

Petite librairie de Caml Light

BATOG Guillaume

30 août 2006

Entiers et flottants

Arithmétique

```
value prefix / : int -> int -> int
value prefix mod : int -> int -> int
  (* Résultats imprévisibles si un des arguments est négatif *)
value abs : int -> int
value max_int : int
value min_int : int
```

Comparaisons

```
value prefix = : 'a -> 'a -> bool
value prefix <> : 'a -> 'a -> bool
value prefix < : 'a -> 'a -> bool
value prefix <= : 'a -> 'a -> bool
value prefix > : 'a -> 'a -> bool
value prefix >= : 'a -> 'a -> bool
value compare: 'a -> 'a -> int
  (* compare x y retourne 0 si x=y, un entier négatif si x<y
    et un entier positif si x>y *)
value min: 'a -> 'a -> 'a
value max: 'a -> 'a -> 'a
value prefix == : 'a -> 'a -> bool
value prefix != : 'a -> 'a -> bool
```

`e1 = e2` teste l'égalité structurelle entre `e1` et `e2`. Des structures mutables (références et tableaux) sont égales si et seulement si leurs contenus sont structurellement égaux, même si les deux objets mutables ne représentent pas le même objet physique. L'égalité entre deux fonctions retourne une exception. L'égalité entre deux structures de données cycliques peut ne pas terminer.

`e1 == e2` teste l'égalité physique entre `e1` et `e2`. Pour les entiers et les caractères, le test est identique à celui de l'égalité structurelle. Pour les structures mutables, `e1 == e2` si et seulement si une modification de `e1` affecte `e2`. Pour les structures non mutables, le comportement de `==` est dépendant de l'implémentation. On a toujours `e1 == e2` qui implique `e1 = e2`.

Opérations sur les flottants

```
value prefix ** : float -> float -> float
value acos : float -> float
value atan : float -> float
value cos : float -> float
value exp : float -> float
value log10 : float -> float
value sinh : float -> float
value tan : float -> float
value ceil : float -> float
value abs_float : float -> float
value mod_float : float -> float -> float
  (* fmod a b retourne le reste de la division euclidienne de a par b. *)
value modf : float -> float * float
  (* modf f retourne les parties décimale et entière de f. *)
value power : float -> float -> float
value asin : float -> float
value atan2 : float -> float -> float
value cosh : float -> float
value log : float -> float
value sin : float -> float
value sqrt : float -> float
value tanh : float -> float
value floor : float -> float
```

Listes et tableaux

```
value list_length : 'a list -> int
value vect_length : 'a vect -> int
value prefix @ : 'a list -> 'a list -> 'a list
value concat_vect : 'a vect -> 'a vect -> 'a vect
value list_of_vect : 'a vect -> 'a list
value vect_of_list : 'a list -> 'a vect
```

Itérateurs

```
value map : ('a -> 'b) -> 'a list -> 'b list
  (* map f [a1; ...; an] construit la liste [f a1; ...; f an] *)
value do_list : ('a -> unit) -> 'a list -> unit
  (* do_list f [a1; ...; an] est équivalent au programme
     f a1; f a2; ...; f an; ();; *)

value map_vect : ('a -> 'b) -> 'a vect -> 'b vect
  (* map_vect f v construit le tableau
     [| f v.(0); f v.(1); ...; f v.(vect_length v - 1) |] *)
value map_vect_list : ('a -> 'b) -> 'a vect -> 'b list
  (* map_vect_list f v construit la liste
     [ f v.(0); f v.(1); ...; f v.(vect_length v - 1) ] *)
value do_vect : ('a -> unit) -> 'a vect -> unit
  (* do_vect f v est équivalent au programme
     f v.(0); f v.(1); ...; f v.(vect_length v - 1); ();; *)

value it_list : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a
  (* it_list f a [b1; ...; bn] retourne f (... (f (f a b1) b2) ...) bn *)
value list_it : ('a -> 'b -> 'b) -> 'a list -> 'b -> 'b
  (* list_it f [a1; ...; an] b retourne f a1 (f a2 (... (f an b) ...)) *)

value flat_map : ('a -> 'b list) -> 'a list -> 'b list
  (* flat_map f [l1; ...; ln] retourne (f l1) @ (f l2) @ ... @ (f ln) *)
```

Parcours de liste

```
value for_all : ('a -> bool) -> 'a list -> bool
  (* for_all p [a1; ...; an] retourne (p a1) & (p a2) & ... & (p an) *)
value exists : ('a -> bool) -> 'a list -> bool
  (* exists p [a1; ...; an] retourne (p a1) or (p a2) or ... or (p an) *)
value mem : 'a -> 'a list -> bool
  (* mem a l retourne true si et seulement si
     a est structurellement égal à un élément de l *)
value memq : 'a -> 'a list -> bool
  (* memq a l retourne true si et seulement si
     a est physiquement égal à un élément de l *)
```

