MLGmpIDL: OCaml interface for GMP library (version 1.1)

April 5, 2012

All files distributed in the MLGmpIDL interface are distributed under LGPL license.
Copyright (C) Bertrand Jeannet 2005-2009 for the MLGmpIDL interface.

Introduction

This package is an OCaml interface for the GMP interface, which is decomposed into 7 submodules, corresponding to C modules:

Mpz	:	GMP integers, with side-effect semantics (as in C library)
Mpq	:	GMP rationals, with side-effect semantics (as in C library)
Mpzf	:	GMP integers, with functional semantics
Mpqf	:	GMP rationals, with functional semantics
Mpf	:	GMP multiprecision floating-point numbers
Gmp_random	:	GMP random number functions
Mpfr	:	MPFR multiprecision floating-point numbers, with side-effect semantics (as in C library)
Mpfr	:	MPFR multiprecision floating-point numbers, with functional semantics

There already exist such an interface, mlgmp, written by D. Monniaux and available at `http://www.di.ens.fr/~monniaux/p`
The motivation for writing a new one in the APRON project were

1. The fact that mlgmp provides by default a functional interface to GMP, potentially more costly in term of memory allocation than an imperative interface. mlgmp provides only a relative small numbers of functions in an imperative version.
2. The compatibility with the CamlIDL tool. MLGmpIDL uses CamlIDL, so that other OCaml/C interfaces written with CamlIDL may reuse the MLGmpIDL `.idl` files.

Requirements

- GMP library (tested with version 4.0 and up)
- MPFR library (optional, tested with version 2.2.x)
- OCaml 3.0 or up (tested with 3.09 and 3.10)
- Camlidl (tested with 1.05)

Installation

Library Set the file Makefile.config using the Makefile.config model to your own setting. You might also have to modify the Makefile for executables

If you download from the subversion repository, type 'make rebuild', which builds .ml, .mli, and _caml.c files from .idl files.

type 'make', possibly 'make debug', and then 'make install'

The OCaml part of the library is named gmp.cma (.cmxa, .a) The C part of the library is named libgmp_caml.a (libgmp_caml_debug.a)

'make install' installs not only .mli, .cmi, but also .idl files.

Be aware however that importing those .idl files from other .idl files will probably request the application of SED editor with the scripts sedscript_caml and sedscript_c (look at the Makefile).

Interpreter and toplevel You may also generate runtime and toplevel with 'make gmprun', 'make gmptop'

Documentation The documentation (currently very sketchy) is generated with ocamldoc.

'make mlapronidl.dvi'

'make html' (put the HTML files in the html subdirectoy)

Miscellaneous 'make clean' and 'make distclean' have the usual behaviour.

'make mostlyclean', in addition to 'make clean', removes the .ml, .mli and _caml.c files generated from .idl files.

Contents

Chapter 1

Module Mpz : GMP multi-precision integers

`type t`

GMP multi-precision integers

The following operations are mapped as much as possible to their C counterpart. In case of imperative functions (like `set`, `add`, ...) the first parameter of type `t` is an out-parameter and holds the result when the function returns. For instance, `add x y z` adds the values of `y` and `z` and stores the result in `x`.

These functions are as efficient as their C counterpart: they do not imply additional memory allocation, unlike the corresponding functions in the module `Mpzf`[2].

1.1 Pretty printing

```
val print : Format.formatter -> t -> unit
```

1.2 Initialization Functions

C documentation[`http://gmplib.org/manual/Initializing-Integers.html#Initializing-Integers`]

```
val init : unit -> t
val init2 : int -> t
val realloc2 : t -> int -> unit
```

1.3 Assignement Functions

C documentation[`http://gmplib.org/manual/Assigning-Integers.html#Assigning-Integers`]

The first parameter holds the result.

```
val set : t -> t -> unit
val set_si : t -> int -> unit
val set_d : t -> float -> unit
```

For `set_q: t -> Mpq.t -> unit`, see `Mpq.get_z`[3.4]

```
val _set_str : t -> string -> int -> unit
val set_str : t -> string -> base:int -> unit
val swap : t -> t -> unit
```

1.4 Combined Initialization and Assignment Functions

C documentation[http://gmplib.org/manual/Simultaneous-Integer-Init-_0026-Assign.html#Simultaneous-Intege

```
val init_set : t -> t
val init_set_si : int -> t
val init_set_d : float -> t
val _init_set_str : string -> int -> t
val init_set_str : string -> base:int -> t
```

1.5 Conversion Functions

C documentation[http://gmplib.org/manual/Converting-Integers.html#Converting-Integers]

```
val get_si : t -> nativeint
val get_int : t -> int
val get_d : t -> float
val get_d_2exp : t -> float * int
val _get_str : int -> t -> string
val get_str : base:int -> t -> string
```

