Cours 01 - Introduction à Caml

MPSI - Prytanée National Militaire

Pascal Delahaye

20 janvier 2009

Le langage informatique utilisé dans ce cours d'informatique est CAML (Categorical Abstract Machine - Meta-Language). Le logiciel de programmation en Caml (caml-light) est disponible gratuitement à l'adresse suivante : `http://pauillac.inria.fr/caml/distrib-caml-light-fra.html`

Ce premier cours donne une présentation rapide de ce langage.

N'hésitez pas à consulter le menu aide de CAML en cas de difficulté !!

En particulier, vous pourrez consulter "the core library" pour une liste des fonctions usuelles de Caml.

1 Un langage fortement typé

Tous les objets manipulés par Caml ont un *type*.
Les types les plus fréquemment rencontrés sont les suivants :

Type	code	exemple 1	exemple 2
Nombre entier	int	2	-2
Nombre réel	float	3.	sqrt 3.
Caractère	char	'h'	
Chaine de caractère	string	"vive caml"	"Erreur d'argument"
Liste	type1 list	[1 ; 2]	[2.1 ; sqrt 3. ; atan 1.]
Vecteur	type1 vect	[|1 ; 2|]	[|2.1 ; sqrt 3. ; atan 1.|]
Type inconnu 1	'a	[ ]	[| |]
Type inconnu 2	'b	[ ],[ ]	[| |],[| |]
Booléen	bool	true	(1=2)
Fonction d'une variable	type1 $->$ type2	int_of_float	min
Action Caml	unit	print_int	v.(2)<-1

Remarque 1.

1. Ces types vous apparaitront peut-être contraignants, mais ils aident le programmeur en :
   (a) l'obligeant à identifier systématiquement la nature des objets qu'il manipule.
   (b) lui donnant des informations sur les erreurs de programmation éventuelles.
2. Le type `unit` correspond à une modification de l'état de la machine.
3. Il existe encore bien d'autres types que l'on rencontrera progressivement au fil des cours.

1.1 Les types de nombres

Exercice : 1

Familiarisation avec les types `int` et `float`
Taper les instructions suivantes et commenter la réponse de Caml dans le tableau suivant :

Instructions	Réponse	Commentaires
let A = 2 ;;		
let A = 3. ;;		
let A = 1 + 2 ;;		
let A = 1 + 2.1 ;;		
let A = 1.2 + 2.1 ;;		
let A = 1.2 +. 2.1 ;;		

Remarque 2.

1. Une série de commandes Caml se termine toujours par `;;`
2. La création (on dit aussi "la déclaration") d'une variable (qui s'exprime sous la forme `> NOM := ...` sous Maple) commence ici toujours par `let NOM = ...`.
3. On prendra l'habitude de mettre des espaces avant et après chaque opérateur.

A retenir !!

1. On ne peut faire d'opérations qu'entre éléments de même type. Ainsi :

2. Les opérateurs associés aux entiers sont différents des opérateurs associés aux réels !!

   Opérateurs entre entiers : + - / * (pas d'opérateur "puissance")
   Opérateurs entre réels : +. -. /. *. **.

3. Les fonctions associées aux entiers sont différentes des fonctions associées aux réels !!

Fonctions sur des entiers :	abs	mod	pred	succ	min	max	float_of_int	random__int
Fonctions sur des réels :	abs_float	exp	log	sqrt	min	max	int_of_float	random__float
	sin	sinh	asin					

Exercice : 2

Tester sous Caml (durant 5 mn) les différents opérateurs et fonctions précédentes.

Remarque 3. Les fonctions `min` et `max` s'appliquent aussi à des couples ...

Remarque 4. **Représentation machine des nombres**

1. Les nombres entiers sont codés en binaire sur 31 bits, le premier bit correspondant au signe.
   Cela signifie qu'il est possible de manipuler des entiers compris entre -2^{30} et $2^{30}-1$.
   (a) Calculer la valeur de $2^{30}-1$ (utiliser `int_of_float`), puis ajouter 1, puis 2 à cette valeur ...
   Que constatez-vous?
   (b) Taper : `int_of_float (2. ** 32.) ;;`
   Que constatez-vous?
   La manipulation informatique de grands nombres entiers peut donc donner des résultats abberrants?

2. Les nombres réels (appelés *flotants*) sont codés sur 64 bits par un bit de signe, une *mantisse* entière et une puissance de dix. Les valeurs limites seront donc très rarement atteintes.

Remarque 5.

1. Caml n'est pas un programme de calcul formel. Il ne connaît donc pas la notion de nombre irrationnel.
   le nombre π pourra donc être défini de façon approchée par : `let pi = 4.*.atan(1.);;`
2. Vous avez constaté qu'il n'existe pas d'opérateur *puissance* sur les entiers.
3. Il existe aussi un type spécial pour les nombres rationnels, mais celui-ci est assez peu employé.

