Informatique MP
Informations relatives à la bibliothèque CaML big_int

Antoine MOTEAU
antoine.moteau@wanadoo.fr

Dernière rédaction : 27 Juillet 2003

.../Bigint.tex (2003)
.../Bigint.tex Compilé le dimanche 11 mars 2018 à 09h 49m 47s avec LaTeX.
Compilable avec LaTeX, PDFLaTeX, LuaLaTeX, XeLaTeX.

Informations :
Module `big_int` de CaML

Le type `int` de `CaML` ne concerne que les "petits" nombres entiers signés compris entre `min_int` et `max_int`

- `min_int` et `max_int` ont des valeurs variables selon la version de `CaML` et l'environnement
- `min_int` = -1073741824 et `max_int` = +1073741823, dans le cas d'une représentation à 31 bits,
  et, pour mémoire, on rappelle que dans `CaML` :

$$\begin{cases} \text{max\_int} + \text{max\_int} & \text{vaut -2} \\ \text{max\_int} \times \text{max\_int} & \text{vaut 1} \\ \text{max\_int} + 1 & \text{vaut -1073741824} = \text{min\_int} \qquad \ldots\text{etc}\ldots \end{cases}$$

Le module "`big_int`" introduit des entiers ayant un nombre (presque) arbitraire de chiffres, en fournissant

- un type `big_int` , abstrait et non documenté,
- les ressources usuelles de calcul relatives à ce type sous forme de fonctions (non documentées).
  ("auto-documentées" par le nom et le type, ce qui n'est pas toujours suffisant),
- sans fichier d'aide ou de documentation associé (du moins je n'en ai pas trouvé . . .).

1 Exemple élémentaire d'utilisation du module "`big_int`"

```
#open "big_int";;                                       (* ouverture du module *)
let a = big_int_of_string "12345678914567815394258004652115925623 56"
```

2 Module "`big_int`" (Interface BIG_INT.mli complet, réarrangé)

```
(* Operation on big integers *)
(* Big integers (type [big_int]) are signed integers of arbitrary size. *)

type big_int;;

value sign_big_int : big_int -> int                   (* +1 ou -1 *)
  and zero_big_int : big_int                          (* 0 en tant que big_int *)
  and unit_big_int : big_int                          (* 1 en tant que big_int *)

  and minus_big_int : big_int -> big_int              (* moins "unaire" = opposé *)
  and abs_big_int : big_int -> big_int

  and compare_big_int : big_int -> big_int -> int     (* -1, 0, +1  *)
  and eq_big_int : big_int -> big_int -> bool
  and le_big_int : big_int -> big_int -> bool
  and ge_big_int : big_int -> big_int -> bool
  and lt_big_int : big_int -> big_int -> bool
  and gt_big_int : big_int -> big_int -> bool

  and max_big_int : big_int -> big_int -> big_int     (* max et mini de deux big_int *)
  and min_big_int : big_int -> big_int -> big_int

  and pred_big_int : big_int -> big_int               (* x -> x - 1, économique *)
  and succ_big_int : big_int -> big_int               (* x -> x + 1, économique *)
```

```
  and is_int_big_int : big_int -> bool                 (* conversions *)
  and big_int_of_int : int -> big_int
  and int_of_big_int : big_int -> int
  and string_of_big_int : big_int -> string
  and big_int_of_string : string -> big_int
  and float_of_big_int : big_int -> float
  and big_int_of_float : float -> big_int

  and square_big_int: big_int -> big_int
  and sqrt_big_int: big_int -> big_int
  and add_big_int : big_int -> big_int -> big_int       (* opérations usuelles *)
  and add_int_big_int : int -> big_int -> big_int               (* économie si ... *)
  and sub_big_int : big_int -> big_int -> big_int
  and mult_big_int : big_int -> big_int -> big_int
  and mult_int_big_int : int -> big_int -> big_int              (* économie si ... *)

  and quomod_big_int : big_int -> big_int -> big_int * big_int
  and div_big_int : big_int -> big_int -> big_int
  and mod_big_int : big_int -> big_int -> big_int
  and gcd_big_int : big_int -> big_int -> big_int

  and base_power_big_int: int -> int -> big_int -> big_int                    (* ?? *)

  and power_int_positive_int: int -> int -> big_int           (* a^x en plusieurs cas *)
  and power_big_int_positive_int: big_int -> int -> big_int
  and power_int_positive_big_int: int -> big_int -> big_int
  and power_big_int_positive_big_int: big_int -> big_int -> big_int

  and approx_big_int: int -> big_int -> string                                (* ?? *)
  and sys_big_int_of_string : int -> string -> int -> int -> big_int          (* ?? *)
  and sys_string_of_big_int : int -> string -> big_int -> string -> string    (* ?? *)
  and simple_big_int_of_string : int -> string -> int -> int -> big_int       (* ?? *)

  and num_digits_big_int : big_int -> int
  and leading_digit_big_int : big_int -> int
  and nth_digit_big_int : big_int -> int -> int
  and round_futur_last_digit : string -> int -> int -> bool                   (* ?? *)
;;
value sys_print_big_int : int -> string -> big_int -> string -> unit;;        (* ?? *)
value print_big_int : big_int -> unit;;

value create_big_int : int -> nat__nat -> big_int
  and abs_value_big_int : big_int -> nat__nat
  and nat_of_big_int : big_int -> nat__nat
  and big_int_of_nat : nat__nat -> big_int
```