1.6 User Conversions

These functions are additions to or renaming of functions offered by the C library.

```
val to_string : t -> string
val to_float : t -> float
val of_string : string -> t
val of_float : float -> t
val of_int : int -> t
```

1.7 Arithmetic Functions

C documentation[http://gmplib.org/manual/Integer-Arithmetic.html#Integer-Arithmetic]
The first parameter holds the result.

```
val add : t -> t -> t -> unit
val add_ui : t -> t -> int -> unit
val sub : t -> t -> t -> unit
val sub_ui : t -> t -> int -> unit
val ui_sub : t -> int -> t -> unit
val mul : t -> t -> t -> unit
val mul_si : t -> t -> int -> unit
val addmul : t -> t -> t -> unit
val addmul_ui : t -> t -> int -> unit
val submul : t -> t -> t -> unit
val submul_ui : t -> t -> int -> unit
val mul_2exp : t -> t -> int -> unit
val neg : t -> t -> unit
val abs : t -> t -> unit
```

1.8 Division Functions

C documentation[http://gmplib.org/manual/Integer-Division.html#Integer-Division]
c stands for ceiling, f for floor, and t for truncate (rounds toward 0).

1.8.1 Ceiling division

```
val cdiv_q : t -> t -> t -> unit
```
The first parameter holds the quotient.

```
val cdiv_r : t -> t -> t -> unit
```
The first parameter holds the remainder.

```
val cdiv_qr : t -> t -> t -> t -> unit
```
The two first parameters hold resp. the quotient and the remainder).

```
val cdiv_q_ui : t -> t -> int -> int
```
The first parameter holds the quotient.

```
val cdiv_r_ui : t -> t -> int -> int
```
The first parameter holds the remainder.

```
val cdiv_qr_ui : t -> t -> t -> int -> int
```
The two first parameters hold resp. the quotient and the remainder).

```
val cdiv_ui : t -> int -> int
val cdiv_q_2exp : t -> t -> int -> unit
```
The first parameter holds the quotient.

```
val cdiv_r_2exp : t -> t -> int -> unit
```
The first parameter holds the remainder.

1.8.2 Floor division

```
val fdiv_q : t -> t -> t -> unit
val fdiv_r : t -> t -> t -> unit
val fdiv_qr : t -> t -> t -> t -> unit
val fdiv_q_ui : t -> t -> int -> int
val fdiv_r_ui : t -> t -> int -> int
val fdiv_qr_ui : t -> t -> t -> int -> int
val fdiv_ui : t -> int -> int
val fdiv_q_2exp : t -> t -> int -> unit
val fdiv_r_2exp : t -> t -> int -> unit
```

1.8.3 Truncate division

```
val tdiv_q : t -> t -> t -> unit
val tdiv_r : t -> t -> t -> unit
val tdiv_qr : t -> t -> t -> t -> unit
val tdiv_q_ui : t -> t -> int -> int
val tdiv_r_ui : t -> t -> int -> int
val tdiv_qr_ui : t -> t -> t -> int -> int
val tdiv_ui : t -> int -> int
val tdiv_q_2exp : t -> t -> int -> unit
val tdiv_r_2exp : t -> t -> int -> unit
```

1.8.4 Other division-related functions

```
val gmod : t -> t -> t -> unit
val gmod_ui : t -> t -> int -> int
val divexact : t -> t -> t -> unit
val divexact_ui : t -> t -> int -> unit
val divisible_p : t -> t -> bool
val divisible_ui_p : t -> int -> bool
val divisible_2exp_p : t -> int -> bool
val congruent_p : t -> t -> t -> bool
val congruent_ui_p : t -> int -> int -> bool
val congruent_2exp_p : t -> t -> int -> bool
```

1.9 Exponentiation Functions

C documentation[http://gmplib.org/manual/Integer-Exponentiation.html#Integer-Exponentiation]

```
val _powm : t -> t -> t -> t -> unit
val _powm_ui : t -> t -> int -> t -> unit
val powm : t -> t -> t -> modulo:t -> unit
val powm_ui : t -> t -> int -> modulo:t -> unit
val pow_ui : t -> t -> int -> unit
val ui_pow_ui : t -> int -> int -> unit
```

1.10 Root Extraction Functions

C documentation[http://gmplib.org/manual/Integer-Roots.html#Integer-Roots]

```
val root : t -> t -> int -> bool
val sqrt : t -> t -> unit
val _sqrtrem : t -> t -> t -> unit
val sqrtrem : t -> remainder:t -> t -> unit
val perfect_power_p : t -> bool
val perfect_square_p : t -> bool
```

