

MLApronIDL: OCaml interface for APRON library

Bertrand Jeannet

October 31, 2007

All files distributed in the APRON library, including MLAPRONIDL subpackage, are distributed under LGPL license.

Copyright (C) Bertrand Jeannet 2005-2006 for the MLAPRONIDL subpackage.

Contents

1	Introduction	7
I	Coefficients	10
2	Module Scalar : APRON Scalar numbers.	11
3	Module Interval : APRON Intervals on scalars	13
4	Module Coeff : APRON Coefficients (either scalars or intervals)	15
II	Managers and Abstract Domains	17
5	Module Manager : APRON Managers	18
6	Module Box : Intervals abstract domain	21
6.1	Compilation information	21
7	Module Oct : Octagon abstract domain.	23
7.1	Compilation information	24
8	Module Polka : Convex Polyhedra and Linear Equalities abstract domains	25
8.1	Compilation information	26
9	Module Ppl : Convex Polyhedra and Linear Congruences abstract domains (PPL wrapper)	27
9.1	Compilation information	28
10	Module PolkaGrid : Reduced product of NewPolka polyhedra and PPL grids	29
10.1	Compilation information	29
III	Level 1 of the interface	31
11	Module Var : APRON Variables	32
12	Module Environment : APRON Environments binding dimensions to names	33
13	Module Linexpr1 : APRON Expressions of level 1	35
14	Module Lincons1 : APRON Constraints and array of constraints of level 1	37
14.1	Type array	39

15 Module Generator1 : APRON Generators and array of generators of level 1	40
15.1 Type earray	41
16 Module Texpr1 : APRON Expressions of level 1	43
16.1 Constructors and Destructor	44
16.2 Tests	44
16.3 Operations	44
16.4 Printing	45
17 Module Tcons1 : APRON tree constraints and array of tree constraints of level 1	46
17.1 Type array	47
18 Module Abstract1 : APRON Abstract values of level 1	48
18.1 General management	48
18.2 Constructor, accessors, tests and property extraction	49
18.3 Operations	51
18.4 Additional operations	54
19 Module Parser : APRON Parsing of expressions	55
19.1 Introduction	55
19.2 Interface	55
IV Level 0 of the interface	57
20 Module Dim : APRON Dimensions and related types	58
21 Module Linexpr0 : APRON Linear expressions of level 0	59
22 Module Lincons0 : APRON Linear constraints of level 0	61
23 Module Generator0 : APRON Generators of level 0	62
24 Module Texpr0	63
24.1 Constructors and Destructor	64
24.2 Tests	64
24.3 Printing	64
24.4 Internal usage for level 1	65
25 Module Tcons0 : APRON tree expressions constraints of level 0	66
26 Module Abstract0 : APRON Abstract value of level 0	67
26.1 General management	67
26.2 Constructor, accessors, tests and property extraction	68
26.3 Operations	69
26.4 Additional operations	72
V MLGmpIDL modules	74
27 Module Mpz : GMP multi-precision integers	75
27.1 Pretty printing	75

27.2	Initialization Functions	75
27.3	Assignment Functions	75
27.4	Combined Initialization and Assignment Functions	75
27.5	Conversion Functions	76
27.6	User Conversions	76
27.7	Arithmetic Functions	76
27.8	Division Functions	76
27.9	Exponentiation Functions	78
27.10	Root Extraction Functions	78
27.11	Number Theoretic Functions	78
27.12	Comparison Functions	79
27.13	Logical and Bit Manipulation Functions	79
27.14	Input and Output Functions: not interfaced	79
27.15	Random Number Functions: see <code>Gmp_random</code> [31] module	79
27.16	Integer Import and Export Functions	79
27.17	Miscellaneous Functions	80
28	Module <code>Mpq</code> : GMP multiprecision rationals	81
28.1	Pretty printing	81
28.2	Initialization and Assignment Functions	81
28.3	Additional Initialization and Assignments functions	81
28.4	Conversion Functions	82
28.5	User Conversions	82
28.6	Arithmetic Functions	82
28.7	Comparison Functions	82
28.8	Applying Integer Functions to Rationals	82
28.9	Input and Output Functions: not interfaced	83
29	Module <code>Mpf</code> : GMP multiprecision floating-point numbers	84
29.1	Pretty printing	84
29.2	Initialization and Assignment Functions	84
29.3	Conversion Functions	85
29.4	User Conversions	85
29.5	Arithmetic Functions	85
29.6	Comparison Functions	86
29.7	Input and Output Functions: not interfaced	86
29.8	Random Number Functions: see <code>Gmp_random</code> [31] module	86
29.9	Miscellaneous Float Functions	86
30	Module <code>Mpfr</code> : MPFR multiprecision floating-point numbers	87
30.1	Pretty printing	87
30.2	Rounding Modes	87
30.3	Exceptions	87
30.4	Initialization and Assignment Functions	88
30.5	Conversion Functions	88
30.6	User Conversions	89
30.7	Basic Arithmetic Functions	89
30.8	Comparison Functions	89

30.9 Special Functions	90
30.10 Input and Output Functions: not interfaced	91
30.11 Miscellaneous Float Functions	91
31 Module Gmp_random : GMP random generation functions	92
31.1 Random State Initialization	92
31.2 Random State Seeding	92
31.3 Random Number Functions	92
32 Module Mpzf : GMP multi-precision integers, functional version	93
32.1 Constructors	93
32.2 Conversions and Printing	93
32.3 Arithmetic Functions	94
32.4 Comparison Functions	94
33 Module Mpqf : GMP multi-precision rationals, functional version	95
33.1 Constructors	95
33.2 Conversions and Printing	95
33.3 Arithmetic Functions	95
33.4 Comparison Functions	96
33.5 Extraction Functions	96

Chapter 1

Introduction

This package is an OCAML interface for the APRON library/interface. The interface is accessed via the module `Apron`, which is decomposed into 15 submodules, corresponding to C modules:

<code>Scalar</code>	: scalars (numbers)
<code>Interval</code>	: intervals on scalars
<code>Coeff</code>	: coefficients (either scalars or intervals)
<code>Dimension</code>	: dimensions and related operations
<code>Linexpr0</code>	: (interval) linear expressions, level 0
<code>Lincons0</code>	: (interval) linear constraints, level 0
<code>Generator0</code>	: generators, level 0
<code>texpr0</code>	: tree expressions, level 0
<code>tcons0</code>	: tree constraints, level 0
<code>Manager</code>	: managers
<code>Abstract0</code>	: abstract values, level 0
<code>Var</code>	: variables
<code>Environment</code>	: environment binding variables to dimensions
<code>Linexpr1</code>	: (interval) linear expressions, level 1
<code>Lincons1</code>	: (interval) linear constraints, level 1
<code>Generator1</code>	: generators, level 1
<code>texpr1</code>	: tree expressions, level 1
<code>tcons1</code>	: tree constraints, level 1
<code>Abstract1</code>	: abstract values, level 1
<code>Parser</code>	: strings parsing

The package also includes the `MLGMPIDL` wrapper to GMP and MPFR libraries.

Requirements

- APRON library
- GMP library version 4.2 or up (tested with version 4.2.1)
- MPFR library version 2.2 or up (tested with version 2.2.1)
- OCaml 3.0 or up (tested with 3.09)
- Camlidl (tested with 1.05)

Optionally,

- GNU M4 preprocessor
- GNU sed

Installation

Library Set the file `../Makefile.config` to your own setting.

type `'make'`, and then `'make install'`

The OCaml part of the library is named `apron.cma` (`.cmxa`, `.a`) The C part of the library, which is automatically referenced by `apron.cma/apron.cmxa`, is named `libapron_caml.a` (`libapron_caml_debug.a`) `'make install'` installs not only `.mli`, `.cmi`, but also `.idl` files.

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

`'make mlapronidl.pdf'`

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

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

Compilation and Linking

To make things clearer, we assume an example file `example.ml` which uses both `NEWPOLKA` (convex polyhedra) and `BOX` (intervals) libraries, in their versions where rationals are GMP rationals (which is the default). We assume that C and OCaml interface and library files are located in directory `$APRON/lib`. The native-code compilation command looks like

```
ocamlopt -I $APRON/lib -o example.opt bigarray.cmxa gmp.cmxa apron.cmxa box.cmxa
polka.cmxa
```

Comments:

1. You need at least the libraries `bigarray` (standard OCAML distribution), `gmp`, and `apron` (standard APRON distribution), plus the one implementing an effective abstract domains: here, `box`, and `polka`.
2. The C libraries associated to those OCAML libraries (*e.g.*, `gmp_caml`, `box_caml`, ...) are automatically looked for, as well as the the libraries implementing abstract domains (*e.g.*, `polka`, `box` in *their default version* (which corresponds to the MPQ suffix).

If other versions of abstract domains library are wanted, you should use the `-noautolink` option as explained below.

The byte-code compilation process looks like

```
ocamlc -I $APRON/lib -make-runtime -o myrun bigarray.cma gmp.cma apron.cma box.cma
polka.cma
```

```
ocamlc -I $APRON/lib -use-runtime myrun -o exampleg bigarray.cma gmp.cma apron.cma
box.cma polka.cma example.ml
```

Comments:

1. One first build a custom bytecode interpreter that includes the new native-code needed;
2. One then compile the `example.ml` file.

The automatic search for C libraries associated to these OCAML libraries can be disabled by the option `-noautolink` supported by both `ocamlc` and `ocamlopt` commands. For instance, the command for native-code compilation can alternatively looks like:

```
ocamlopt -I $APRON/lib -noautolink -o example.opt bigarray.cmxa gmp.cmxa apron.cmxa
box.cmxa polka.cmxa -cclib "-lpolka_caml lpolka -lbox_caml -lbox -lapron_caml
-lapron -lgmp_caml -LMPFR -lmpfr -LGMP/lib -lgmp"
```


This is mandatory if you want to use non-default versions of abstract domains library. For instance; if you want to use POLKA in its I11 version (`long long int`) and BOX in its D version (`double`), the command looks like:

```
ocamlopt -I $APRON/lib -noautolink -o example.opt bigarray.cmxa gmp.cmxa apron.cmxa
box.cmxa polka.cmxa -cclib "-lpolka_caml lpolkaI11 -lbox_caml -lboxD -lapron_caml
-lapron -lgmp_caml -LMPFR -lmpfr -LGMP/lib -lgmp"
```

The option `-verbose` helps to understand what is happening in case of problem. More details are given in the modules implementing a specific abstract domain.

Part I

Coefficients

Chapter 2

Module Scalar : APRON Scalar numbers.

See `Mpqf`[33] for operations on GMP multiprecision rational numbers.

```
type t =  
  | Float of float  
  | Mpqf of Mpqf.t
```

APRON Scalar numbers. See `Mpqf`[33] for operations on GMP multiprecision rational numbers.