3 Compléments : autres modules sur les grands nombres

3.1 Module nat (extraits de l'interface NAT.mli)

```
(* Operation on natural numbers *)

(* Natural numbers (type [nat]) are positive integers of arbitrary size.
   All operations on [nat] are performed in-place. *)

type nat == fnat__nat;;

value ...
```

3.2 Module fnat (extraits de l'interface FNAT.mli)

```
(* Operation on natural numbers *)

type nat;;
(* Natural numbers (type [nat]) are positive integers of arbitrary size.
   All operations on [nat] are performed in-place. *)

value ...
```

3.3 Module num (extraits de l'interface NUM.mli)

```
(* Operations on numbers *)

#open "ref";;
#open "big_int";;
#open "ratio";;

(* Numbers (type [num]) are arbitrary-precision rational numbers,
   plus the special elements [1/0] (infinity) and [0/0] (undefined). *)

type num = Int of int | Big_int of big_int | Ratio of ratio;;     (* The type of numbers. *)

value  ...
```

3.4 Module ref (interface REF.mli, complet)

```
(* Operations on references *)

type 'a ref = ref of mutable 'a;;
   (* The type of references (mutable indirection cells) containing a value of type ['a]. *)

value prefix ! : 'a ref -> 'a = 1 "field0"
        (* [!r] returns the current contents of reference [r].
           Could be defined as [fun (ref x) -> x]. *)
  and prefix := : 'a ref -> 'a -> unit = 2 "setfield0"
        (* [r := a] stores the value of [a] in reference [r]. *)
  and incr : int ref -> unit = 1 "incr"
        (* Increment the integer contained in the given reference.
           Could be defined as [fun r -> r := succ !r]. *)
  and decr : int ref -> unit = 1 "decr"
        (* Decrement the integer contained in the given reference.
           Could be defined as [fun r -> r := pred !r]. *)
;;
```

3.5 Module ratio (extraits de l'interface RATIO.mli)

```
(* Operation on rationals *)

#open "ref";;
#open "big_int";;

(* Rationals (type [ratio]) are arbitrary-precision rational numbers,
   plus the special elements [1/0] (infinity) and [0/0] (undefined).
   In constrast with numbers (type [num]), the special cases of
   small integers and big integers are not optimized specially. *)

type ratio;;

value ...
```

4 Informations non documentées sur le type big_int

4.1 Interprétation positive des entiers signés (int)

Extrait de l'interface INT.mli du module int de Caml :

```
(* Integers are 31 bits wide (or 63 bits on 64-bit processors).
   All operations are taken modulo $2^{31}$ (or $2^{63}$).
   They do not fail on overflow. *)
```

On supposera que l'on est dans le cas d'une représentation utilisant 31 bits.