1.2 Les types de "tableau ligne"

Exercice : 3

Familiarisation avec les types `liste` et `vect`

Taper les instructions suivantes et commenter la réponse de Caml dans le tableau suivant :

Instructions	Réponse	Commentaires
let L1 = [1;2;4;8] ;;		
list_length L1 ;;		
let L2 = [1,2,4,8] ;;		
list_length L2 ;;		
let V1 = [|1;2;4;8|] ;;		
vect_length V1 ;;		
let V2 = [|1,2,4,8|] ;;		
vect_length V2 ;;		
let V = vect_of_list L1 ;;		
let L = list_of_vect V1 ;;		
V = V1 ;;		
L1.(1) ;;		
V1.(1) ;;		
V1.(1) <- 0 ;;		
let L3 = [1;2.] ;;		
let V3 = [|1;2.|] ;;		

A retenir pour l'instant !!

1. Les éléments d'une liste ou d'un vecteur sont tous du même type.
2. Les éléments d'un vecteur de longueur n sont numérotés de 0 à $n-1$.
3. Les longueurs d'une liste et d'un vecteur sont données par les fonctions `vect_length` et `list_length`
4. On ne peut pas récupérer facilement une composante quelconque d'une liste.
5. On peut récupérer facilement une composante quelconque d'un vecteur.
6. On ne peut pas modifier les composantes d'une liste.
7. On peut modifier les composantes d'un vecteur.
8. Les fonctions `vect_of_list` et `list_of_vect` permettent de transformer un vecteur en liste et inversement.

Remarque 6. Nous verrons plus tard que le type Vecteur sera utilisé en programmation itérative alors que le type Liste sera utilisé en programmation récursive.

Exercice : 4

Compléments sur le type `liste`

Taper les instructions suivantes et commenter la réponse de Caml dans le tableau suivant :

Instructions	Réponse	Commentaires

Instructions	Réponse	Commentaires
let L1 = [1;2;4;8] ;;		
hd L1 ;;		
tl L1 ;;		
let L2 = 0::L1 ;;		
let L3 = L1@L2 ;;		

A retenir pour l'instant !!

1. On ne peut pas obtenir directement la valeur d'une composante quelconque d'une liste, mais on peut en revanche :
   (a) Obtenir la valeur en tête de liste par la commande `hd L;;` ("hd" signifiant "head")
   (b) Obtenir la liste moins la première composante par la commande `tl L;;` ("tl" signifiant "tail")
2. On peut ajouter un terme `a` en tête d'une liste `L` par la commande `let L1 = a::L;;`
3. On peut concaténer une liste `L1` à une liste `L2` par la commande `let L = L1@L2;;`

`Exemple 1.` Déterminer deux algorithmes très simples permettant de :
1. extraire la p^{eme} composante d'une liste
2. modifier la p^{eme} composante d'une liste

`Exemple 2.` Comment définir sous Caml la variable : $T = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}$?

1.3 Les types de "mots"

Exercice : 5

Familiarisation avec les types `char` et `string`

Taper les instructions suivantes et commenter la réponse de Caml dans le tableau suivant :

Instructions	Réponse	Commentaires
let carac = 'l' ;;		
let mot = "'addition s'il vous plait !" ;;		
carac ^ mot ;;		
let carac = string_of_char carac ;;		
carac ^ mot ;;		

A retenir pour l'instant !!

1. Les caractères et les chaines de caractères ont des types différents. (`char` et `string`)
2. On peut transformer un caractère en chaîne de caractères avec la fonction `string_of_char`
   L'opération inverse est impossible ...
3. On peut concaténer deux chaînes de caractères entre elles par la commande `let Chaine = ch1 ^ ch2;;`

Remarque 7. **Compléments**
1. La fonction `string_length` permet de déterminer la longueur d'une chaîne de caractère.
2. Les chaînes de caractères se comportent comme des vecteurs dont les éléments sont des caractères.
   (a) On obtient le n^{eme} caractère d'une chaîne NOM par la commande `NOM.[n-1];;`

(b) On peut modifier le n^{eme} caractère d'une chaîne NOM par la commande `NOM.[n-1] <- 'A';;`

Appliquer ces manipulations sur des exemples de votre choix !

1.4 Les n_uplets

A partir des types de base, on peut générer de nouveaux ensembles à la façon des produits cartésiens d'ensembles. On crée ainsi de nouveaux types.

Exercice : 6

Familiarisation avec les types `uplet`

Taper les instructions suivantes et commenter la réponse de Caml dans le tableau suivant :

Instructions	Réponse	Commentaires
let couple1 = (1,2) ;;		
let couple2 = (3,1) ;;		
couple1 = couple2 ;;		
couple1 > couple2 ;;		
min couple1 couple2 ;;		
let uplet1 = (”toto”, 2, false) ;;		
let uplet1 = ”toto”, 2, false ;;		

A retenir pour l'instant !!