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

```
val of_mpqf : Mpqf.t -> t
```

```
val of_int : int -> t
```

```
val of_frac : int -> int -> t
```

Create a scalar of type `Mpqf` from resp.

- A multi-precision rational `Mpq.t`
- A multi-precision rational `Mpqf.t`
- an integer
- a fraction `x/y`

```
val of_float : float -> t
```

Create a scalar of type `Float` with the given value

```
val of_infty : int -> t
```

Create a scalar of type `Float` with the value multiplied by infinity (resulting in minus infinity, zero, or infinity)

```
val is_infty : t -> int
```

Infinity test. `is_infty x` returns `-1` if `x` is `-oo`, `1` if `x` is `+oo`, and `0` if `x` is finite.

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

Return the sign of the coefficient, which may be a negative value, zero or a positive value.

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

Compare two coefficients, possibly converting one from `float` to `Mpqf.t`. `compare x y` returns a negative number if `x` is less than `y`, `0` if they are equal, and a positive number if `x` is greater than `y`.

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

Compare a coefficient with an integer

```
val equal : t -> t -> bool
```

Equality test, possibly using a conversion from `float` to `Mpqf.t`. Return `true` if the 2 values are equal. Two infinite values of the same signs are considered as equal.

```
val equal_int : t -> int -> bool
```

Equality test with an integer

```
val neg : t -> t
```

Negation

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

Conversion to string, using either `string_of_double` or `Mpqf.to_string`

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

Print a coefficient

Chapter 3

Module Interval : APRON Intervals on scalars

```
type t = {
  mutable inf : Scalar.t ;
  mutable sup : Scalar.t ;
}
APRON Intervals on scalars
val of_scalar : Scalar.t -> Scalar.t -> t
    Build an interval from a lower and an upper bound

val of_infsup : Scalar.t -> Scalar.t -> t
    deprecated

val of_mpq : Mpq.t -> Mpq.t -> t
val of_mpqf : Mpqf.t -> Mpqf.t -> t
val of_int : int -> int -> t
val of_frac : int -> int -> int -> int -> t
val of_float : float -> float -> t
    Create an interval from resp. two
        • multi-precision rationals Mpq.t
        • multi-precision rationals Mpqf.t
        • integers
        • fractions x/y and z/w
        • floats

val is_top : t -> bool
    Does the interval represent the universe  $([-\infty, +\infty])$  ?

val is_bottom : t -> bool
    Does the interval contain no value  $([a, b]$  with  $a > b$ ) ?

val is_leq : t -> t -> bool
    Inclusion test. is_leq x y returns true if x is included in y
```

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

Non Total Comparison: 0: equality -1: i1 included in i2 +1: i2 included in i1 -2: i1.inf less than or equal to i2.inf +2: i1.inf greater than i2.inf

```
val equal : t -> t -> bool
```

Equality test

```
val is_zero : t -> bool
```

Is the coefficient equal to scalar 0 or interval 0,0 ?

```
val neg : t -> t
```

Negation

```
val top : t
```

```
val bottom : t
```

Top and bottom intervals (using DOUBLE coefficients)

```
val set_infsup : t -> Scalar.t -> Scalar.t -> unit
```

Fill the interval with the given lower and upper bounds

```
val set_top : t -> unit
```

```
val set_bottom : t -> unit
```

Fill the interval with top (resp. bottom) value

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

Print an interval, under the format [inf,sup]

Chapter 4

Module Coeff : APRON Coefficients (either scalars or intervals)

```
type union_5 =  
  | Scalar of Scalar.t  
  | Interval of Interval.t
```

```
type t = union_5
```

APRON Coefficients (either scalars or intervals)

```
val s_of_mpq : Mpq.t -> t
```

```
val s_of_mpqf : Mpqf.t -> t
```

```
val s_of_int : int -> t
```

```
val s_of_frac : int -> int -> t
```

Create a scalar coefficient of type `Mpqf.t` from resp.

- A multi-precision rational `Mpq.t`
- A multi-precision rational `Mpqf.t`
- an integer
- a fraction `x/y`

```
val s_of_float : float -> t
```

Create an interval coefficient of type `Float` with the given value

```
val i_of_scalar : Scalar.t -> Scalar.t -> t
```

Build an interval from a lower and an upper bound

```
val i_of_mpq : Mpq.t -> Mpq.t -> t
```

```
val i_of_mpqf : Mpqf.t -> Mpqf.t -> t
```

```
val i_of_int : int -> int -> t
```

```
val i_of_frac : int -> int -> int -> int -> t
```

```
val i_of_float : float -> float -> t
```

Create an interval coefficient from resp. two

- multi-precision rationals `Mpq.t`
- multi-precision rationals `Mpqf.t`
- integers

- fractions x/y and z/w
- floats

`val is_scalar : t -> bool`

`val is_interval : t -> bool`

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

Non Total Comparison:

- If the 2 coefficients are both scalars, corresp. to `Scalar.cmp`
- If the 2 coefficients are both intervals, corresp. to `Interval.cmp`
- otherwise, -3 if the first is a scalar, 3 otherwise

`val equal : t -> t -> bool`

Equality test

`val is_zero : t -> bool`

Is the coefficient equal to scalar 0 or interval 0,0 ?

`val neg : t -> t`

Negation

`val reduce : t -> t`

Convert interval to scalar if possible

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

Printing

Part II

Managers and Abstract Domains

Chapter 5

Module Manager : APRON Managers

```
type tbool =
  | False
  | True
  | Top

type funid =
  | Funid_unknown
  | Funid_copy
  | Funid_free
  | Funid_asize
  | Funid_minimize
  | Funid_canonicalize
  | Funid_approximate
  | Funid_fprint
  | Funid_fprintdiff
  | Funid_fdump
  | Funid_serialize_raw
  | Funid_deserialize_raw
  | Funid_bottom
  | Funid_top
  | Funid_of_box
  | Funid_dimension
  | Funid_is_bottom
  | Funid_is_top
  | Funid_is_leq
  | Funid_is_eq
  | Funid_is_dimension_unconstrained
  | Funid_sat_interval
  | Funid_sat_lincons
  | Funid_sat_tcons
  | Funid_bound_dimension
  | Funid_bound_linexpr
  | Funid_bound_texpr
  | Funid_to_box
  | Funid_to_lincons_array
  | Funid_to_tcons_array
  | Funid_to_generator_array
  | Funid_meet
  | Funid_meet_array
  | Funid_meet_lincons_array
```

```
| Funid_meet_tcons_array
| Funid_join
| Funid_join_array
| Funid_add_ray_array
| Funid_assign_linexpr_array
| Funid_substitute_linexpr_array
| Funid_assign_texpr_array
| Funid_substitute_texpr_array
| Funid_add_dimensions
| Funid_remove_dimensions
| Funid_permute_dimensions
| Funid_forget_array
| Funid_expand
| Funid_fold
| Funid_widening
| Funid_closure
| Funid_change_environment
| Funid_rename_array

type funopt = {
  algorithm : int ;
  timeout : int ;
  max_object_size : int ;
  flag_exact_wanted : bool ;
  flag_best_wanted : bool ;
}

type exc =
| Exc_none
| Exc_timeout
| Exc_out_of_space
| Exc_overflow
| Exc_invalid_argument
| Exc_not_implemented

type exclog = {
  exn : exc ;
  funid : funid ;
  msg : string ;
}

type 'a t
```

APRON Managers

The type parameter `'a` allows to distinguish managers allocated by different underlying abstract domains. Concerning the other types,

- `tbool` extends Booleans;
- `funid` defines identifiers for the generic function working on abstract values;
- `funopt` defines the options associated to generic functions;
- `exc` defines the different kind of exceptions;
- `exclog` defines the exceptions raised by APRON functions.

```
val get_library : 'a t -> string
```

Get the name of the effective library which allocated the manager

```

val get_version : 'a t -> string
    Get the version of the effective library which allocated the manager

val funopt_make : unit -> funopt
    Return the default options for any function (0 or false for al fields)

val get_funopt : 'a t -> funid -> funopt
    Get the options sets for the function. The result is a copy of the internal structure and may be
    freely modified. funid should be different from Funid_change_environment and
    Funid_rename_array (no option associated to them).

val set_funopt : 'a t -> funid -> funopt -> unit
    Set the options for the function. funid should be different from Funid_change_environment and
    Funid_rename_array (no option associated to them).

val get_flag_exact : 'a t -> tbool
    Get the corresponding result flag

val get_flag_best : 'a t -> tbool
    Get the corresponding result flag

exception Error of exclog
    Exception raised by functions of the interface

val string_of_tbool : tbool -> string
val string_of_funid : funid -> string
val string_of_exc : exc -> string
val print_tbool : Format.formatter -> tbool -> unit
val print_funid : Format.formatter -> funid -> unit
val print_funopt : Format.formatter -> funopt -> unit
val print_exc : Format.formatter -> exc -> unit
val print_exclog : Format.formatter -> exclog -> unit
    Printing functions

val set_deserialize : 'a t -> unit
    Set / get the global manager used for deserialization

val get_deserialize : unit -> 'a t

```

Chapter 6

Module Box : Intervals abstract domain

type t

Type of boxes.

Boxes constrains each dimension/variable `x_i` to belong to an interval `I_i`.

Abstract values which are boxes have the type `t Apron.AbstractX.t`.

Managers allocated for boxes have the type `t Apron.manager.t`.

val manager_alloc : unit -> t Apron.Manager.t

Create a Box manager.

6.1 Compilation information

6.1.1 Bytecode compilation

To compile to bytecode, you should first generate a custom interpreter with a command which should look like:

```
ocamlc -I $APRON_PREFIX/lib -make-runtime -o myrun bigarray.cma gmp.cma apron.cma box.cma
```

and then you compile and link your example `X.ml` with

```
ocamlc -I $APRON_PREFIX/lib -c X.ml and
```

```
ocamlc -I $APRON_PREFIX/lib -use-runtime myrun -o X bigarray.cma gmp.cma apron.cma box.cma X.cmo
```

Comments: The C libraries related to `gmp.cma` and `apron.cma` are automatically looked for (thanks to the auto-linking feature provided by `ocamlc`). For `box.cma`, the library `libbox.a`, identic to `lib-boxMPQ.a`, is selected by default. The `-noautolink` option should be used to select a differetn version. See the C documentation of Box library for details.