Création de tableaux

```
value vect_item : 'a vect -> int -> 'a (* v.(n) *)
value vect_assign : 'a vect -> int -> 'a -> unit (* v.(n) <- x *)
value make_vect : int -> 'a -> 'a vect
  (* make_vect n x retourne un vecteur frais de longueur n
     dont tous les éléments sont physiquement égaux à x *)
value make_matrix : int -> int -> 'a -> 'a vect vect
  (* make_matrix dimx dimy e retourne une matrice fraîche de dimensions dimx
     et dimy dont tous les éléments sont physiquement égaux à x; on accède à
     l'élément (x,y) de m par m.(x).(y) *)
value init_vect : int -> (int -> 'a) -> 'a vect
  (* init_vect n f retourne un tableau frais de longueur n
     dont l'élément de position i fait f i *)
value sub_vect : 'a vect -> int -> int -> 'a vect
```

```

    (* sub_vect v i j retourne un tableau frais de longueur j
       contenant les éléments d'indice i à i+j-1 du tableau v *)
value copy_vect : 'a vect -> 'a vect
    (* copy_vect v retourne une copie fraîche de v *)

```

Mise à jour de tableaux

```

value fill_vect : 'a vect -> int -> int -> 'a -> unit
    (* fill_vect v i j x remplace par x dans v les éléments d'indice i à i+j-1 *)
value blit_vect : 'a vect -> int -> 'a vect -> int -> int -> unit
    (* blit_vect v1 o1 v2 o2 len copie len éléments du tableau v1 à partir de
       l'indice o1 et les remplace dans le tableau v2 à partir de l'indice o2.
       La fonction est bien définie même lorsque v1 et v2 sont les mêmes
       tableaux et les intervalles de départ et d'arrivée se superposent. *)

```

Caractères et chaînes

```

value string_of_int : int -> string
value int_of_string : string -> int
value string_of_float : float -> string
value float_of_string : string -> float
value string_of_bool : bool -> string

```

```

value int_of_char : char -> int
value char_of_int : int -> char
value string_of_char : char -> string

```

Création et mise à jour de chaîne

```

value string_length : string -> int
value nth_char : string -> int -> char    (* s.[n] *)
value set_nth_char : string -> int -> char -> unit (* s.[n] <- c *)
value prefix ^ : string -> string -> string
value concat : string list -> string
value sub_string : string -> int -> int -> string
    (* Correspond à sub_vect pour les tableaux *)
value create_string : int -> string
    (* create_string n retourne une chaîne de longueur n
       contenant des caractères arbitraires *)
value make_string : int -> char -> string
    (* make_string n c retourne une chaîne de longueur n
       donc tous les caractères sont égaux à c *)
value fill_string : string -> int -> int -> char -> unit
value blit_string : string -> int -> string -> int -> int -> unit
    (* Correspondent à fill_vect et blit_vect des tableaux *)
value replace_string : string -> string -> int -> unit
    (* replace_string dest src start copie tous les caractères de src
       dans dest à partir du caractère d'indice start dans dst *)

```

Entrées et sorties

```

type in_channel
type out_channel
exception End_of_file
value stdin : in_channel
value stdout : out_channel
value exit : int -> 'a

```

Fonctions de sortie sur la sortie standard

```

value print_char : char -> unit

```

```

value print_string : string -> unit
value print_int : int -> unit
value print_float : float -> unit
value print_endline : string -> unit
value print_newline : unit -> unit

```

Fonctions d'entrée sur l'entrée standard

```

value read_line : unit -> string
value read_int : unit -> int
value read_float : unit -> float

```

Fonctions générales de sortie

```

value open_out : string -> out_channel
  (* open_out crée le fichier indiqué ou l'écrase s'il existe *)
value flush : out_channel -> unit
  (* exécute toutes les opérations en attente sur le canal *)
value output_char : out_channel -> char -> unit
value output_string : out_channel -> string -> unit
value output : out_channel -> string -> int -> int -> unit
  (* output oc s p l écrit sur le canal de sortie oc la sous-chaîne de s
    de longueur l et commençant au caractère d'indice p *)
value close_out : out_channel -> unit

```

Fonctions générales d'entrée

```

value open_in : string -> in_channel
  (* open_in ouvre le fichier indiqué s'il existe *)
value input_char : in_channel -> char
value input_line : in_channel -> string
value input : in_channel -> string -> int -> int -> int
  (* input ic s p l tente de lire l caractères sur un canal d'entrée ic,
    et les stocke dans la chaîne s à partir de la position p;
    le nombre de caractères effectivement lus est retourné. *)
value close_in : in_channel -> unit

```

Module printf

```

# #open "printf";;
# printf "nom = %s, age = %d, \n note = %f" "hubert" 32 18.32;;
nom = hubert, age = 32,
  note = 18.32- : unit = ()
# let s = sprintf "%10d\n%.6f\n" (-14034) (2.3000000003);;
s : string = "      -14034\n2.300000\n"
# print_string s;;
-14034
2.300000
- : unit = ()

```