1.11 Number Theoretic Functions

C documentation[http://gmplib.org/manual/Number-Theoretic-Functions.html#Number-Theoretic-Functions]

```
val probab_prime_p : t -> int -> int
val nextprime : t -> t -> unit
val gcd : t -> t -> t -> unit
val gcd_ui : t option -> t -> int -> int
val _gcdext : t -> t -> t -> t -> t -> unit
val gcdext : gcd:t -> alpha:t -> beta:t -> t -> t -> unit
val lcm : t -> t -> t -> unit
val lcm_ui : t -> t -> int -> unit
val invert : t -> t -> t -> bool
val jacobi : t -> t -> int
val legendre : t -> t -> int
val kronecker : t -> t -> int
val kronecker_si : t -> int -> int
val si_kronecker : int -> t -> int
val remove : t -> t -> t -> int
val fac_ui : t -> int -> unit
val bin_ui : t -> t -> int -> unit
val bin_uiui : t -> int -> int -> unit
val fib_ui : t -> int -> unit
val fib2_ui : t -> t -> int -> unit
val lucnum_ui : t -> int -> unit
val lucnum2_ui : t -> t -> int -> unit
```

1.12 Comparison Functions

C documentation[http://gmplib.org/manual/Integer-Comparisons.html#Integer-Comparisons]

```
val cmp : t -> t -> int
val cmp_d : t -> float -> int
val cmp_si : t -> int -> int
val cmpabs : t -> t -> int
val cmpabs_d : t -> float -> int
val cmpabs_ui : t -> int -> int
val sgn : t -> int
```

1.13 Logical and Bit Manipulation Functions

C documentation[http://gmplib.org/manual/Integer-Logic-and-Bit-Fiddling.html#Integer-Logic-and-Bit-Fiddl

```
val gand : t -> t -> t -> unit
val ior : t -> t -> t -> unit
val xor : t -> t -> t -> unit
val com : t -> t -> unit
val popcount : t -> int
val hamdist : t -> t -> int
```

```
val scan0 : t -> int -> int
val scan1 : t -> int -> int
val setbit : t -> int -> unit
val clrbit : t -> int -> unit
val tstbit : t -> int -> bool
```

1.14 Input and Output Functions: not interfaced

1.15 Random Number Functions: see Gmp_random[8] module

1.16 Integer Import and Export Functions

C documentation[ttp://gmplib.org/manual/Integer-Import-and-Export.html#Integer-Import-and-Export]

```
val _import :
  t ->
  (int, Bigarray.int32_elt, Bigarray.c_layout) Bigarray.Array1.t ->
  int -> int -> unit
val _export :
  t ->
  int -> int -> (int, Bigarray.int32_elt, Bigarray.c_layout) Bigarray.Array1.t
val import :
  dest:t ->
  (int, Bigarray.int32_elt, Bigarray.c_layout) Bigarray.Array1.t ->
  order:int -> endian:int -> unit
val export :
  t ->
  order:int ->
  endian:int -> (int, Bigarray.int32_elt, Bigarray.c_layout) Bigarray.Array1.t
```

1.17 Miscellaneous Functions

C documentation[http://gmplib.org/manual/Miscellaneous-Integer-Functions.html#Miscellaneous-Integer-Funct

```
val fits_int_p : t -> bool
val odd_p : t -> bool
val even_p : t -> bool
val size : t -> int
val sizeinbase : t -> int -> int
val fits_ulong_p : t -> bool
val fits_slong_p : t -> bool
val fits_uint_p : t -> bool
val fits_sint_p : t -> bool
val fits_ushort_p : t -> bool
val fits_sshort_p : t -> bool
```

Chapter 2

Module Mpzf : GMP multi-precision integers, functional version

Functions in this module has a functional semantics, unlike the corresponding functions in Mpz[1]. These functions are less efficients, due to the additional memory allocation neded for the result.
This module could be extended to offer more functions with a functional semantics.

```
type t
```

multi-precision integer

```
val to_mpz : t -> Mpz.t
val of_mpz : Mpz.t -> t
```

Safe conversion from and to Mpz.t.

There is no sharing between the argument and the result.

```
val mpz : t -> Mpz.t
val mpzf : Mpz.t -> t
```

Unsafe conversion from and to Mpz.t.

The argument and the result actually share the same number: be cautious !

