TP1 : Introduction au langage Caml

Christophe Rose

Vendredi 18 janvier 2008

1 L'interpréteur Caml Light

Caml Light est une implémentation du langage Caml. Il est disponible sur de nombreuses plateformes comme Windows, MacOS X, Linux et la plupart des Unix.

Les différentes versions sont téléchargeables à l'adresse suivante :

`http://caml.inria.fr/caml-light/release.fr.html`

Après avoir lancé Caml Light, on obtient un terminal contenant les lignes suivantes :

```
>       Caml Light version 0.75

#
```

Le dièse `#` signifie que vous pouvez commencer à écrire dans le terminal.

On indique la fin d'une commande par deux points-virgule à la suite `;;` et par un retour charriot (la touche Entrée). Un saut à la ligne sans les `;;` est considéré comme un espace par Caml Light.

```
#2;;
- : int = 2
#let x = 2;;
x : int = 2
#let
x
=
2
;;
x : int = 2
```

À chaque fois qu'on entre une commande, Caml répond par une expression de la forme `nom : type = valeur`, puis rend la main à l'utilisateur en indiquant un `#`.

Il arrive parfois que Caml ne rend pas la main au bout de certain temps.

- Soit le calcul demandé est très long.
- Soit Caml est entré dans une boucle infinie.
- Il existe plusieurs commandes pour interrompre le calcul en cours, qui dépendent de la version de Caml et de la plateforme utilisée. Souvent, `^C` (Ctrl-C) fonctionne.

Dans ce TP, certaines questions vous demandent de taper plusieurs commandes.

1. Avant de saisir et d'exécuter chaque commande sur Caml Light, prévoyez quel sera le résultat.
2. Interprétez ce que Caml vous répond.
3. Si une erreur se produit, essayez de comprendre le message d'erreur, et déduisez-en pourquoi l'erreur s'est produite. Éventuellement, corrigez l'erreur.

2 Types, définitions et opérations élémentaires

Caml est un langage fonctionnel orienté objet. Tout ce que l'on manipule (constantes, variables, fonctions, etc.) sont des objets. Ils sont distingués par leur nom, leur type et leur valeur.

La commande suivante :

```
let x = 2 ;;
x : int = 2
```

définit un objet appelé `x`, qui a pour valeur `2`, et qui a le même type que `2`. Cet objet est donc de type `int`, c'est-à-dire un entier relatif (*integer* en anglais). Nous verrons aussi le type `float` des flottants, qui forment une bonne représentation des réels.

Question 1. Tapez les commandes suivantes :

```
2+4;;
let x = 3 + 1 ;;
-4*(3+2);;
2-1*0+3 ;;
```

Les parenthèses indiquent quelle est la priorité à respecter lors des calculs. La plupart du temps, Caml calcule de la gauche vers la droite, mais les priorités entre les opérations `+ - * /` sont respectées.

Question 2. Tapez les commandes suivantes :

```
1 + 2.5;;
1. +. 2.5;;
4.*.1.2 -. 0.8;;
2. **. 10.;;
3e5-.1e1;;
```

Caml se sert des types pour détecter une erreur. Par exemple, on ne peut pas additionner des `int` et des `float`. Il y a même deux opérateurs distincits `+` et `+.` pour l'addition.

Question 3. Tapez **dans l'ordre** les commandes suivantes :

```
2*x+1;;
x = 7;;
let x = 7;;
2*x+1;;
let y = 2*x+1;;
let x = y;;
let x = (x-1)/2;;
```

Les définitions en Caml peuvent être globales ou locales. Quand on définit globalement une variable déjà définie, sa valeur précédente est écrasée. On utilise la syntaxe `let ... in ...` pour faire une définition locale.

Question 4. Tapez dans l'ordre les commandes suivantes :

```
let x=0 ;;
let x=1 in 10*x;;
let x=1 in let y=10*x;;
let y = let x=1 in 10*x;;
x;;
```

Toutefois, il est préférable de ne pas utiliser le même nom pour une variable définie localement et pour une variable définie globalement.