With the `-noautolink` option, the generation of the custom runtime executable should be done with

```
ocamlc -I $APRON_PREFIX/lib -noautolink -make-runtime -o myrun bigarray.cma gmp.cma apron.cma box.cma -ccopt "-L$GMP_PREFIX/lib ..." -cclib "-lbox_caml -lboxMPQ -lapron_caml -lapron -lgmp_caml -lmpfr -lgmp -lbigarray -lcamlidl"
```

6.1.2 Native-code compilation

You compile and link with

```
ocamlopt -I $APRON_PREFIX/lib -c X.ml and
```

```
ocamlopt -I $APRON_PREFIX/lib -o X bigarray.cmx gmp.cmx apron.cmx box.cmx X.cmx
```

Comments: Same as for bytecode compilation. With the `-noautolink` option, the linking command becomes

```
ocamlopt -I $APRON_PREFIX/lib -o X bigarray.cmx gmp.cmx apron.cmx box.cmx -ccopt "-  
L$GMP_PREFIX/lib ..." -cclib "-lbox_caml -lboxMPQ -lapron_caml -lapron -lgmp_caml -lmpfr  
-lgmp -lbigarray -lcamlidl" X.cmx
```

Chapter 7

Module Oct : Octagon abstract domain.

```
type internal
Octagon abstract domain.
type t
  Type of octagons.
  Octagons are defined by conjunctions of inequalities of the form  $\pm x_i \pm x_j \geq 0$ .
  Abstract values which are octagons have the type t Apron.AbstractX.t.
  Managers allocated for octagons have the type t Apron.manager.t.

val manager_alloc : unit -> t Apron.Manager.t
  Allocate a new manager to manipulate octagons.

val manager_get_internal : t Apron.Manager.t -> internal
  No internal parameters for now...

val of_generator_array :
  t Apron.Manager.t ->
  int -> int -> Apron.Generator0.t array -> t Apron.Abstract0.t
  Approximate a set of generators to an abstract value, with best precision.

val widening_thresholds :
  t Apron.Manager.t ->
  t Apron.Abstract0.t ->
  t Apron.Abstract0.t -> Apron.Scalar.t array -> t Apron.Abstract0.t
  Widening with scalar thresholds.

val narrowing :
  t Apron.Manager.t ->
  t Apron.Abstract0.t -> t Apron.Abstract0.t -> t Apron.Abstract0.t
  Standard narrowing.

val add_epsilon :
  t Apron.Manager.t ->
  t Apron.Abstract0.t -> Apron.Scalar.t -> t Apron.Abstract0.t
  Perturbation.
```

```

val add_epsilon_bin :
  t Apron.Manager.t ->
  t Apron.Abstract0.t ->
  t Apron.Abstract0.t -> Apron.Scalar.t -> t Apron.Abstract0.t
  Perturbation.

val pre_widening : int
  Algorithms.

```

7.1 Compilation information

7.1.1 Bytecode compilation

To compile to bytecode, you should first generate a custom interpreter with a command which should look like:

```

ocamlc -I $APRON_PREFIX/lib -make-runtime -o myrun bigarray.cma gmp.cma apron.cma oct.cma
and then you compile and link your example X.ml with
ocamlc -I $APRON_PREFIX/lib -c X.ml and
ocamlc -I $APRON_PREFIX/lib -use-runtime myrun -o X bigarray.cma gmp.cma apron.cma oct.cma
X.cmo

```

Comments: The C libraries related to `gmp.cma` and `apron.cma` are automatically looked for (thanks to the auto-linking feature provided by `ocamlc`). For `oct.cma`, the library `liboct.a`, identic to `liboctMPQ.a`, is selected by default. The `-noautolink` option should be used to select a differetn version. See the C documentation of `Oct` library for details.

With the `-noautolink` option, the generation of the custom runtime executable should be done with

```

ocamlc -I $APRON_PREFIX/lib -noautolink -make-runtime -o myrun bigarray.cma gmp.cma apron.cma
oct.cma -ccopt "-L$GMP_PREFIX/lib ..." -cclib "-loct_caml -loctMPQ -lapron_caml -lapron
-lgmp_caml -lmpfr -lgmp -lbigarray -lcamlidl"

```

7.1.2 Native-code compilation

You compile and link with

```

ocamlopt -I $APRON_PREFIX/lib -c X.ml and
ocamlopt -I $APRON_PREFIX/lib -o X bigarray.cmxa gmp.cmxa apron.cmxa oct.cmxa X.cmx

```

Comments: Same as for bytecode compilation. With the `-noautolink` option, the linking command becomes

```

ocamlopt -I $APRON_PREFIX/lib -o X bigarray.cmxa gmp.cmxa apron.cmxa oct.cmxa -ccopt "-
L$GMP_PREFIX/lib ..." -cclib "-loct_caml -loctMPQ -lapron_caml -lapron -lgmp_caml -lmpfr
-lgmp -lbigarray -lcamlidl" X.cmx

```


Chapter 8

Module Polka : Convex Polyhedra and Linear Equalities abstract domains

`type internal`

Convex Polyhedra and Linear Equalities abstract domains

`type loose`

`type strict`

Two flavors for convex polyhedra: loose or strict.

Loose polyhedra cannot have strict inequality constraints like $x > 0$. They are algorithmically more efficient (less generators, simpler normalization).

Convex polyhedra are defined by the conjunction of a set of linear constraints of the form $a_0 \cdot x_0 + \dots + a_n \cdot x_n + b \geq 0$ or $a_0 \cdot x_0 + \dots + a_n \cdot x_n + b > 0$ where a_0, \dots, a_n, b, c are constants and x_0, \dots, x_n variables.

`type equalities`

Linear equalities.

Linear equalities are conjunctions of linear equalities of the form $a_0 \cdot x_0 + \dots + a_n \cdot x_n + b = 0$.

`type 'a t`

Type of convex polyhedra/linear equalities, where 'a is loose, strict or equalities.

Abstract values which are convex polyhedra have the type `(loose t) Apron.Abstract0.t` or `(loose t) Apron.Abstract1.t` or `(strict t) Apron.Abstract0.t` or `(strict t) Apron.Abstract1.t`.

Abstract values which are conjunction of linear equalities have the type `(equalities t) Apron.Abstract0.t` or `(equalities t) Apron.Abstract1.t`.

Managers allocated by NewPolka have the type `'a t Apron.Manager.t`.

`val manager_alloc_loose : unit -> loose t Apron.Manager.t`

Create a NewPolka manager for loose convex polyhedra.

`val manager_alloc_strict : unit -> strict t Apron.Manager.t`

Create a NewPolka manager for strict convex polyhedra.

```
val manager_alloc_equalities : unit -> equalities t Apron.Manager.t
```

Create a NewPolka manager for conjunctions of linear equalities.

```
val manager_get_internal : 'a t Apron.Manager.t -> internal
```

Get the internal submanager of a NewPolka manager.

Various options. See the C documentation

```
val set_max_coeff_size : internal -> int -> unit
```

```
val set_approximate_max_coeff_size : internal -> int -> unit
```

```
val get_max_coeff_size : internal -> int
```

```
val get_approximate_max_coeff_size : internal -> int
```

8.1 Compilation information

8.1.1 Bytecode compilation

To compile to bytecode, you should first generate a custom interpreter with a command which should look like:

```
ocamlc -I $APRON_PREFIX/lib -make-runtime -o myrun bigarray.cma gmp.cma apron.cma polka.cma -cclib "-lpolkag"
```

and then you compile and link your example X.ml with

```
ocamlc -I $APRON_PREFIX/lib -c X.ml and
```

```
ocamlc -I $APRON_PREFIX/lib -use-runtime myrun -o X bigarray.cma gmp.cma apron.cma polka.cma X.cmo
```

Comments: The C libraries related to `gmp.cma` and `apron.cma` are automatically looked for (thanks to the auto-linking feature provided by `ocamlc`). For `polka.cma`, the library `libpolka.a`, identic to `libpolkaMPQ.a`, is selected by default. The `-noautolink` option should be used to select a differetn version. See the C documentation of Polka library for details.

With the `-noautolink` option, the generation of the custom runtime executable should be done with

```
ocamlc -I $APRON_PREFIX/lib -noautolink -make-runtime -o myrun bigarray.cma gmp.cma apron.cma polka.cma -ccopt "-L$GMP_PREFIX/lib ..." -cclib "-lpolka_caml -lpolka -lapron_caml -lapron -lgmp_caml -lmpfr -lgmp -lbigarray -lcamlidl"
```

8.1.2 Native-code compilation

You compile and link with

```
ocamlopt -I $APRON_PREFIX/lib -c X.ml and
```

```
ocamlopt -I $APRON_PREFIX/lib -o X bigarray.cmxa gmp.cmxa apron.cmxa polka.cmxa -cclib "-lpolkag" X.cmx
```

Comments: Same as for bytecode compilation. With the `-noautolink` option, the linking command becomes

```
ocamlopt -I $APRON_PREFIX/lib -o X bigarray.cmxa gmp.cmxa apron.cmxa polka.cmxa -ccopt "-L$GMP_PREFIX/lib ..." -cclib "-lpolka_caml -lpolkaMPQ -lapron_caml -lapron -lgmp_caml -lmpfr -lgmp -lbigarray -lcamlidl" X.cmx
```

Chapter 9

Module Ppl : Convex Polyhedra and Linear Congruences abstract domains (PPL wrapper)

This module is a wrapper around the Parma Polyhedra Library.

`type loose`

`type strict`

Two flavors for convex polyhedra: loose or strict.

Loose polyhedra cannot have strict inequality constraints like $x > 0$. They are algorithmically more efficient (less generators, simpler normalization).

Convex polyhedra are defined by the conjunction of a set of linear constraints of the form $a_0 \cdot x_0 + \dots + a_n \cdot x_n + b \geq 0$ or $a_0 \cdot x_0 + \dots + a_n \cdot x_n + b > 0$ where a_0, \dots, a_n, b, c are constants and x_0, \dots, x_n variables.

`type grid`

Linear congruences.

Linear congruences are defined by the conjunction of equality constraints modulo a rational number, of the form $a_0 \cdot x_0 + \dots + a_n \cdot x_n = b \bmod c$, where a_0, \dots, a_n, b, c are constants and x_0, \dots, x_n variables.

`type 'a t`

Type of convex polyhedra/linear congruences, where `'a` is `loose`, `strict` or `grid`.

Abstract values which are convex polyhedra have the type `loose t Apron.AbstractX.t` or `strict t Apron.AbstractX.t`.

Abstract values which are conjunction of linear congruences equalities have the type `grid t Apron.AbstractX.t`.

Managers allocated by PPL have the type `'a t Apron.Manager.t`.

`val manager_alloc_loose : unit -> loose t Apron.Manager.t`

Allocate a PPL manager for loose convex polyhedra.

`val manager_alloc_strict : unit -> strict t Apron.Manager.t`

Allocate a PPL manager for strict convex polyhedra.