2.1 Pretty-printing

```
val print : Format.formatter -> t -> unit
```

2.2 Constructors

```
val of_string : string -> t
val of_float : float -> t
val of_int : int -> t
```

2.3 Conversions

```
val to_string : t -> string
val to_float : t -> float
```

2.4 Arithmetic Functions

```
val add : t -> t -> t
val add_int : t -> int -> t
val sub : t -> t -> t
val sub_int : t -> int -> t
val mul : t -> t -> t
val mul_int : t -> int -> t
val cdiv_q : t -> t -> t
val cdiv_r : t -> t -> t
val cdiv_qr : t -> t -> t * t
val fdiv_q : t -> t -> t
val fdiv_r : t -> t -> t
val fdiv_qr : t -> t -> t * t
val tdiv_q : t -> t -> t
val tdiv_r : t -> t -> t
val tdiv_qr : t -> t -> t * t
val divexact : t -> t -> t
val gmod : t -> t -> t
val gcd : t -> t -> t
val lcm : t -> t -> t
val neg : t -> t
val abs : t -> t
```

2.5 Comparison Functions

```
val cmp : t -> t -> int
val cmp_int : t -> int -> int
val sgn : t -> int
```

Chapter 3

Module Mpq : GMP multiprecision rationals

type t

GMP multiprecision rationals

The following operations are mapped as much as possible to their C counterpart. In case of imperative functions (like set, add, ...) the first parameter of type t is an out-parameter and holds the result when the function returns. For instance, add x y z adds the values of y and z and stores the result in x.

These functions are as efficient as their C counterpart: they do not imply additional memory allocation, unlike the corresponding functions in the module Mpqf[4].

```
val canonicalize : t -> unit
```

3.1 Pretty printing

```
val print : Format.formatter -> t -> unit
```

3.2 Initialization and Assignment Functions

C documentation[http://gmplib.org/manual/Initializing-Rationals.html#Initializing-Rationals]

```
val init : unit -> t
val set : t -> t -> unit
val set_z : t -> Mpz.t -> unit
val set_si : t -> int -> int -> unit
val _set_str : t -> string -> int -> unit
val set_str : t -> string -> base:int -> unit
val swap : t -> t -> unit
```

3.3 Additional Initialization and Assignements functions

These functions are additions to or renaming of functions offered by the C library.

```
val init_set : t -> t
val init_set_z : Mpz.t -> t
val init_set_si : int -> int -> t
val init_set_str : string -> base:int -> t
```

```
val init_set_d : float -> t
```

3.4 Conversion Functions

C documentation[http://gmplib.org/manual/Rational-Conversions.html#Rational-Conversions]

```
val get_d : t -> float
val set_d : t -> float -> unit
val get_z : Mpz.t -> t -> unit
val _get_str : int -> t -> string
val get_str : base:int -> t -> string
```

3.5 User Conversions

These functionss are additions to or renaming of functions offeered by the C library.

```
val to_string : t -> string
val to_float : t -> float
val of_string : string -> t
val of_float : float -> t
val of_int : int -> t
val of_frac : int -> int -> t
val of_mpz : Mpz.t -> t
val of_mpz2 : Mpz.t -> Mpz.t -> t
```

3.6 Arithmetic Functions

C documentation[http://gmplib.org/manual/Rational-Arithmetic.html#Rational-Arithmetic]

```
val add : t -> t -> t -> unit
val sub : t -> t -> t -> unit
val mul : t -> t -> t -> unit
val mul_2exp : t -> t -> int -> unit
val div : t -> t -> t -> unit
val div_2exp : t -> t -> int -> unit
val neg : t -> t -> unit
val abs : t -> t -> unit
val inv : t -> t -> unit
```

3.7 Comparison Functions

C documentation[http://gmplib.org/manual/Comparing-Rationals.html#Comparing-Rationals]

```
val cmp : t -> t -> int
val cmp_si : t -> int -> int -> int
val sgn : t -> int
val equal : t -> t -> bool
```

3.8 Applying Integer Functions to Rationals

C documentation[http://gmplib.org/manual/Applying-Integer-Functions.html#Applying-Integer-Functions]

```
val get_num : Mpz.t -> t -> unit
val get_den : Mpz.t -> t -> unit
val set_num : t -> Mpz.t -> unit
val set_den : t -> Mpz.t -> unit
```

3.9 Input and Output Functions: not interfaced

Chapter 4

Module Mpqf : GMP multi-precision rationals, functional version

Functions in this module has a functional semantics, unlike the corresponding functions in Mpq[3]. These functions are less efficients, due to the additional memory allocation neded for the result.

```
type t
```

multi-precision rationals

```
val to_mpq : t -> Mpq.t
val of_mpq : Mpq.t -> t
```

Safe conversion from and to Mpq.t.

There is no sharing between the argument and the result.

```
val mpq : t -> Mpq.t
val mpqf : Mpq.t -> t
```

Unsafe conversion from and to Mpq.t.

The argument and the result actually share the same number: be cautious !