3 Fonctions

Comme en mathématiques, on peut définir des fonctions en Caml.

Question 5. Tapez les commandes suivantes :

```
let f x = 2*x+1;;
f(0);;
f(10);;
f(x);;
f(0.5);;
f(nondef);;
```

Lors de la définition de la fonction `f`, `x` est une variable muette, donc locale. Ici, lorsqu'on appelle `f(x)`, Caml utilise la définition globale de `x`.

Attention ! La syntaxe `f(x)` ne permet pas de remonter à la définition de `f`. Et pire, si `x` est déjà défini, on obtient un résultat aberrant au lieu d'un message d'erreur normal, et on peut ne pas s'en rendre compte !

Une fonction est vue comme un objet pour Caml, elle a donc un type. Dans le cas précédent, `f` a pour type `int -> int`. Le premier `int` est le domaine de définition de la fontion, c'est-à-dire le type de la variable muette `x`. Le deuxième `int` est l'ensemble d'arrivée de la fonction.

On peut définir les fonctions de plusieurs façons :

```
let suivant = fun x -> x+1;;
let suivant = function x -> x+1;;
let suivant x = x+1;;
let suivant(x) = x+1;;
```

et les utiliser de plusieurs façons :

```
suivant 3;;
suivant(3);;
let x=3 in suivant(x);;
```

Question 6. Tapez les commandes suivantes :

```
let id x = x;;
id(3);;
id(4.5);;
id(id);;
(id id)(3);;
id (id 3);;
```

Lorsqu'on définit une fonction, Caml essaye de deviner son type. Mais ce n'est pas toujours possible. Caml utilise alors des types muets. La notation `'a->'a` signifie que la fonction `id` a le même ensemble de départ que d'arrivée.

4 Booléens et tests

Caml possède le type booléen `bool` qui peut prendre l'une des deux valeurs suivantes : `true` et `false`.

Question 7. Tapez les commandes suivantes :

```
1 < 2 ;;
1 <= 1 ;;
1<2 or 1.>2.;;
not 1=2 & 1<1;;
if 1<2 then 3 else 4;;
if true then 0 else 0.;;
if true
then function x -> x
else function x -> x+1;;
if false
then (if false then true else true=false)
else (true);;
```

Les tests vont nous permettre de construire des fonctions plus complexes que les seules compositions d'opérations algébriques.

Question 8. Programmez une fonction `va_int` de type `int -> int` qui prend en argument un entier relatif n et qui renvoie sa valeur absolue $|n|$. Programmez une fonction `signe_int` de type `int -> int` qui prend en argument un entier relatif n et qui renvoie 1 s'il est positif, -1 s'il est négatif et 0 s'il est nul.

Question 9. Programmez les fonctions `va_float` de type `float -> float` et `signe_float` de type `float -> int`.

5 Types produits, fonctions à plusieurs variables

Question 10. Tapez les commandes suivantes ;

```
(true,0);;
1,0;;
fst(1,2);;
snd(1,2);;
1,(2,3);;
1,2,3;;
fst(1,2,3);;
();;
let f() = 2;; f();;
```

Les types « produits cartésiens » permettent de définir des fonctions à plusieurs variables.

Question 11. Tapez les commandes suivantes :

```
let distance (x,y) =
if x <= y then y-x else x-y;;
distance (5,3);;
```

Il existe une autre manière de définir une fonction à plusieurs variables.

Par exemple, si on veut définir la fonction $f : \mathbb{R}^* \times \mathbb{R} \to \mathbb{C} : (x, y) \mapsto x^2 + y/x$, on peut commencer par définir les fonctions $f_x : \mathbb{R} \to \mathbb{C} : y \mapsto x^2 + y/x$, puis définir la fonction $\widehat{f} : \mathbb{R}^* \to \mathcal{F}(\mathbb{R} \to \mathbb{C}) : x \mapsto f_x$.

Ainsi, $f(x, y) = (\widehat{f}(x))(y)$. Les fonctions f et $\widehat{f}$ donnent un résultat identique pour x et y donnés, mais leur syntaxe n'est pas la même.