1. Il est possible de créer des n-uplets de longueur quelconque
2. Les parenthèses ne sont pas indispensables mais sont très utiles pour garantir la lisibilité des commandes.
3. Les composantes des n-uplets peuvent être de types différents
4. Il est possible d'effectuer des tests de comparaison sur des n-uplets d'entiers ou de flottants.

1.5 Les booléens

Les booléens sont utilisés dès lors que la réalisation d'une commande est conditionnelle à la vérification d'une condition (de nature booléenne).

Exercice : 7

Familiarisation avec le type `booléen`

Taper les instructions suivantes et commenter la réponse de Caml dans le tableau suivant :

Instructions	Réponse	Commentaires
let bool1 = (2>=3) ;;		
let bool2 = (”toti”<>”toto”) ;;		
bool1 & bool2 ;;		
bool1 or bool2 ;;		
let bool3 = not (3 < 2) ;;		

A retenir pour l'instant !!

1. le booléen "A et B" s'écrit `A & B`
2. le booléen "A ou B" s'écrit `A or B`
3. le booléen "contraire de A" s'écrit `not A`
4. Pour comparer des entiers on utilise les comparateurs standards : <, >, =, >=, <= et <>
5. Pour comparer des reéls on utilise les comparateurs standards : <., >., =., >=., <=. et <>.
6. Pour comparer des chaînes de caractères on utilise les comparateurs standards : = et <>.
   Il en existe d'autres basés sur l'ordre lexicographique.

1.6 Modification du type d'une variable

Il est possible de transformer le type d'un élément par la fonction `type1_of_type2`.

Exercice : 8
Transformer le type d'éléments de votre choix.

Transformation du type	Code	résultat
Entier en réel		
Réel en entier		
Réel en chaine		
Entier en chaine		
Chaine en entier		
Chaine en réel		
Caractère en chaine		
Booléen en chaine		
Vecteur en liste		
liste en vecteur		

A retenir pour l'instant !!

1. On peut changer le type d'un élément de `type2` en un élement de `type1` avec la commande `type1_of_type2`.
2. Mais attention aux commandes `int_of_char` et `char_of_int` !!
   Celles-ci permettent de déterminer le code ASCII d'un caractère donné ou de trouver le caractère associé à un code ASCII donné.

`Exemple 3.`

1. Déterminer le code ASCII du caractère `@`
2. Déterminer le caractère dont le code ASCII est 35

Les caractères d'imprimerie ainsi que la tabulation et le "retour chariot" sont codés sur 8 bits. Cela donne en tout 256 caractères. Vérifiez cela à l'aide de `char_of_int`...

Remarque 8. Nous verrons plus tard que Caml permet à l'utilisateur qui le désire, de créer ses propres types.

2 Déclaration des constantes

Les constantes sont des variables dont la valeur reste constante au cours du projet.
Il y a deux types de constantes :

1. Les constantes globales

2. Les constantes locales

Exercice : 9

Familiarisation avec les deux types de constantes

Taper les instructions suivantes et commenter la réponse de Caml dans le tableau suivant :

Instructions	Réponse	Commentaires
let A = 25 ;;		
let B = 2 in 3*B ;;		
A ;;		
B ;;		

A retenir !!

1. Les constantes globales sont définies par l'instruction `let NOM = valeur ;;`
2. Les constantes locales sont définies par l'instruction `let NOM = valeur in ... ;;`
3. Vous rencontrerez parfois le message d'erreur : `The value identifier B is unbound.` Il signifie que la variable `B` est inconnue de Caml (et donc que sa déclaration a échoué).

Exercice : 10

Déclaration de plusieurs constantes locales :

Taper les instructions suivantes et commenter la réponse de Caml dans le tableau suivant :

Instructions	Réponse	Commentaires
let A = 2 and B = 3 in A + B ;;		
let A = 2 and B = 3*A in A + B ;;		
let A = 2 in let B = 3*A ;;		
let B = let A = 2 in 3*A ;;		
let A = 2 in let B = 3*A in A + B ;;		

A retenir !!

1. Une variable globale B dépendant d'une autre variable A se déclare de la façon suivante : `let B = let A = ... in ...`
2. On peut créer des constantes locales à des constantes locales. Pour cela on utilise la syntaxe `let NOM1 = valeur1 in let NOM2 = valeur2 in ..procedure.. ;;`. Ici, la variable `NOM1` est locale à la constante `NOM2` qui elle-même est locale à la procédure.
3. On utilise la commande `and` lorsqu'on souhaite déclarer plusieurs constantes locales de même niveau.

Remarque 9. Si l'on souhaite ajouter des commentaires à un programme, on utilisera la syntaxe `(* .... *)`

Instructions	Réponse	Commentaires
(* je déclare une constante globale A *) let A = 2 ;;		

Remarque 10. Les commentaires sont indispensables pour garantir la bonne lisibilité d'un programme.