4.1 Pretty-printing

```
val print : Format.formatter -> t -> unit
```

4.2 Constructors

```
val of_string : string -> t
val of_float : float -> t
val of_int : int -> t
val of_frac : int -> int -> t
val of_mpz : Mpz.t -> t
val of_mpz2 : Mpz.t -> Mpz.t -> t
val of_mpzf : Mpzf.t -> t
val of_mpzf2 : Mpzf.t -> Mpzf.t -> t
```

4.3 Conversions

```
val to_string : t -> string
val to_float : t -> float
val to_mpzf2 : t -> Mpzf.t * Mpzf.t
```

4.4 Arithmetic Functions

```
val add : t -> t -> t
val sub : t -> t -> t
val mul : t -> t -> t
val div : t -> t -> t
val neg : t -> t
val abs : t -> t
val inv : t -> t
val equal : t -> t -> bool
```

4.5 Comparison Functions

```
val cmp : t -> t -> int
val cmp_int : t -> int -> int
val cmp_frac : t -> int -> int -> int
val sgn : t -> int
```

4.6 Extraction Functions

```
val get_num : t -> Mpzf.t
val get_den : t -> Mpzf.t
```

Chapter 5

Module Mpf : GMP multiprecision floating-point numbers

type t

GMP multiprecision floating-point numbers

The following operations are mapped as much as possible to their C counterpart. In case of imperative functions (like set, add, ...) the first parameter of type t is an out-parameter and holds the result when the function returns. For instance, add x y z adds the values of y and z and stores the result in x.

These functions are as efficient as their C counterpart: they do not imply additional memory allocation.

5.1 Pretty printing

```
val print : Format.formatter -> t -> unit
```

5.2 Initialization Functions

C documentation[http://gmplib.org/manual/Initializing-Floats.html#Initializing-Floats]

```
val set_default_prec : int -> unit
val get_default_prec : unit -> int
val init : unit -> t
val init2 : int -> t
val get_prec : t -> int
val set_prec : t -> int -> unit
val set_prec_raw : t -> int -> unit
```

5.3 Assignement Functions

C documentation[http://gmplib.org/manual/Assigning-Floats.html#Assigning-Floats]

```
val set : t -> t -> unit
val set_si : t -> int -> unit
val set_d : t -> float -> unit
val set_z : t -> Mpz.t -> unit
val set_q : t -> Mpq.t -> unit
```

```
val _set_str : t -> string -> int -> unit
val set_str : t -> string -> base:int -> unit
val swap : t -> t -> unit
```

5.4 Combined Initialization and Assignement Functions

C documentation[http://gmplib.org/manual/Simultaneous-Float-Init-_0026-Assign.html#Simultaneous-Float-In

```
val init_set : t -> t
val init_set_si : int -> t
val init_set_d : float -> t
val _init_set_str : string -> int -> t
val init_set_str : string -> base:int -> t
```

5.5 Conversion Functions

C documentation[http://gmplib.org/manual/Converting-Floats.html#Converting-Floats]

```
val get_d : t -> float
val get_d_2exp : t -> float * int
val get_si : t -> nativeint
val get_int : t -> int
val get_z : Mpz.t -> t -> unit
val get_q : Mpq.t -> t -> unit
val _get_str : int -> int -> t -> string * int
val get_str : base:int -> digits:int -> t -> string * int
```

5.6 User Conversions

These functionss are additions to or renaming of functions offered by the C library.

```
val to_string : t -> string
val to_float : t -> float
val of_string : string -> t
val of_float : float -> t
val of_int : int -> t
val of_mpz : Mpz.t -> t
val of_mpq : Mpq.t -> t
val is_integer : t -> bool
```

5.7 Arithmetic Functions

C documentation[http://gmplib.org/manual/Float-Arithmetic.html#Float-Arithmetic]

```
val add : t -> t -> t -> unit
val add_ui : t -> t -> int -> unit
val sub : t -> t -> t -> unit
val ui_sub : t -> int -> t -> unit
val sub_ui : t -> t -> int -> unit
```

```
val mul : t -> t -> t -> unit
val mul_ui : t -> t -> int -> unit
val mul_2exp : t -> t -> int -> unit
val div : t -> t -> t -> unit
val ui_div : t -> int -> t -> unit
val div_ui : t -> t -> int -> unit
val div_2exp : t -> t -> int -> unit
val sqrt : t -> t -> unit
val pow_ui : t -> t -> int -> unit
val neg : t -> t -> unit
val abs : t -> t -> unit
```

5.8 Comparison Functions

C documentation[http://gmplib.org/manual/Float-Comparison.html#Float-Comparison]

```
val cmp : t -> t -> int
val cmp_d : t -> float -> int
val cmp_si : t -> int -> int
val sgn : t -> int
val _equal : t -> t -> int -> bool
val equal : t -> t -> bits:int -> bool
val reldiff : t -> t -> t -> unit
```