`val manager_alloc_grid : unit -> grid t Apron.Manager.t`

Allocate a new manager for linear congruences (grids)

9.1.1 Bytecode compilation

To compile to bytecode, you should first generate a custom interpreter with a command which should look like:

```
ocamlc -I $APRON_PREFIX/lib -make-runtime -o myrun -cc "g++" bigarray.cma gmp.cma apron.cma ppl.cma
```

and then you compile and link your example `X.ml` with

```
ocamlc -I $APRON_PREFIX/lib -c X.ml and
```

```
ocamlc -I $APRON_PREFIX/lib -use-runtime myrun -o X bigarray.cma gmp.cma apron.cma ppl.cma X.cmo
```

Comments:

Do not forget the `-cc "g++"` option: PPL is a C++ library which requires a C++ linker.

The C libraries related to `gmp.cma`, `apron.cma` and `ppl.cma` are automatically looked for (thanks to the auto-linking feature provided by `ocamlc`). Be aware that PPL requires the C++ wrapper `libgmpxx.a` library on top of GMP library, which should thus be installed. With the `-noautolink` option, the generation of the custom runtime executable should be done with

```
ocamlc -I $APRON_PREFIX/lib -noautolink -make-runtime -o myrun -cc "g++" bigarray.cma gmp.cma apron.cma ppl.cma -ccopt "-L$GMP_PREFIX/lib ..." -cclib "-lap_ppl_caml -lap_ppl -lppl -lgmpxx -lapron_caml -lapron -lgmp_caml -lmpfr -lgmp -lbigarray -lcamlidl"
```

9.1.2 Native-code compilation

You compile and link with

```
ocamlopt -I $APRON_PREFIX/lib -c X.ml and
```

```
ocamlopt -I $APRON_PREFIX/lib -o X -cc "g++" bigarray.cmxa gmp.cmxa apron.cmxa ppl.cmxa X.cmx
```

Comments: Same as for bytecode compilation. Do not forget the `-cc "g++"` option. With the `-noautolink` option, the linking command becomes

```
ocamlopt -I $APRON_PREFIX/lib -o X -cc "g++" bigarray.cmxa gmp.cmxa apron.cmxa ppl.cmxa -ccopt "-L$GMP_PREFIX/lib ..." -cclib "-lap_ppl_caml -lap_ppl -lppl -lgmpxx -lapron_caml -lapron -lgmp_caml -lmpfr -lgmp -lbigarray -lcamlidl" X.cmx
```

Chapter 10

Module PolkaGrid : Reduced product of NewPolka polyhedra and PPL grids

`type 'a t`

Type of abstract values, where 'a is `Polka.loose` or `Polka.strict`.

`val manager_alloc_loose : unit -> Polka.loose t Apron.Manager.t`

Create a PolkaGrid manager with loose convex polyhedra.

`val manager_alloc_strict : unit -> Polka.strict t Apron.Manager.t`

Create a PolkaGrid manager with strict convex polyhedra.

10.1 Compilation information

10.1.1 Bytecode compilation

To compile to bytecode, you should first generate a custom interpreter with a command which should look like:

```
ocamlc -I $APRON_PREFIX/lib -make-runtime -o myrun -cc "g++" bigarray.cma gmp.cma apron.cma polkaGrid.cma
```

and then you compile and link your example `X.ml` with

```
ocamlc -I $APRON_PREFIX/lib -c X.ml and
```

```
ocamlc -I $APRON_PREFIX/lib -use-runtime myrun -o X bigarray.cma gmp.cma apron.cma polkaGrid.cma X.cmo
```

Comments: The C libraries related to `gmp.cma`, `apron.cma` and `polkaGrid.cma` are automatically looked for (thanks to the auto-linking feature provided by `ocamlc`).

With the `-noautolink` option, the generation of the custom runtime executable should be done with

```
ocamlc -I $APRON_PREFIX/lib -noautolink -make-runtime -o myrun bigarray.cma gmp.cma apron.cma polkaGrid.cma -ccopt "-L$GMP_PREFIX/lib ..." -cclib "-lpolkaGrid_caml -lap_pkggrid -lpolka -lap_ppl -lppl -lgmpxx -lapron_caml -lapron -lgmp_caml -lmpfr -lgmp -lbigarray -lcamlidl"
```

10.1.2 Native-code compilation

You compile and link with

`ocamlopt -I $APRON_PREFIX/lib -c X.ml` and

`ocamlopt -I $APRON_PREFIX/lib -o X bigarray.cmx gmp.cmx apron.cmx polkaGrid.cmx X.cmx`

Comments: Same as for bytecode compilation. With the `-noautolink` option, the linking command becomes

`ocamlopt -I $APRON_PREFIX/lib -o X bigarray.cmx gmp.cmx apron.cmx polka.cmx ppl.cmx
-cc "g++" -ccopt "-L$GMP_PREFIX/lib ..." -cclib "-lpolkaGrid_caml -lap_pkggrid -lpolka -
lap_ppl -lppl -lgmpxx -lapron_caml -lapron -lgmp_caml -lmpfr -lgmp -lbigarray -lcamlidl"
X.cmx`

Part III

Level 1 of the interface

Chapter 11

Module Var : APRON Variables

```
type t
val of_string : string -> t
val compare : t -> t -> int
val to_string : t -> string
val hash : t -> int
APRON Variables
val print : Format.formatter -> t -> unit
```


Chapter 12

Module Environment : APRON Environments binding dimensions to names

```
type typvar =  
  | INT  
  | REAL  
type t  
APRON Environments binding dimensions to names  
val make : Var.t array -> Var.t array -> t  
    Making an environment from a set of integer and real variables. Raise Failure in case of name  
    conflict.  
  
val add : t -> Var.t array -> Var.t array -> t  
    Adding to an environment a set of integer and real variables. Raise Failure in case of name  
    conflict.  
  
val remove : t -> Var.t array -> t  
    Remove from an environment a set variables. Raise Failure in case of non-existing variables.  
  
val lce : t -> t -> t  
    Compute the least common environment of 2 environment, that is, the environment composed of  
    all the variables of the 2 environments. Raise Failure if the same variable has different types in  
    the 2 environment.  
  
val equal : t -> t -> bool  
    Test equality if two environments  
  
val compare : t -> t -> int  
    Compare two environment. compare env1 env2 return -2 if the environements are not  
    compatible (a variable has different types in the 2 environements), -1 if env1 is a subset of env2,  
    0 if equality, +1 if env1 is a superset of env2, and +2 otherwise (the lce exists and is a strict  
    superset of both)  
  
val dimension : t -> Dim.dimension  
    Return the dimension of the environment
```

```
val size : t -> int
```

Return the size of the environment

```
val mem_var : t -> Var.t -> bool
```

Return true if the variable is present in the environment.

```
val typ_of_var : t -> Var.t -> typvar
```

Return the type of variables in the environment. If the variable does not belong to the environment, raise a **Failure** exception.

```
val vars : t -> Var.t array * Var.t array
```

Return the (lexicographically ordered) sets of integer and real variables in the environment

```
val var_of_dim : t -> Dim.t -> Var.t
```

Return the variable corresponding to the given dimension in the environment. Raise **Failure** if the dimension is out of the range of the environment (greater than or equal to `dim env`)

```
val dim_of_var : t -> Var.t -> Dim.t
```

Return the dimension associated to the given variable in the environment. Raise **Failure** if the variable does not belong to the environment.

```
val print :
```

```
  ?first:(unit, Format.formatter, unit) Pervasives.format ->
```

```
  ?sep:(unit, Format.formatter, unit) Pervasives.format ->
```

```
  ?last:(unit, Format.formatter, unit) Pervasives.format ->
```

```
  Format.formatter -> t -> unit
```

Printing

Chapter 13

Module Linexpr1 : APRON Expressions of level 1

```
type t = {  
  mutable linexpr0 : Linexpr0.t ;  
  mutable env : Environment.t ;  
}  
APRON Expressions of level 1  
val make : ?sparse:bool -> Environment.t -> t  
    Build a linear expression defined on the given argument, which is sparse by default.  
  
val minimize : t -> unit  
    In case of sparse representation, remove zero coefficients  
  
val copy : t -> t  
    Copy  
  
val print : Format.formatter -> t -> unit  
    Print the linear expression  
  
val set_list : t -> (Coeff.t * Var.t) list -> Coeff.t option -> unit  
    Set simultaneously a number of coefficients.  
    set_list expr [(c1,"x"); (c2,"y")] (Some cst) assigns coefficients c1 to variable "x",  
    coefficient c2 to variable "y", and coefficient cst to the constant. If (Some cst) is replaced by  
    None, the constant coefficient is not assigned.  
  
val set_array : t -> (Coeff.t * Var.t) array -> Coeff.t option -> unit  
    Set simultaneously a number of coefficients, as set_list.  
  
val iter : (Coeff.t -> Var.t -> unit) -> t -> unit  
    Iter the function on the pair coefficient/variable of the linear expression  
  
val get_cst : t -> Coeff.t  
    Get the constant  
  
val set_cst : t -> Coeff.t -> unit  
    Set the constant
```

```
val get_coeff : t -> Var.t -> Coeff.t
```

Get the coefficient of the variable

```
val set_coeff : t -> Var.t -> Coeff.t -> unit
```

Set the coefficient of the variable

```
val extend_environment : t -> Environment.t -> t
```

Change the environment of the expression for a super-environnement. Raise **Failure** if it is not the case

```
val extend_environment_with : t -> Environment.t -> unit
```

Side-effet version of the previous function

```
val is_integer : t -> bool
```

Does the linear expression depend only on integer variables ?

```
val is_real : t -> bool
```

Does the linear expression depend only on real variables ?

```
val get_linexpr0 : t -> Linexpr0.t
```

Get the underlying expression of level 0 (which is not a copy).

```
val get_env : t -> Environment.t
```

Get the environnement of the expression

Chapter 14

Module Lincons1 : APRON Constraints and array of constraints of level 1

```
type t = {
  mutable lincons0 : Lincons0.t ;
  mutable env : Environment.t ;
}

type earray = {
  mutable lincons0_array : Lincons0.t array ;
  mutable array_env : Environment.t ;
}

APRON Constraints and array of constraints of level 1

type typ = Lincons0.typ =
| EQ
| SUPEQ
| SUP
| DISEQ
| EQMOD of Scalar.t

val make : Linexpr1.t -> typ -> t
    Make a linear constraint. Modifying later the linear expression (not advisable) modifies
    correspondingly the linear constraint and conversely, except for changes of environements

val copy : t -> t
    Copy (deep copy)

val string_of_typ : typ -> string
    Convert a constraint type to a string (=,>=, or >)

val print : Format.formatter -> t -> unit
    Print the linear constraint

val get_typ : t -> typ
    Get the constraint type

val iter : (Coeff.t -> Var.t -> unit) -> t -> unit
```

Iter the function on the pair coefficient/variable of the underlying linear expression

`val get_cst : t -> Coeff.t`

Get the constant of the underlying linear expression

`val set_typ : t -> typ -> unit`

Set the constraint type

`val set_list : t -> (Coeff.t * Var.t) list -> Coeff.t option -> unit`

Set simultaneously a number of coefficients.

