`
- Exemples :

```
# abs (-3);;
# string_of_int(3);;
- : string = "3"
```

Rappel: les types de base

Type **réel** (float)

- Représentation : $m \times 10^e$
- Opérations : `+. -. *. /. sqrt **`
- Quelques fonctions prédéfinies :
  - `float_of_int : int -> float = <fun>`
  - `int_of_float : float -> int = <fun>`
  - `max : 'a -> 'a -> 'a = <fun>`
  - `cos tan sin ...`
- Exemple :

```
# sqrt (4.);;
```

Rappel: les types de base

Type **booléen** (bool)

- Valeurs : `true, false`
- Opérations : `&& or !`
- Utilisation dans les conditionnelles (`if then else`)
- Exemples :

```
# 4. > 2.;;
# (4 > 2) && (1 > 3);;
```

Plan

Rappel: Caml en mode interactif

Rappel: les types de base

Fonctions

Fonctions

Définition (Fonction)

Une fonction f d'un ensemble E vers un ensemble F est une **correspondance** *qui associe à* **chaque** *élément de E* **au plus** *un élément de F.*

- E est appelé **domaine** (de définition)
- F est appelé **codomaine**
- Signature de la fonction : $E \longrightarrow F$
  (`int -> int = <fun>`)

Fonctions : syntaxe

- Notation proche des mathématiques

  $f : x \longrightarrow f(x) \quad \approx \quad$ `# function x -> expr(x)`

- Exemple : `function x -> x + 1;;`
  - `x` est le paramètre
  - `x + 1` est une expression utilisant le paramètre `x`
  - fonction anonyme

- Définition par cas (pattern matching) :

```
# function 0 -> valeur
           | x -> expr(x)

# function 0 -> 1
           | x -> 1/x
```

Définition de fonction

- fonction anonyme :

```
# function x -> x *. x ;;
```

- fonction nommée :

```
# let f = function x -> x *. x;;
```

ou

```
# let f x = x *. x;;
```

- NB: application de fonction via

```
# f 3.;;
# (f 3.);;
# f (3.);;
# (f (3.));;
```

Application d'une fonction et typage

Définition (Application)

L'application (appel) d'une fonction est le calcul de la valeur de la fonction en un point donné.

- Attention à la notation (cf transparent précédent)
- Typage de l'appel d'une fonction :

  `(f x)` a le type `t2` ssi
  - `f` a le type `t1 -> t2`
  - `x` a le type `t1`

Evaluer l'application d'une fonction

Evaluer `(f x)`

1. réécriture de `f` jusqu'à obtenir une valeur fonctionnelle
2. réécriture de `x` jusqu'à obtenir une valeur
3. évaluation de `f` sur `x`
   - 3.1 liaison entre `x` et le paramètre de `f`
   - 3.2 réécriture de `f` avec la valeur de `x`

Evaluer l'application d'une fonction, exemples

```
# let a = 2;;
# let f = function x -> x + a;;

# (f 3) * ((function x -> x + 3) (3*5));;

# let g = function n -> (function p -> p + 1) n;;

# (g 1) + (g 1);;
```

Fonction et typage

- Type fonctionnel : `t1 -> t2`
  - `t1` type du domaine
  - `t2` type du codomaine
- A un type fonctionnel correspond une valeur fonctionnelle
- Le type fonctionnel exprime une correspondance entre deux types (ensembles)

  $\longrightarrow$ NE PAS CONFONDRE avec la valeur en un point !
- NB: une fonction est une valeur !
  - le résultat d'un calcul peut être une fonction
  - les données d'un calcul peuvent être des fonctions
  - on peut lier un nom à une fonction

Règles de typage fonctionnel

Règle 1 On cherche à construire une expression de la forme `t1 -> t2`

Règle 2 La technique est similaire à la résolution d'un système d'équations

- Y a-t-il une valeur évidente pour un des deux types ?
- Quelle valeur en déduire alors pour l'autre ?

Règle 3 Tous les cas doivent avoir le même type (pattern matching)

Type de : `function x -> float_of_int (x) +. 1.` ?

Exercice de typage

```
# function k -> k*k - 3*k + 1;;

# function k -> 220.0 *. sin (float_of_int(k)*.omega);;

# function z -> (z 2) & (z 4);;

# function 0.0 -> 1.0
         | x -> exp (x *.(log x));;

# function f ->  function x ->
      ((f (x +. h)) -. (f x)) /. h ;;
```

Notion de fermeture

Définition (Fermeture)

On appelle fermeture la valeur obtenue par application successive de réécritures d'expressions (tant qu'il reste des noms à remplacer).

Exemple :

```
# let a = 1;;
# function n -> n + a;;
```

Evaluer la fonction revient à :
1. remplacer a par 1
2. évaluer `n -> n+1` (fermeture puisqu'il ne reste que le paramètre, NB: c'est une valeur fonctionnelle)

Remarques

- Une fonction peut prendre plusieurs paramètres (arguments) :

```
# function x -> function y -> 2. *. (x +. y);;
```

- Une fonction peut être l'argument d'une autre fonction :

```
# function f -> function x -> (f x);;
```

- Une application de fonction en ne fixant qu'une partie de ses arguments produit une valeur fonctionnelle :

```
# let g = function x -> function y -> x + y;;
val g : int -> int -> int = <fun>;;
# let h = function x -> (g x);;
val h : int -> (int -> int) = <fun>;;
```