5.9 Input and Output Functions: not interfaced

5.10 Random Number Functions: see Gmp_random[8] module

5.11 Miscellaneous Float Functions

C documentation[http://gmplib.org/manual/Miscellaneous-Float-Functions.html#Miscellaneous-Float-Functions

```
val ceil : t -> t -> unit
val floor : t -> t -> unit
val trunc : t -> t -> unit
val integer_p : t -> bool
val fits_int_p : t -> bool
val fits_ulong_p : t -> bool
val fits_slong_p : t -> bool
val fits_uint_p : t -> bool
val fits_sint_p : t -> bool
val fits_ushort_p : t -> bool
val fits_sshort_p : t -> bool
```

Chapter 6

Module Mpfr : MPFR multiprecision floating-point numbers

```
type t
type round =
  | Near
  | Zero
  | Up
  | Down
```

MPFR multiprecision floating-point numbers

The following operations are mapped as much as possible to their C counterpart. In case of imperative functions (like `set`, `add`, ...) the first parameter of type `t` is an out-parameter and holds the result when the function returns. For instance, `add x y z` adds the values of `y` and `z` and stores the result in `x`.

These functions are as efficient as their C counterpart: they do not imply additional memory allocation.

6.1 Pretty printing

```
val print : Format.formatter -> t -> unit
val print_round : Format.formatter -> round -> unit
val string_of_round : round -> string
```

6.2 Rounding Modes

C documentation[http://www.mpfr.org/mpfr-current/mpfr.html#Rounding-Related-Functions]

```
val set_default_rounding_mode : round -> unit
val get_default_rounding_mode : unit -> round
val round_prec : t -> round -> int -> int
```

6.3 Exceptions

C documentation[http://www.mpfr.org/mpfr-current/mpfr.html#Exception-Related-Functions]

```
val get_emin : unit -> int
val get_emax : unit -> int
val set_emin : int -> unit
```

```
val set_emax : int -> unit
val check_range : t -> int -> round -> int
val clear_underflow : unit -> unit
val clear_overflow : unit -> unit
val clear_nanflag : unit -> unit
val clear_inexflag : unit -> unit
val clear_flags : unit -> unit
val underflow_p : unit -> bool
val overflow_p : unit -> bool
val nanflag_p : unit -> bool
val inexflag_p : unit -> bool
```

6.4 Initialization Functions

C documentation[http://www.mpfr.org/mpfr-current/mpfr.html#Initialization-Functions]

```
val set_default_prec : int -> unit
val get_default_prec : unit -> int
val init : unit -> t
val init2 : int -> t
val get_prec : t -> int
val set_prec : t -> int -> unit
val set_prec_raw : t -> int -> unit
```

6.5 Assignment Functions

C documentation[http://www.mpfr.org/mpfr-current/mpfr.html#Assignment-Functions]

```
val set : t -> t -> round -> int
val set_si : t -> int -> round -> int
val set_d : t -> float -> round -> int
val set_z : t -> Mpz.t -> round -> int
val set_q : t -> Mpq.t -> round -> int
val _set_str : t -> string -> int -> round -> unit
val set_str : t -> string -> base:int -> round -> unit
val set_f : t -> Mpf.t -> round -> int
val set_si_2exp : t -> int -> int -> round -> int
val set_inf : t -> int -> unit
val set_nan : t -> unit
val swap : t -> t -> unit
```

6.6 Combined Initialization and Assignment Functions

C documentation[http://www.mpfr.org/mpfr-current/mpfr.html#Combined-Initialization-and-Assignment-Functi

```
val init_set : t -> round -> int * t
val init_set_si : int -> round -> int * t
val init_set_d : float -> round -> int * t
```

```
val init_set_f : Mpf.t -> round -> int * t
val init_set_z : Mpz.t -> round -> int * t
val init_set_q : Mpq.t -> round -> int * t
val _init_set_str : string -> int -> round -> t
val init_set_str : string -> base:int -> round -> t
```

6.7 Conversion Functions

C documentation[http://www.mpfr.org/mpfr-current/mpfr.html#Conversion-Functions]

```
val get_d : t -> round -> float
val get_d1 : t -> float
val get_z_exp : Mpz.t -> t -> int
val get_z : Mpz.t -> t -> round -> unit
val _get_str : int -> int -> t -> round -> string * int
val get_str : base:int -> digits:int -> t -> round -> string * int
```

6.8 User Conversions

These functionss are additions to or renaming of functions offered by the C library.

```
val to_string : t -> string
val to_float : ?round:round -> t -> float
val to_mpq : t -> Mpq.t
val of_string : string -> round -> t
val of_float : float -> round -> t
val of_int : int -> round -> t
val of_frac : int -> int -> round -> t
val of_mpz : Mpz.t -> round -> t
val of_mpz2 : Mpz.t -> Mpz.t -> round -> t
val of_mpq : Mpq.t -> round -> t
```