`set_list` `expr [(c1,"x"); (c2,"y")] (Some cst)` assigns coefficients `c1` to variable `"x"`, coefficient `c2` to variable `"y"`, and coefficient `cst` to the constant. If `(Some cst)` is replaced by `None`, the constant coefficient is not assigned.

`val set_array : t -> (Coeff.t * Var.t) array -> Coeff.t option -> unit`

Set simultaneously a number of coefficients, as `set_list`.

`val set_cst : t -> Coeff.t -> unit`

Set the constant of the underlying linear expression

`val get_coeff : t -> Var.t -> Coeff.t`

Get the coefficient of the variable in the underlying linear expression

`val set_coeff : t -> Var.t -> Coeff.t -> unit`

Set the coefficient of the variable in the underlying linear expression

`val make_unsat : Environment.t -> t`

Build the unsatisfiable constraint $-1 \geq 0$

`val is_unsat : t -> bool`

Is the constraint not satisfiable ?

`val extend_environment : t -> Environment.t -> t`

Change the environnement of the constraint for a super-environment. Raise `Failure` if it is not the case

`val extend_environment_with : t -> Environment.t -> unit`

Side-effect version of the previous function

`val get_env : t -> Environment.t`

Get the environnement of the linear constraint

`val get_linexpr1 : t -> Linexpr1.t`

Get the underlying linear expression. Modifying the linear expression (*not advisable*) modifies correspondingly the linear constraint and conversely, except for changes of environments

`val get_lincons0 : t -> Lincons0.t`

Get the underlying linear constraint of level 0. Modifying the constraint of level 0 (*not advisable*) modifies correspondingly the linear constraint and conversely, except for changes of environments

14.1 Type array

```
val array_make : Environment.t -> int -> earray
```

Make an array of linear constraints with the given size and defined on the given environment.
The elements are initialized with the constraint `0=0`.

```
val array_print :  
  ?first:(unit, Format.formatter, unit) Pervasives.format ->  
  ?sep:(unit, Format.formatter, unit) Pervasives.format ->  
  ?last:(unit, Format.formatter, unit) Pervasives.format ->  
  Format.formatter -> earray -> unit
```

Print an array of constraints

```
val array_length : earray -> int
```

Get the size of the array

```
val array_get_env : earray -> Environment.t
```

Get the environment of the array

```
val array_get : earray -> int -> t
```

Get the element of the given index (which is not a copy)

```
val array_set : earray -> int -> t -> unit
```

Set the element of the given index (without any copy). The array and the constraint should be defined on the same environment; otherwise a **Failure** exception is raised.

```
val array_extend_environment : earray -> Environment.t -> earray
```

Change the environment of the array of constraints for a super-environment. Raise **Failure** if it is not the case

```
val array_extend_environment_with : earray -> Environment.t -> unit
```

Side-effect version of the previous function

Chapter 15

Module Generator1 : APRON Generators and array of generators of level 1

```
type t = {
  mutable generator0 : Generator0.t ;
  mutable env : Environment.t ;
}

type earray = {
  mutable generator0_array : Generator0.t array ;
  mutable array_env : Environment.t ;
}

APRON Generators and array of generators of level 1
type typ = Generator0.typ =
  | LINE
  | RAY
  | VERTEX
  | LINEMOD
  | RAYMOD

val make : Linexpr1.t -> Generator0.typ -> t
    Make a generator. Modifying later the linear expression (not advisable) modifies correspondingly
    the generator and conversely, except for changes of environements

val copy : t -> t
    Copy (deep copy)

val print : Format.formatter -> t -> unit
    Print the generator

val get_typ : t -> Generator0.typ
    Get the generator type

val iter : (Coeff.t -> Var.t -> unit) -> t -> unit
    Iter the function on the pair coefficient/variable of the underlying linear expression

val set_typ : t -> Generator0.typ -> unit
```


Set the generator type

```
val set_list : t -> (Coeff.t * Var.t) list -> unit
```

Set simultaneously a number of coefficients.

`set_list expr [(c1,"x"); (c2,"y")]` assigns coefficients `c1` to variable `"x"` and coefficient `c2` to variable `"y"`.

```
val set_array : t -> (Coeff.t * Var.t) array -> unit
```

Set simultaneously a number of coefficients, as `set_list`.

```
val get_coeff : t -> Var.t -> Coeff.t
```

Get the coefficient of the variable in the underlying linear expression

```
val set_coeff : t -> Var.t -> Coeff.t -> unit
```

Set the coefficient of the variable in the underlying linear expression

```
val extend_environment : t -> Environment.t -> t
```

Change the environnement of the generator for a super-environment. Raise **Failure** if it is not the case

```
val extend_environment_with : t -> Environment.t -> unit
```

Side-effect version of the previous function

15.1 Type earray

```
val array_make : Environment.t -> int -> earray
```

Make an array of generators with the given size and defined on the given environnement. The elements are initialized with the line 0.

```
val array_print :
```

```
  ?first:(unit, Format.formatter, unit) Pervasives.format ->
```

```
  ?sep:(unit, Format.formatter, unit) Pervasives.format ->
```

```
  ?last:(unit, Format.formatter, unit) Pervasives.format ->
```

```
  Format.formatter -> earray -> unit
```

Print an array of generators

```
val array_length : earray -> int
```

Get the size of the array

```
val array_get : earray -> int -> t
```

Get the element of the given index (which is not a copy)

```
val array_set : earray -> int -> t -> unit
```

Set the element of the given index (without any copy). The array and the generator should be defined on the same environnement; otherwise a **Failure** exception is raised.

```
val array_extend_environment : earray -> Environment.t -> earray
```

Change the environnement of the array of generators for a super-environment. Raise **Failure** if it is not the case

```
val array_extend_environment_with : earray -> Environment.t -> unit
```

Side-effect version of the previous function

`val get_env : t -> Environment.t`

Get the environnement of the generator

`val get_linexpr1 : t -> Linexpr1.t`

Get the underlying linear expression. Modifying the linear expression (*not advisable*) modifies correspondingly the generator and conversely, except for changes of environements

`val get_generator0 : t -> Generator0.t`

Get the underlying generator of level 0. Modifying the generator of level 0 (*not advisable*) modifies correspondingly the generator and conversely, except for changes of environements

Chapter 16

Module Texpr1 : APRON Expressions of level 1

```
type t = {  
  mutable texpr0 : Texpr0.t ;  
  mutable env : Environment.t ;  
}
```

APRON Expressions of level 1

```
type unop = Texpr0.unop =  
  | Neg  
  | Cast  
  | Sqrt  
  Unary operators
```

```
type binop = Texpr0.binop =  
  | Add  
  | Sub  
  | Mul  
  | Div  
  | Mod  
  Binary operators
```

```
type typ = Texpr0.typ =  
  | Real  
  | Int  
  | Single  
  | Double  
  | Extended  
  | Quad  
  Destination type for rounding
```

```
type round = Texpr0.round =  
  | Near  
  | Zero  
  | Up  
  | Down  
  | Rnd  
  Rounding direction
```

```

type expr =
  | Cst of Coeff.t
  | Var of Var.t
  | Unop of unop * expr * typ * round
  | Binop of binop * expr * expr * typ * round
  User type for tree expressions

```

16.1 Constructors and Destructor

```

val of_expr : Environment.t -> expr -> t
  General constructor (actually the most efficient)

val copy : t -> t
  Copy

val of_linexpr : Linexpr1.t -> t
  Conversion

val to_expr : t -> expr
  General destructor

```

16.1.1 Incremental constructors

```

val cst : Environment.t -> Coeff.t -> t
val var : Environment.t -> Var.t -> t
val unop : Texpr0.unop -> t -> Texpr0.typ -> Texpr0.round -> t
val binop : Texpr0.binop ->
  t -> t -> Texpr0.typ -> Texpr0.round -> t

```

16.2 Tests

```

val is_interval_cst : t -> bool
val is_interval_linear : t -> bool
val is_interval_polynomial : t -> bool
val is_interval_polyfrac : t -> bool
val is_scalar : t -> bool

```

16.3 Operations

```

val extend_environment : t -> Environment.t -> t
  Change the environment of the expression for a super-environment. Raise Failure if it is not
  the case

val extend_environment_with : t -> Environment.t -> unit
  Side-effet version of the previous function

val get_texpr0 : t -> Texpr0.t

```

Get the underlying expression of level 0 (which is not a copy).

```
val get_env : t -> Environment.t
```

Get the environnement of the expression

16.4 Printing

```
val string_of_unop : unop -> string
```

```
val string_of_binop : binop -> string
```

```
val string_of_typ : typ -> string
```

```
val string_of_round : round -> string
```

```
val print_unop : Format.formatter -> unop -> unit
```

```
val print_binop : Format.formatter -> binop -> unit
```

```
val print_typ : Format.formatter -> typ -> unit
```

```
val print_round : Format.formatter -> round -> unit
```

```
val print_expr : Format.formatter -> expr -> unit
```

Print a tree expression

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

Print an abstract tree expression

Chapter 17

Module Tcons1 : APRON tree constraints and array of tree constraints of level 1

```
type t = {
  mutable tcons0 : Tcons0.t ;
  mutable env : Environment.t ;
}

type earray = {
  mutable tcons0_array : Tcons0.t array ;
  mutable array_env : Environment.t ;
}

APRON tree constraints and array of tree constraints of level 1
type typ = Lincons0.typ =
| EQ
| SUPEQ
| SUP
| DISEQ
| EQMOD of Scalar.t

val make : Texpr1.t -> typ -> t
    Make a tree expression constraint. Modifying later the linear expression (not advisable) modifies
    correspondingly the tree expression constraint and conversely, except for changes of environements

val copy : t -> t
    Copy (deep copy)

val string_of_typ : typ -> string
    Convert a constraint type to a string (=,>=, or >)

val print : Format.formatter -> t -> unit
    Print the tree expression constraint

val get_typ : t -> typ
    Get the constraint type

val set_typ : t -> typ -> unit
```

Set the constraint type

```
val extend_environment : t -> Environment.t -> t
```

Change the environnement of the constraint for a super-environment. Raise **Failure** if it is not the case

```
val extend_environment_with : t -> Environment.t -> unit
```

Side-effect version of the previous function

```
val get_env : t -> Environment.t
```

Get the environnement of the tree expression constraint

```
val get_texpr1 : t -> Texpr1.t
```

Get the underlying linear expression. Modifying the linear expression (*not advisable*) modifies correspondingly the tree expression constraint and conversely, except for changes of environnements

```
val get_tcons0 : t -> Tcons0.t
```

Get the underlying tree expression constraint of level 0. Modifying the constraint of level 0 (*not advisable*) modifies correspondingly the tree expression constraint and conversely, except for changes of environnements