Question 12. Que font les fonctions suivantes et quels sont leurs types ?

```
let curry f = function x -> (fun y -> f(x,y));;
let uncurry g = function (x,y) -> (g(x))(y);;
```

On appelle la forme `f(x,y)` d'une fonction la forme *décurryfiée*, tandis que `(f(x))(y)` est appelée la forme *curryfiée*.

Question 13. Que font les fonctions suivantes :

```
let id f = f;;
let evalue f x = f x;;
```

Question 14. Dans quels cas `id f` et `evalue f` ne sont pas la même chose ?

6 Conventions des différents types

Question 15. Tapez les commandes suivantes :

```
5/5 ;;
4/5 ;;
4/5-1;;
(-1)/5;;
14 mod 5;;
(-1) mod 5;;

100000*100000;;
let x=512*1024*1024;;
2*x-1;;
2*x;;
2*x+1;;
4*x-1;;
4*x+1;;
```

Il existe une infinité d'éléments de $\mathbb{Z}$, et avec la mémoire d'un ordinateur, aussi grande qu'elle soit, on ne peut coder qu'un nombre fini d'entiers. On pose donc une limite arbitraire sur la taille des entiers. En Caml Light, le type `int` correspond à l'intervalle $[\![-2^{30}; +2^{30} - 1]\!]$.

Par conséquent, il faut avoir en tête que `a<b` et `c<d` n'implique pas toujours `a+c<b+d` quand on utilise des grands nombres. C'est l'origine de nombreux bogues (*integer overflow*).

Question 16. Tapez les commandes suivantes :

```
1/0 ;;
1 mod 0;;
1./.0. ;;
1./.(-0.) ;;
(-1.)/.0. ;;
infinity+.neg_infinity;;
-1.**.0.5;;
0.**.0.;;
0.**.(-1.);;
sqrt(2.);;
sqrt(-1.);;
```

Comme en mathématiques, certaines opérations comme la division par zéro ne sont pas permises. Cependant, le type `float` accepte des opérations qui sont interdites dans $\mathbb{R}$.

Question 17. Tapez les commandes suivantes :

```
int_of_float(3.);;
int_of_float(3.9);;
int_of_float(3.1);;
int_of_float(-0.1);;
int_of_float(-0.9);;
int_of_float(2147e6);;
int_of_float(2148e6);;
float_of_int(5)+.3.14;;
```

Question 18. Testez les fonctions suivantes sur des flottants bien choisis :

```
cos sin tan
acos asin atan
cosh sinh tanh
exp log
sqrt
```

Il existe aussi les types `char` et `string`.

Question 19. Tapez les commandes suivantes :

```
'a';;
"a";;
'a'="a";;
'a' < 'b';;
"yzzz" < "za";;
"ab".[1];;
let s="abc";;
s.[0]<-'A'; s;;
```

7 Interactivité

Question 20. Que sont les types des fonctions suivantes, que font-elles et quelles sont leurs limites ?

```
string_of_int
string_of_float
string_of_bool
int_of_string
float_of_string
bool_of_string
string_of_char

print_int
print_float
print_char
print_string
print_newline

read_int
read_float
read_line

random__int
```

Question 21. Écrire un programme `multi` de type `unit -> unit`, qui choisit deux entiers entre 5 et 40, et qui demande à l'utilisateur d'en donner le produit. Si la réponse est incorrecte, le programme donne la bonne réponse.

Les objets dans Caml sont sensés rester constants depuis qu'ils sont définis jusqu'à la fin de la session. En Caml, on utilise des *références* pour utiliser des variables qui changent de valeur au cours du temps.

Les références se définissent ainsi : `let x=ref(0);;`. On lit et on écrit dans une référence comme ceci : `!x ;; x:=5 ;;`.

Question 22. En utilisant les boucles inconditionnelles `while ... do ... done`, ainsi que les références, faites en sorte que le programme précédent pose des questions jusqu'à ce que l'utilisateur donne une mauvaise réponse, et indique le nombre de bonnes réponses consécutives à la fin.