6.9 Basic Arithmetic Functions

C documentation[http://www.mpfr.org/mpfr-current/mpfr.html#Basic-Arithmetic-Functions]

```
val add : t -> t -> t -> round -> int
val add_ui : t -> t -> int -> round -> int
val add_z : t -> t -> Mpz.t -> round -> int
val add_q : t -> t -> Mpq.t -> round -> int
val sub : t -> t -> t -> round -> int
val ui_sub : t -> int -> t -> round -> int
val sub_ui : t -> t -> int -> round -> int
val sub_z : t -> t -> Mpz.t -> round -> int
val sub_q : t -> t -> Mpq.t -> round -> int
val mul : t -> t -> t -> round -> int
val mul_ui : t -> t -> int -> round -> int
val mul_z : t -> t -> Mpz.t -> round -> int
```

```
val mul_q : t -> t -> Mpq.t -> round -> int
val mul_2ui : t -> t -> int -> round -> int
val mul_2si : t -> t -> int -> round -> int
val mul_2exp : t -> t -> int -> round -> int
val div : t -> t -> t -> round -> int
val ui_div : t -> int -> t -> round -> int
val div_ui : t -> t -> int -> round -> int
val div_z : t -> t -> Mpz.t -> round -> int
val div_q : t -> t -> Mpq.t -> round -> int
val div_2ui : t -> t -> int -> round -> int
val div_2si : t -> t -> int -> round -> int
val div_2exp : t -> t -> int -> round -> int
val sqrt : t -> t -> round -> bool
val sqrt_ui : t -> int -> round -> bool
val pow_ui : t -> t -> int -> round -> bool
val pow_si : t -> t -> int -> round -> bool
val ui_pow_ui : t -> int -> int -> round -> bool
val ui_pow : t -> int -> t -> round -> bool
val pow : t -> t -> t -> round -> bool
val neg : t -> t -> round -> int
val abs : t -> t -> round -> int
```

6.10 Comparison Functions

C documentation[http://www.mpfr.org/mpfr-current/mpfr.html#Comparison-Functions]

```
val cmp : t -> t -> int
val cmp_si : t -> int -> int
val cmp_si_2exp : t -> int -> int -> int
val sgn : t -> int
val _equal : t -> t -> int -> bool
val equal : t -> t -> bits:int -> bool
val nan_p : t -> bool
val inf_p : t -> bool
val number_p : t -> bool
val reldiff : t -> t -> t -> round -> unit
```

6.11 Special Functions

C documentation[http://www.mpfr.org/mpfr-current/mpfr.html#Special-Functions]

```
val log : t -> t -> round -> int
val log2 : t -> t -> round -> int
val log10 : t -> t -> round -> int
val exp : t -> t -> round -> int
val exp2 : t -> t -> round -> int
val exp10 : t -> t -> round -> int
```

```
val cos : t -> t -> round -> int
val sin : t -> t -> round -> int
val tan : t -> t -> round -> int
val sec : t -> t -> round -> int
val csc : t -> t -> round -> int
val cot : t -> t -> round -> int
val sin_cos : t -> t -> t -> round -> bool
val acos : t -> t -> round -> int
val asin : t -> t -> round -> int
val atan : t -> t -> round -> int
val atan2 : t -> t -> t -> round -> int
val cosh : t -> t -> round -> int
val sinh : t -> t -> round -> int
val tanh : t -> t -> round -> int
val sech : t -> t -> round -> int
val csch : t -> t -> round -> int
val coth : t -> t -> round -> int
val acosh : t -> t -> round -> int
val asinh : t -> t -> round -> int
val atanh : t -> t -> round -> int
val fac_ui : t -> int -> round -> int
val log1p : t -> t -> round -> int
val expm1 : t -> t -> round -> int
val eint : t -> t -> round -> int
val gamma : t -> t -> round -> int
val lngamma : t -> t -> round -> int
val zeta : t -> t -> round -> int
val erf : t -> t -> round -> int
val erfc : t -> t -> round -> int
val fma : t -> t -> t -> t -> round -> int
val agm : t -> t -> t -> round -> int
val hypot : t -> t -> t -> round -> int
val const_log2 : t -> round -> int
val const_pi : t -> round -> int
val const_euler : t -> round -> int
val const_catalan : t -> round -> int
```

6.12 Input and Output Functions: not interfaced

6.13 Miscellaneous Float Functions

C documentation[http://www.mpfr.org/mpfr-current/mpfr.html#Rounding-Related-Functions]

```
val rint : t -> t -> round -> int
val ceil : t -> t -> int
val floor : t -> t -> int
```

```
val round : t -> t -> int
val trunc : t -> t -> int
val integer_p : t -> bool
val nexttoward : t -> t -> unit
val nextabove : t -> unit
val nextbelow : t -> unit
val min : t -> t -> t -> round -> int
val max : t -> t -> t -> round -> int
val get_exp : t -> int
val set_exp : t -> int -> int
```

Chapter 7

Module `Mpfrf` : MPFR multi-precision floating-point version, functional version

Functions in this module has a functional semantics, unlike the corresponding functions in `Mpfr`[6]. These functions do not return the rounding information and are less efficients, due to the additional memory allocation neded for the result.

```
type t = Mpfr.t
```

multi-precision floating-point numbers

```
val to_mpfr : t -> Mpfr.t
val of_mpfr : Mpfr.t -> t
```

Safe conversion from and to Mpfr.t.