17.1 Type array

```
val array_make : Environment.t -> int -> earray
```

Make an array of tree expression constraints with the given size and defined on the given environnement. The elements are initialized with the constraint 0=0.

```
val array_print :
```

```
  ?first:(unit, Format.formatter, unit) Pervasives.format ->
```

```
  ?sep:(unit, Format.formatter, unit) Pervasives.format ->
```

```
  ?last:(unit, Format.formatter, unit) Pervasives.format ->
```

```
  Format.formatter -> earray -> unit
```

Print an array of constraints

```
val array_length : earray -> int
```

Get the size of the array

```
val array_get_env : earray -> Environment.t
```

Get the environment of the array

```
val array_get : earray -> int -> t
```

Get the element of the given index (which is not a copy)

```
val array_set : earray -> int -> t -> unit
```

Set the element of the given index (without any copy). The array and the constraint should be defined on the same environnement; otherwise a **Failure** exception is raised.

```
val array_extend_environment : earray -> Environment.t -> earray
```

Change the environnement of the array of constraints for a super-environment. Raise **Failure** if it is not the case

```
val array_extend_environment_with : earray -> Environment.t -> unit
```

Side-effect version of the previous function

Chapter 18

Module Abstract1 : APRON

Abstract values of level 1

```
type 'a t = {  
  mutable abstract0 : 'a Abstract0.t ;  
  mutable env : Environment.t ;  
}
```

APRON Abstract values of level 1

The type parameter 'a allows to distinguish abstract values with different underlying abstract domains.

```
type box1 = {  
  mutable interval_array : Interval.t array ;  
  mutable box1_env : Environment.t ;  
}
```

18.1 General management

18.1.1 Memory

```
val copy : 'a Manager.t -> 'a t -> 'a t  
  Copy a value
```

```
val size : 'a Manager.t -> 'a t -> int  
  Return the abstract size of a value
```

18.1.2 Control of internal representation

```
val minimize : 'a Manager.t -> 'a t -> unit  
  Minimize the size of the representation of the value. This may result in a later recomputation of  
  internal information.
```

```
val canonicalize : 'a Manager.t -> 'a t -> unit  
  Put the abstract value in canonical form. (not yet clear definition)
```

```
val approximate : 'a Manager.t -> 'a t -> int -> unit  
  approximate man abs alg perform some transformation on the abstract value, guided by the  
  argument alg. The transformation may lose information. The argument alg overrides the field
```


algorithm of the structure of type `Manager.funopt` associated to `ap_abstract0_approximate` (commodity feature).

18.1.3 Printing

```
val fdump : 'a Manager.t -> 'a t -> unit
```

Dump on the `stdout` C stream the internal representation of an abstract value, for debugging purposes

```
val print : Format.formatter -> 'a t -> unit
```

Print as a set of constraints

18.1.4 Serialization

18.2 Constructor, accessors, tests and property extraction

18.2.1 Basic constructors

All these functions request explicitly an environment in their arguments.

```
val bottom : 'a Manager.t -> Environment.t -> 'a t
```

Create a bottom (empty) value defined on the given environment

```
val top : 'a Manager.t -> Environment.t -> 'a t
```

Create a top (universe) value defined on the given environment

```
val of_box :
```

```
'a Manager.t ->
```

```
Environment.t -> Var.t array -> Interval.t array -> 'a t
```

Abstract an hypercube.

`of_box man env tvar tinterval` abstracts an hypercube defined by the arrays `tvar` and `tinterval`. The result is defined on the environment `env`, which should contain all the variables in `tvar` (and defines their type)

18.2.2 Accessors

```
val manager : 'a t -> 'a Manager.t
```

```
val env : 'a t -> Environment.t
```

```
val abstract0 : 'a t -> 'a Abstract0.t
```

Return resp. the underlying manager, environment and abstract value of level 0

18.2.3 Tests

```
val is_bottom : 'a Manager.t -> 'a t -> Manager.tbool
```

Emptiness test

```
val is_top : 'a Manager.t -> 'a t -> Manager.tbool
```

Universality test

```
val is_leq : 'a Manager.t -> 'a t -> 'a t -> Manager.tbool
```

Inclusion test. The 2 abstract values should be compatible.

```
val is_eq : 'a Manager.t -> 'a t -> 'a t -> Manager.tbool
```

Equality test. The 2 abstract values should be compatible.

```
val sat_lincons : 'a Manager.t -> 'a t -> Lincons1.t -> Manager.tbool
```

Does the abstract value satisfy the linear constraint ?

```
val sat_tcons : 'a Manager.t -> 'a t -> Tcons1.t -> Manager.tbool
```

Does the abstract value satisfy the tree expression constraint ?

```
val sat_interval :  
  'a Manager.t -> 'a t -> Var.t -> Interval.t -> Manager.tbool
```

Does the abstract value satisfy the constraint `dim in interval` ?

```
val is_variable_unconstrained :  
  'a Manager.t -> 'a t -> Var.t -> Manager.tbool
```

Is the variable unconstrained in the abstract value ? If yes, this means that the existential quantification of the dimension does not change the value.

18.2.4 Extraction of properties

```
val bound_variable : 'a Manager.t -> 'a t -> Var.t -> Interval.t
```

Return the interval of variation of the variable in the abstract value.

```
val bound_linexpr : 'a Manager.t -> 'a t -> Linexpr1.t -> Interval.t
```

Return the interval of variation of the linear expression in the abstract value.

Implement a form of linear programming, where the argument linear expression is the one to optimize under the constraints induced by the abstract value.

```
val bound_texpr : 'a Manager.t -> 'a t -> Texpr1.t -> Interval.t
```

Return the interval of variation of the tree expression in the abstract value.

```
val to_box : 'a Manager.t -> 'a t -> box1
```

Convert the abstract value to an hypercube

```
val to_lincons_array : 'a Manager.t -> 'a t -> Lincons1.earray
```

Convert the abstract value to a conjunction of linear constraints.

Convert the abstract value to a conjunction of tree expressions constraints.

```
val to_tcons_array : 'a Manager.t -> 'a t -> Tcons1.earray
```

```
val to_generator_array : 'a Manager.t -> 'a t -> Generator1.earray
```

Convert the abstract value to a set of generators that defines it.

18.3 Operations

18.3.1 Meet and Join

```
val meet : 'a Manager.t -> 'a t -> 'a t -> 'a t
```

Meet of 2 abstract values.

```
val meet_array : 'a Manager.t -> 'a t array -> 'a t
```

Meet of a non empty array of abstract values.

```
val meet_lincons_array : 'a Manager.t -> 'a t -> Lincons1.earray -> 'a t
```

Meet of an abstract value with an array of linear constraints.

```
val meet_tcons_array : 'a Manager.t -> 'a t -> Tcons1.earray -> 'a t
```

Meet of an abstract value with an array of tree expressions constraints.

```
val join : 'a Manager.t -> 'a t -> 'a t -> 'a t
```

Join of 2 abstract values.

```
val join_array : 'a Manager.t -> 'a t array -> 'a t
```

Join of a non empty array of abstract values.

```
val add_ray_array : 'a Manager.t -> 'a t -> Generator1.earray -> 'a t
```

Add the array of generators to the abstract value (time elapse operator).

The generators should either lines or rays, not vertices.

18.3.1.0.1 Side-effect versions of the previous functions

```
val meet_with : 'a Manager.t -> 'a t -> 'a t -> unit
```

```
val meet_lincons_array_with : 'a Manager.t -> 'a t -> Lincons1.earray -> unit
```

```
val meet_tcons_array_with : 'a Manager.t -> 'a t -> Tcons1.earray -> unit
```

```
val join_with : 'a Manager.t -> 'a t -> 'a t -> unit
```

```
val add_ray_array_with : 'a Manager.t -> 'a t -> Generator1.earray -> unit
```

18.3.2 Assignment and Substitutions

```
val assign_linexpr_array :
```

```
'a Manager.t ->
```

```
'a t ->
```

```
Var.t array -> Linexpr1.t array -> 'a t option -> 'a t
```

Parallel assignement of an array of dimensions by an array of same size of linear expressions

```
val substitute_linexpr_array :
```

```
'a Manager.t ->
```

```
'a t ->
```

```
Var.t array -> Linexpr1.t array -> 'a t option -> 'a t
```

Parallel substitution of an array of dimensions by an array of same size of linear expressions

```
val assign_texpr_array :
```

```
'a Manager.t ->
```

```
'a t ->
```

```
Var.t array -> Texpr1.t array -> 'a t option -> 'a t
```

Parallel assignment of an array of dimensions by an array of same size of tree expressions

```
val substitute_expr_array :
  'a Manager.t ->
  'a t ->
  Var.t array -> Texpr1.t array -> 'a t option -> 'a t
```

Parallel substitution of an array of dimensions by an array of same size of tree expressions

18.3.2.0.2 Side-effect versions of the previous functions

```
val assign_linexpr_array_with :
  'a Manager.t ->
  'a t ->
  Var.t array -> Linexpr1.t array -> 'a t option -> unit

val substitute_linexpr_array_with :
  'a Manager.t ->
  'a t ->
  Var.t array -> Linexpr1.t array -> 'a t option -> unit

val assign_expr_array_with :
  'a Manager.t ->
  'a t ->
  Var.t array -> Texpr1.t array -> 'a t option -> unit

val substitute_expr_array_with :
  'a Manager.t ->
  'a t ->
  Var.t array -> Texpr1.t array -> 'a t option -> unit
```

18.3.3 Projections

These functions implements forgetting (existential quantification) of (array of) variables. Both functional and side-effect versions are provided. The Boolean, if true, adds a projection onto 0-plane.

```
val forget_array : 'a Manager.t -> 'a t -> Var.t array -> bool -> 'a t
val forget_array_with : 'a Manager.t -> 'a t -> Var.t array -> bool -> unit
```

18.3.4 Change and permutation of dimensions

```
val change_environment :
  'a Manager.t -> 'a t -> Environment.t -> bool -> 'a t
```

Change the environnement of the abstract values.

Variables that are removed are first existentially quantified, and variables that are introduced are unconstrained. The Boolean, if true, adds a projection onto 0-plane for these ones.

```
val minimize_environment : 'a Manager.t -> 'a t -> 'a t
```

Remove from the environment of the abstract value and from the abstract value itself variables that are unconstrained in it.

```
val rename_array :
  'a Manager.t ->
  'a t -> Var.t array -> Var.t array -> 'a t
```

Parallel renaming of the environment of the abstract value.

