Langages et Programmation 2

TP1 - Introduction à OCAML

Objectifs

Il s'agit de se familiariser avec l'environnement de développement EMACS couplé au mode tuareg et de coder des fonctions fréquemment utilisées lorsque l'on programme en CAML.

La première partie permet de s'interroger sur les différents paradigmes du langage à travers des exercices "jouets". La seconde partie est dédiée à la structure de données des listes, dont vous implémenterez ses fonctions de manipulation classiques.

Il est **fortement conseillé** de tester vos fonctions avec une ou plusieurs entrées quelconques. Toutes les constructions syntaxiques qui vous seront nécessaire ne sont pas données ou décrites dans le détail : référez-vous à la documentation en ligne[1].

1 Découverte d'OCaml

En CAML un programme consiste à déclarer un ensemble d'expressions. Le mot clé `let` vous permet de définir une expression, par exemple `let x = 2` définit une "valeur" `x` qui vaut 2. Vous pouvez réutiliser les valeurs précédemment définies, par exemple dans `let y = x - 1`. Vous disposez également de conditionnelles : `let z = if (x = 2) then 10 else 20` défini la valeur `z` selon la valeur de `x`.

Exercice 1 Évaluez les exemples d'expressions ci-dessus dans EMACS. Remarquez l'évaluation de `z`.

Des fonctions Vous pouvez également déclarer des fonctions, de la manière suivante : `let f = fun a -> a - 1`. La fonction définie (`f`) est une fonction de `a` (`fun a`) qui renvoie (`->`) la valeur de `a` moins un. Les fonctions peuvent être passées en arguments comme tout autre élément : on parle de programmation fonctionnelle. C'est par ce biais que le *calcul* s'effectue, passant de fonction en fonction.

Exercice 2 Écrivez une fonction `f1` (resp. `g1`) qui calcule la multiplication de trois (resp. deux) entiers et `f2` (resp. `g2`) leur somme. Écrivez une fonction `foo` qui permet de calculer la somme ou la multiplication de trois entiers.

[1] `http://caml.inria.fr` dans "resources" suivre "Objective Caml manual" puis "The core language"

Du typage Chaque expression que vous pouvez former en CAML a un type (c'est une propriété des langages typés). Le compilateur, comme l'évaluateur, calcule le type de toute expression et vérifie que celui-ci existe. Puisqu'il s'agit d'un langage fonctionnel, un des types de base est celui des fonctions que l'on écrit à l'aide de la flèche $t_1 \rightarrow t_2$ qui signifie qu'une fonction ayant ce type transforme un élément de type t_1 en un élément de type t_2. Les t_i peuvent être des types de base, comme des fonctions. Enfin il existe les types dits anonymes ou encore polymorphes, notés par des lettres grecques, qui représentent un type quelconque existant.

Exercice 3 Donnez le type des fonctions `g1`, `g2` et `foo`. Donnez également celui des fonctions : `f3 = fun x -> x`, `f4 = fun x -> fun y -> y x` et `f5 = fun x -> x x`. Enfin, pour chacun des types suivants donnez une fonction ayant ce type : *(int* $\rightarrow \alpha$*)* $\rightarrow \alpha$, *(int* $\rightarrow \alpha$*)* $\rightarrow$ *(int, int)* $\rightarrow \alpha$ et $\alpha \rightarrow \beta$.

De l'évaluation partielle On peut évaluer certains paramètres d'une fonction et ainsi créer une instance spécialisée de celle-ci[2]. Considérer la fonction suivante qui incrémente une variable entière `x` de `i` et utilise un effet de bord : `f = fun i -> print i ; fun x -> x + i`. La fonction `g = f 2` est l'évaluation partielle de `f` pour `i` égale 2.

Exercice 4 Utilisez les fonctions `f` et `g`, et remarquez le traitement de l'effet de bord. Donnez les fonctions `h1` et `h2`, respectivement multiplicateur et additionneur sur trois entiers, à partir de la fonction `foo` de l'exercice 2.