There is no sharing between the argument and the result.

```
val mpfr : t -> Mpfr.t
val mpfrf : Mpfr.t -> t
```

Unsafe conversion from and to Mpfr.t.

The argument and the result actually share the same number: be cautious !

Conversion from and to Mpz.t, Mpq.t and Mpfr.t There is no sharing between the argument and the result.

7.1 Pretty-printing

```
val print : Format.formatter -> t -> unit
```

7.2 Constructors

```
val of_string : string -> Mpfr.round -> t
val of_float : float -> Mpfr.round -> t
val of_int : int -> Mpfr.round -> t
val of_frac : int -> int -> Mpfr.round -> t
```

```
val of_mpz : Mpz.t -> Mpfr.round -> t
val of_mpz2 : Mpz.t -> Mpz.t -> Mpfr.round -> t
val of_mpzf : Mpzf.t -> Mpfr.round -> t
val of_mpzf2 : Mpzf.t -> Mpzf.t -> Mpfr.round -> t
val of_mpq : Mpq.t -> Mpfr.round -> t
val of_mpqf : Mpqf.t -> Mpfr.round -> t
```

7.3 Conversions

```
val to_string : t -> string
val to_float : ?round:Mpfr.round -> t -> float
val to_mpqf : t -> Mpqf.t
```

7.4 Arithmetic Functions

```
val add : t -> t -> Mpfr.round -> t
val add_int : t -> int -> Mpfr.round -> t
val sub : t -> t -> Mpfr.round -> t
val sub_int : t -> int -> Mpfr.round -> t
val mul : t -> t -> Mpfr.round -> t
val mul_ui : t -> int -> Mpfr.round -> t
val ui_div : int -> t -> Mpfr.round -> t
val div : t -> t -> Mpfr.round -> t
val div_ui : t -> int -> Mpfr.round -> t
val sqrt : t -> Mpfr.round -> t
val ui_pow : int -> t -> Mpfr.round -> t
val pow : t -> t -> Mpfr.round -> t
val pow_int : t -> int -> Mpfr.round -> t
val neg : t -> Mpfr.round -> t
val abs : t -> Mpfr.round -> t
```

7.5 Comparison Functions

```
val equal : t -> t -> bits:int -> bool
val cmp : t -> t -> int
val cmp_int : t -> int -> int
val sgn : t -> int
val nan_p : t -> bool
val inf_p : t -> bool
val number_p : t -> bool
```

Chapter 8

Module Gmp_random : GMP random generation functions

```
type state
```
GMP random generation functions

8.1 Random State Initialization

C documentation[http://gmplib.org/manual/Random-State-Initialization.html#Random-State-Initialization]

```
val init_default : unit -> state
val init_lc_2exp : Mpz.t -> int -> int -> state
val init_lc_2exp_size : int -> state
```

8.2 Random State Seeding

C documentation[http://gmplib.org/manual/Random-State-Seeding.html#Random-State-Seeding]

```
val seed : state -> Mpz.t -> unit
val seed_ui : state -> int -> unit
```

8.3 Random Number Functions

8.3.1 Integers (Mpz[1])

C documentation[http://gmplib.org/manual/Integer-Random-Numbers.html#Integer-Random-Numbers]

```
module Mpz :
  sig

     val urandomb : Mpz.t -> Gmp_random.state -> int -> unit
     val urandomm : Mpz.t -> Gmp_random.state -> Mpz.t -> unit
     val rrandomb : Mpz.t -> Gmp_random.state -> int -> unit
  end
```

8.3.2 Floating-point (Mpf[5])

C documentation[http://gmplib.org/manual/Miscellaneous-Float-Functions.html#Miscellaneous-Float-Functions]

```
module Mpf :
  sig

    val urandomb : Mpf.t -> Gmp_random.state -> int -> unit
  end
```

8.3.3 Floating-point (Mpfr[6])

C documentation[http://www.mpfr.org/mpfr-current/mpfr.html#Miscellaneous-Functions]

```
module Mpfr :
  sig

    val urandomb : Mpfr.t -> Gmp_random.state -> unit
  end
```

Index