The new variables should not interfere with the variables that are not renamed.

```

val change_environment_with :
  'a Manager.t -> 'a t -> Environment.t -> bool -> unit
val minimize_environment_with : 'a Manager.t -> 'a t -> unit
val rename_array_with :
  'a Manager.t -> 'a t -> Var.t array -> Var.t array -> unit

```

18.3.5 Expansion and folding of dimensions

These functions allows to expand one dimension into several ones having the same properties with respect to the other dimensions, and to fold several dimensions into one. Formally,

- expand $P(x,y,z) \ z \ w = P(x,y,z)$ inter $P(x,y,w)$ if z is expanded in z and w
- fold $Q(x,y,z,w) \ z \ w = \text{exists } w:Q(x,y,z,w) \text{ union } (\text{exist } z:Q(x,y,z,w))(z \leftarrow w)$ if z and w are folded onto z

```

val expand : 'a Manager.t -> 'a t -> Var.t -> Var.t array -> 'a t

```

Expansion: `expand a var tvar` expands the variable `var` into itself and the additional variables in `tvar`, which are given the same type as `var`.

It results in $(n+1)$ unrelated variables having same relations with other variables. The additional variables are added to the environment of the argument for making the environment of the result, so they should not belong to the initial environnement.

```

val fold : 'a Manager.t -> 'a t -> Var.t array -> 'a t

```

Folding: `fold a tvar` fold the variables in the array `tvar` of size $n \geq 1$ and put the result in the first variable of the array. The other variables of the array are then removed, both from the environment and the abstract value.

```

val expand_with : 'a Manager.t -> 'a t -> Var.t -> Var.t array -> unit

```

```

val fold_with : 'a Manager.t -> 'a t -> Var.t array -> unit

```

18.3.6 Widening

```

val widening : 'a Manager.t -> 'a t -> 'a t -> 'a t

```

Widening

```

val widening_threshold :
  'a Manager.t ->
  'a t -> 'a t -> Lincons1.earray -> 'a t

```

18.3.7 Closure operation

```

val closure : 'a Manager.t -> 'a t -> 'a t

```

Closure: transform strict constraints into non-strict ones.

```

val closure_with : 'a Manager.t -> 'a t -> unit

```

Side-effect version

18.4 Additional operations

```

val of_lincons_array :
  'a Manager.t -> Environment.t -> Lincons1.earray -> 'a t
val of_tcons_array : 'a Manager.t -> Environment.t -> Tcons1.earray -> 'a t
  Abstract a conjunction of constraints

val assign_linexpr :
  'a Manager.t ->
  'a t ->
  Var.t -> Linexpr1.t -> 'a t option -> 'a t
val substitute_linexpr :
  'a Manager.t ->
  'a t ->
  Var.t -> Linexpr1.t -> 'a t option -> 'a t
val assign_texpr :
  'a Manager.t ->
  'a t ->
  Var.t -> Texpr1.t -> 'a t option -> 'a t
val substitute_texpr :
  'a Manager.t ->
  'a t ->
  Var.t -> Texpr1.t -> 'a t option -> 'a t
  Assignement/Substitution of a single dimension by a single expression

val assign_linexpr_with :
  'a Manager.t ->
  'a t -> Var.t -> Linexpr1.t -> 'a t option -> unit
val substitute_linexpr_with :
  'a Manager.t ->
  'a t -> Var.t -> Linexpr1.t -> 'a t option -> unit
val assign_texpr_with :
  'a Manager.t ->
  'a t -> Var.t -> Texpr1.t -> 'a t option -> unit
val substitute_texpr_with :
  'a Manager.t ->
  'a t -> Var.t -> Texpr1.t -> 'a t option -> unit
  Side-effect version of the previous functions

val unify : 'a Manager.t -> 'a t -> 'a t -> 'a t
  Unification of 2 abstract values on their least common environment

val unify_with : 'a Manager.t -> 'a t -> 'a t -> unit
  Side-effect version

```

Chapter 19

Module Parser : APRON Parsing of expressions

19.1 Introduction

This small module implements the parsing of expressions, constraints and generators. The allowed syntax is simple (no parenthesis) but supports interval expressions.

```
cons ::= expr ('>' | '>=' | '=' | '!=' | '<' | '<=' | '<') expr | expr = expr 'mod' scalar
gen ::= ('V:' | 'R:' | 'L:' | 'RM:' | 'LM:') expr
expr ::= expr '+' term | expr '-' term | term
term ::= coeff ['*'] identifier | coeff | ['-'] identifier
coeff ::= scalar | ['-'] '[' scalar ';' scalar ']'
scalar ::= ['-'] (integer | rational | floating_point_number)
```

There is the possibility to parse directly from a lexing buffer, or from a string (from which one can generate a buffer with the function `Lexing.from_string`).

This module uses the underlying modules `Apron_lexer` and `Apron_parser`.

19.2 Interface

exception Error of string

 Raised by conversion functions

```
val linexpr1_of_lexbuf : Environment.t -> Lexing.lexbuf -> Linexpr1.t
```

```
val lincons1_of_lexbuf : Environment.t -> Lexing.lexbuf -> Lincons1.t
```

```
val generator1_of_lexbuf : Environment.t -> Lexing.lexbuf -> Generator1.t
```

 Conversion from lexing buffers to resp. linear expressions, linear constraints and generators, defined on the given environment.

```
val texpr1expr_of_lexbuf : Lexing.lexbuf -> Texpr1.expr
```

```
val texpr1_of_lexbuf : Environment.t -> Lexing.lexbuf -> Texpr1.t
```

```
val tcons1_of_lexbuf : Environment.t -> Lexing.lexbuf -> Tcons1.t
```

 Conversion from lexing buffers to resp. tree expressions and constraints, defined on the given environment.

```
val linexpr1_of_string : Environment.t -> string -> Linexpr1.t
```

```

val lincons1_of_string : Environment.t -> string -> Lincons1.t
val generator1_of_string : Environment.t -> string -> Generator1.t
    Conversion from strings to resp. linear expressions, linear constraints and generators, defined on
    the given environment.

val texpr1expr_of_string : string -> Texpr1.expr
val texpr1_of_string : Environment.t -> string -> Texpr1.t
val tcons1_of_string : Environment.t -> string -> Tcons1.t
    Conversion from lexing buffers to resp. tree expressions and constraints, defined on the given
    environment.

val lincons1_of_lstring : Environment.t -> string list -> Lincons1.earray
val generator1_of_lstring : Environment.t -> string list -> Generator1.earray
    Conversion from lists of strings to array of resp. linear constraints and generators, defined on the
    given environment.

val tcons1_of_lstring : Environment.t -> string list -> Tcons1.earray
    Conversion from lists of strings to array of tree constraints.

val of_lstring :
  'a Manager.t -> Environment.t -> string list -> 'a Abstract1.t
    Abstraction of lists of strings representing constraints to abstract values, on the abstract domain
    defined by the given manager.

```


Part IV

Level 0 of the interface

Chapter 20

Module Dim : APRON Dimensions and related types

```
type t = int
type change = {
  dim : int array ;
  intdim : int ;
  realdim : int ;
}
type perm = int array
type dimension = {
  intd : int ;
  reald : int ;
}
APRON Dimensions and related types
```

Chapter 21

Module Linexpr0 : APRON Linear expressions of level 0

```
type t
APRON Linear expressions of level 0
val make : int option -> t
    Create a linear expression. Its representation is sparse if None is provided, dense of size size if
    Some size is provided.
val minimize : t -> unit
    In case of sparse representation, remove zero coefficients
val copy : t -> t
    Copy
val compare : t -> t -> int
    Comparison with lexicographic ordering using Coeff.cmp, terminating by constant
val hash : t -> int
    Hashing function
val get_size : t -> int
    Get the size of the linear expression (which may be sparse or dense)
val get_cst : t -> Coeff.t
    Get the constant
val get_coeff : t -> int -> Coeff.t
    Get the coefficient corresponding to the dimension
val set_list : t -> (Coeff.t * Dim.t) list -> Coeff.t option -> unit
    Set simultaneously a number of coefficients.
    set_list expr [(c1,1); (c2,2)] (Some cst) assigns coefficients c1 to dimension 1,
    coefficient c2 to dimension 2, and coefficient cst to the constant. If (Some cst) is replaced by
    None, the constant coefficient is not assigned.
val set_array : t -> (Coeff.t * Dim.t) array -> Coeff.t option -> unit
```

Set simultaneously a number of coefficients, as `set_list`.

```
val set_cst : t -> Coeff.t -> unit
```

Set the constant

```
val set_coeff : t -> int -> Coeff.t -> unit
```

Set the coefficient corresponding to the dimension

Iter the function on the pairs coefficient/dimension of the linear expression

```
val iter : (Coeff.t -> Dim.t -> unit) -> t -> unit
```

```
val print : (Dim.t -> string) -> Format.formatter -> t -> unit
```

Print a linear expression, using a function converting from dimensions to names

Chapter 22

Module Lincons0 : APRON Linear constraints of level 0

```
type t = {
  mutable linexpr0 : Linexpr0.t ;
  mutable typ : typ ;
}

type typ =
| EQ
| SUPEQ
| SUP
| DISEQ
| EQMOD of Scalar.t
  APRON Linear constraints of level 0

val make : Linexpr0.t -> typ -> t
  Make a linear constraint. Modifying later the linear expression modifies correspondingly the
  linear constraint and conversely

val copy : t -> t
  Copy a linear constraint (deep copy)

val string_of_typ : typ -> string
  Convert a constraint type to a string (=,>=, or >)

val print : (Dim.t -> string) -> Format.formatter -> t -> unit
  Print a constraint
```

Chapter 23

Module Generator0 : APRON Generators of level 0

```
type typ =  
  | LINE  
  | RAY  
  | VERTEX  
  | LINEMOD  
  | RAYMOD  
type t = {  
  mutable linexpr0 : Linexpr0.t ;  
  mutable typ : typ ;  
}  
APRON Generators of level 0  
val make : Linexpr0.t -> typ -> t  
    Making a generator. The constant coefficient of the linear expression is ignored. Modifying later  
    the linear expression modifies correspondingly the generator and conversely.  
  
val copy : t -> t  
    Copy a generator (deep copy)  
  
val string_of_typ : typ -> string  
    Convert a generator type to a string (LIN,RAY, or VTX)  
  
val print : (Dim.t -> string) -> Format.formatter -> t -> unit  
    Print a generator
```

Chapter 24

Module Texpr0

```
type t
type unop =
  | Neg
  | Cast
  | Sqrt
    Unary operators

type binop =
  | Add
  | Sub
  | Mul
  | Div
  | Mod
    Binary operators

type typ =
  | Real
  | Int
  | Single
  | Double
  | Extended
  | Quad
    Destination type for rounding

type round =
  | Near
  | Zero
  | Up
  | Down
  | Rnd
    Rounding direction

APRON tree expressions of level 0
type expr =
  | Cst of Coeff.t
  | Dim of Dim.t
```

```
| Unop of unop * expr * typ * round
| Binop of binop * expr * expr * typ * round
  User type for tree expressions
```