De la récursion Lorsque l'on définit la fonction factorielle comme suit : `f = fun x -> if (x = 0) then 1 else x` $\times$ `f (x-1)`, on voit que le retour du premier appel à `f` intervient que lorsque le calcul des multiplications atteint le cas de base (factorielle zéro). Ainsi pour factorielle n c'est n appels de fonctions qui sont empilés. On dit que le fonction est non-récursive terminale.

Exercice 5 Donnez une version récursive terminale de factorielle. *(plus difficile)* Faites de même pour la suite de Fibonacci ($\{1, 1, 2, 3, 5, 8, 13, \ldots\}$).

2 Les listes d'entiers

Un type représente un ensemble d'éléments sémantiquement cohérent. Des constructeurs sont associés au type afin de pouvoir en construire un élément quelconque. Par exemple le type booléen détient deux constructeurs, `True` et `False`. Les parenthèses et la virgule sont des constructeurs de couples. Le mot clé `let` qui vous permet de déclarer une fonction est bien un constructeur du type fonctionnel.

En CAML vous avez à votre disposition les types "sommes", qui vous permettent de déclarer un type par l'union de ses constructeurs. Par exemple le type booléen n'est autre que l'union de deux constructeurs : `type Booleen = True | False`. On peut également déclarer des constructeurs fonctionnels, qui prennent en arguments un ou des types et construisent un élément du type. Ainsi déclare-t-on les couples, par exemple d'entiers : `type Couple_integer = Couple of int * int`.

[2] *c.f.* Théorème S_{mn} de Kleene

Le couple (5,7) s'écrie alors `Couple (5,7)`. Notez que vous pouvez utiliser comme type celui même que vous êtes en train de définir.

Exercice 6 Donnez le type `Integer_list` des listes d'entiers. On souhaite pouvoir considérer la liste vide. Déclarez quelques listes tests pour la suite.

Aux constructeurs de type sont couplés des destructeurs. Pour les fonctions il s'agit tout simplement de l'application (par exemple une fonction de type $int \rightarrow bool$ appliquée à un entier renvoie une valeur de type booléen). Pour déconstruire les types “sommes” vous avez à votre disposition l'instruction `match ... with ...` qui se base sur les mêmes constructeurs que ceux que vous avez introduits dans la définition du type. Par exemple, pour récupérer le deuxième élément d'un couple contenu dans la variable `c` on écrit : `match c with Couple (i1, i2) -> i2`.

Exercice 7 Écrivez les fonctions : `length` qui renvoie la longueur d'une liste et `equal` qui renvoie vraie si deux listes sont identiques. Donnez leurs types.

Exercice 8 Écrivez les fonctions : `head` qui renvoie le premier élément d'une liste, `tail` qui renvoie une liste amputée de son premier élément et `nth` qui renvoie le nième élément d'une liste. On pourra s'intéresser au mécanisme des exceptions. Donnez les types des fonctions.

Exercice 9 *(plus difficile)* Écrivez la fonction `inverse` qui renvoie l'inverse d'une liste. On pourra s'inspirer de l'exercice 5.

Notes

Vous trouverez sur le site des TPs[3] de nombreuses ressources : pointeurs vers des cours en ligne et la documentation caml, une bibliographie, les sujets de TPs, etc.

Ce TP fera l'objet d'une évaluation durant la scéance du 8 février. Regroupez l'ensemble des solutions (commentées et agrémentées de tests pertinents !) aux différents exercices dans un fichier `tar`, nommé `grXX-tp1.tar` où `XX` est votre numéro de groupe. Envoyez le par mail à l'adresse `mathias.peron@imag.fr` avant le 5 février minuit.

[3] `http://www-verimag.imag.fr/~peron`. Suivre “enseignement” puis “LP2”