Avec 31 bits on peut représenter les entiers positifs de l'intervalle $[0, 2^{31} = 2147483648]$ et, pour représenter des entiers signés, on **réserve un bit pour le signe**, ce qui limite l'amplitude.
Les entiers signés (int) prennent leurs valeurs dans l'intervalle $[\texttt{min\_int}, \texttt{max\_int}]$ où

$$\begin{cases} \texttt{min\_int} = -2^{30} = -1073741824 \\ \texttt{max\_int} = 2^{30} - 1 = +1073741823. \end{cases}$$

Les entiers signés (`int`), de l'intervalle $[min\_int, max\_int]$, sont interprétés comme des entiers naturels (non signés) de l'intervalle $[0, 2 \times max\_int + 1]$:

- si $n \in [0, max\_int]$, à n est associée la valeur n
- si $n \in [min\_int, 0[$, à n est associée la valeur $-min\_int - 1 - n = max\_int - n$

Par exemple,

$$\begin{cases} \text{à } 21 & \text{est associée } 21 \\ \text{à } -212356 & \text{est associée } -min\_int - 1 + 212356 = max\_int + 212356 = 1073529467 \\ \text{à } min\_int & \text{est associée } -min\_int - 1 - min\_int = max\_int - min\_int = 2 \times max\_int + 1 = 2147483647 \\ \text{à } -1 & \text{est associée } -min\_int - 1 + 1 = max\_int + 1 = 1073741824 = 2^{30} \end{cases}$$

4.2 Description du type big_int

Bien que le type big_int soit non documenté, il semble que les big_int soient des nombres entiers représentés, dans la base $m = 2\times$ `max_int` $+1$, sous la forme :

$$N = s\,,\, n_0 + n_1 \times m + n_2 \times m^2 + \cdots + n_k \times m^k \text{ où } \begin{cases} s \text{ est le signe} \\ \text{les digits } n_i \text{ sont des "entiers" de } [\![0 \cdots 2\times \texttt{max\_int+1}]\!] \\ k \text{ est un entier (vrai int)}, \quad 0 \leqslant k \leqslant \texttt{max\_int} \end{cases}$$

le type étant implanté comme une "collection" (informelle) de s et des "digits" : $s\,,\, n_0, n_1, n_2, \cdots, n_k$.

On peut alors représenter (du moins en théorie) des `big_int` dont la valeur absolue peut aller jusqu'à

$$(2 \times \texttt{max\_int} + 1)^{\texttt{max\_int}} - 1 \qquad \text{soit} \quad 2147483647^{1073741823} - 1 \quad \text{dans une représentation à 31 bits}$$

Remarque : `max_int` et `min_int` ont des valeurs qui dépendent de l'environnement, et tout programme qui utilise explicitement leurs valeurs numérique est ... "FAUX" !

Resources d'accès à la représentation de $N = s\,,\, n_0 + n_1 \times m + n_2 \times m^2 + \cdots + n_k \times m^k$:

- `(nth_digit_big_int N i)` est le i-ième "digit" de N, soit n_i sous forme d'entier signé !
- `(num_digits_big_int N)` est le nombre de digits de N, soit k
- `(leading_digit_big_int N)` est le digit de poids fort de N, soit n_k sous forme d'entier signé !
- `(round_futur_last_digit s i j)` ????

Si `(nth_digit_big_int N i)` renvoie -173, cela signifie la valeur positive $max\_int + 173 = 1073741996$ (du moins dans un environnement où la représentation des `int` utilise 31 bits).

$< \mathscr{FIN} >$