24.1 Constructors and Destructor

```
val of_expr : expr -> t
  General constructor (actually the most efficient)

val copy : t -> t
  Copy

val of_linexpr : Linexpr0.t -> t
  Conversion

val to_expr : t -> expr
  General destructor
```

24.1.1 Incremental constructors

```
val cst : Coeff.t -> t
val dim : Dim.t -> t
val unop : unop -> t -> typ -> round -> t
val binop : binop ->
  typ -> round -> t -> t -> t
```

24.2 Tests

```
val is_interval_cst : t -> bool
val is_interval_linear : t -> bool
val is_interval_polynomial : t -> bool
val is_interval_polyfrac : t -> bool
val is_scalar : t -> bool
```

24.3 Printing

```
val string_of_unop : unop -> string
val string_of_binop : binop -> string
val string_of_typ : typ -> string
val string_of_round : round -> string
val print_unop : Format.formatter -> unop -> unit
val print_binop : Format.formatter -> binop -> unit
val print_typ : Format.formatter -> typ -> unit
val print_round : Format.formatter -> round -> unit
val print_expr : (Dim.t -> string) -> Format.formatter -> expr -> unit
  Print a tree expression, using a function converting from dimensions to names

val print : (Dim.t -> string) -> Format.formatter -> t -> unit
  Print an abstract tree expression, using a function converting from dimensions to names
```


24.4 Internal usage for level 1

```
val print_sprint_unop : unop -> typ -> round -> string
val print_sprint_binop : binop -> typ -> round -> string
val print_precedence_of_unop : unop -> int
val print_precedence_of_binop : binop -> int
```

Chapter 25

Module Tcons0 : APRON tree expressions constraints of level 0

```
type t = {  
  mutable texpr0 : Texpr0.t ;  
  mutable typ : Lincons0.typ ;  
}
```

APRON tree expressions constraints of level 0

```
type typ = Lincons0.typ =
```

```
| EQ  
| SUPEQ  
| SUP  
| DISEQ  
| EQMOD of Scalar.t
```

```
val make : Texpr0.t -> typ -> t
```

Make a tree expression constraint. Modifying later the tree expression expression modifies correspondingly the tree expression constraint and conversely

```
val copy : t -> t
```

Copy a tree expression constraint (deep copy)

```
val string_of_typ : typ -> string
```

Convert a constraint type to a string (=,>=, or >)

```
val print : (Dim.t -> string) -> Format.formatter -> t -> unit
```

Print a constraint

Chapter 26

Module Abstract0 : APRON Abstract value of level 0

```
type 'a t
APRON Abstract value of level 0
The type parameter 'a allows to distinguish abstract values with different underlying abstract domains.
val set_gc : int -> unit
    TO BE DOCUMENTED
```

26.1 General management

26.1.1 Memory

```
val copy : 'a Manager.t -> 'a t -> 'a t
    Copy a value

val size : 'a Manager.t -> 'a t -> int
    Return the abstract size of a value
```

26.1.2 Control of internal representation

```
val minimize : 'a Manager.t -> 'a t -> unit
    Minimize the size of the representation of the value. This may result in a later recomputation of
    internal information.

val canonicalize : 'a Manager.t -> 'a t -> unit
    Put the abstract value in canonical form. (not yet clear definition)

val approximate : 'a Manager.t -> 'a t -> int -> unit
    approximate man abs alg perform some transformation on the abstract value, guided by the
    argument alg. The transformation may lose information. The argument alg overrides the field
    algorithm of the structure of type Manager.funopt associated to ap_abstract0_approximate
    (commodity feature).
```

26.1.3 Printing

```
val fdump : 'a Manager.t -> 'a t -> unit
```

Dump on the `stdout` C stream the internal representation of an abstract value, for debugging purposes

```
val print : (int -> string) -> Format.formatter -> 'a t -> unit
```

Print as a set of constraints

26.1.4 Serialization

26.2 Constructor, accessors, tests and property extraction

26.2.1 Basic constructors

```
val bottom : 'a Manager.t -> int -> int -> 'a t
```

Create a bottom (empty) value with the given number of integer and real variables

```
val top : 'a Manager.t -> int -> int -> 'a t
```

Create a top (universe) value with the given number of integer and real variables

```
val of_box : 'a Manager.t -> int -> int -> Interval.t array -> 'a t
```

Abstract an hypercube.

`of_box man intdim realdim array` abstracts an hypercube defined by the array of intervals of size `intdim+realdim`

26.2.2 Accessors

```
val dimension : 'a Manager.t -> 'a t -> Dim.dimension
```

```
val manager : 'a t -> 'a Manager.t
```

26.2.3 Tests

```
val is_bottom : 'a Manager.t -> 'a t -> Manager.tbool
```

Emptiness test

```
val is_top : 'a Manager.t -> 'a t -> Manager.tbool
```

Universality test

```
val is_leq : 'a Manager.t -> 'a t -> 'a t -> Manager.tbool
```

Inclusion test. The 2 abstract values should be compatible.

```
val is_eq : 'a Manager.t -> 'a t -> 'a t -> Manager.tbool
```

Equality test. The 2 abstract values should be compatible.

```
val sat_lincons : 'a Manager.t -> 'a t -> Lincons0.t -> Manager.tbool
```

Does the abstract value satisfy the linear constraint ?

```
val sat_tcons : 'a Manager.t -> 'a t -> Tcons0.t -> Manager.tbool
```

Does the abstract value satisfy the tree expression constraint ?

```
val sat_interval :
  'a Manager.t -> 'a t -> Dim.t -> Interval.t -> Manager.tbool
  Does the abstract value satisfy the constraint dim in interval ?

val is_dimension_unconstrained :
  'a Manager.t -> 'a t -> Dim.t -> Manager.tbool
  Is the dimension unconstrained in the abstract value ? If yes, this means that the existential
  quantification of the dimension does not change the value.
```

26.2.4 Extraction of properties

```
val bound_dimension : 'a Manager.t -> 'a t -> Dim.t -> Interval.t
  Return the interval of variation of the dimension in the abstract value.

val bound_linexpr : 'a Manager.t -> 'a t -> Linexpr0.t -> Interval.t
  Return the interval of variation of the linear expression in the abstract value.
  Implement a form of linear programming, where the argument linear expression is the one to
  optimize under the constraints induced by the abstract value.

val bound_texpr : 'a Manager.t -> 'a t -> Texpr0.t -> Interval.t
  Return the interval of variation of the tree expression in the abstract value.

val to_box : 'a Manager.t -> 'a t -> Interval.t array
  Convert the abstract value to an hypercube

val to_lincons_array : 'a Manager.t -> 'a t -> Lincons0.t array
  Convert the abstract value to a conjunction of linear constraints.

val to_tcons_array : 'a Manager.t -> 'a t -> Tcons0.t array
  Convert the abstract value to a conjunction of tree expression constraints.

val to_generator_array : 'a Manager.t -> 'a t -> Generator0.t array
  Convert the abstract value to a set of generators that defines it.
```

26.3 Operations

26.3.1 Meet and Join

```
val meet : 'a Manager.t -> 'a t -> 'a t -> 'a t
  Meet of 2 abstract values.

val meet_array : 'a Manager.t -> 'a t array -> 'a t
  Meet of a non empty array of abstract values.

val meet_lincons_array : 'a Manager.t -> 'a t -> Lincons0.t array -> 'a t
  Meet of an abstract value with an array of linear constraints.

val meet_tcons_array : 'a Manager.t -> 'a t -> Tcons0.t array -> 'a t
  Meet of an abstract value with an array of tree expression constraints.
```

```
val join : 'a Manager.t -> 'a t -> 'a t -> 'a t
```

Join of 2 abstract values.

```
val join_array : 'a Manager.t -> 'a t array -> 'a t
```

Join of a non empty array of abstract values.

```
val add_ray_array : 'a Manager.t -> 'a t -> Generator0.t array -> 'a t
```

Add the array of generators to the abstract value (time elapse operator).

The generators should either lines or rays, not vertices.

26.3.1.0.3 Side-effect versions of the previous functions

```
val meet_with : 'a Manager.t -> 'a t -> 'a t -> unit
```

```
val meet_lincons_array_with :
```

```
'a Manager.t -> 'a t -> Lincons0.t array -> unit
```

```
val meet_tcons_array_with : 'a Manager.t -> 'a t -> Tcons0.t array -> unit
```

```
val join_with : 'a Manager.t -> 'a t -> 'a t -> unit
```

```
val add_ray_array_with : 'a Manager.t -> 'a t -> Generator0.t array -> unit
```

26.3.2 Assignements and Substitutions

```
val assign_linexpr_array :
```

```
'a Manager.t ->
```

```
'a t ->
```

```
Dim.t array -> Linexpr0.t array -> 'a t option -> 'a t
```

Parallel assignement of an array of dimensions by an array of same size of linear expressions

```
val substitute_linexpr_array :
```

```
'a Manager.t ->
```

```
'a t ->
```

```
Dim.t array -> Linexpr0.t array -> 'a t option -> 'a t
```

Parallel substitution of an array of dimensions by an array of same size of linear expressions

```
val assign_texpr_array :
```

```
'a Manager.t ->
```

```
'a t ->
```

```
Dim.t array -> Texpr0.t array -> 'a t option -> 'a t
```

Parallel assignement of an array of dimensions by an array of same size of tree expressions

```
val substitute_texpr_array :
```

```
'a Manager.t ->
```

```
'a t ->
```

```
Dim.t array -> Texpr0.t array -> 'a t option -> 'a t
```

Parallel substitution of an array of dimensions by an array of same size of tree expressions

26.3.2.0.4 Side-effect versions of the previous functions

```
val assign_linexpr_array_with :
```

```
'a Manager.t ->
```

```
'a t ->
```

```
Dim.t array -> Linexpr0.t array -> 'a t option -> unit
```

```
val substitute_linexpr_array_with :
```

```

'a Manager.t ->
'a t ->
Dim.t array -> Linexpr0.t array -> 'a t option -> unit
val assign_texpr_array_with :
'a Manager.t ->
'a t ->
Dim.t array -> Texpr0.t array -> 'a t option -> unit
val substitute_texpr_array_with :
'a Manager.t ->
'a t ->
Dim.t array -> Texpr0.t array -> 'a t option -> unit

```

26.3.3 Projections

These functions implements forgetting (existential quantification) of (array of) dimensions. Both functional and side-effect versions are provided. The Boolean, if true, adds a projection onto 0-plane.

```

val forget_array : 'a Manager.t -> 'a t -> Dim.t array -> bool -> 'a t
val forget_array_with : 'a Manager.t -> 'a t -> Dim.t array -> bool -> unit

```

26.3.4 Change and permutation of dimensions

```

val add_dimensions : 'a Manager.t -> 'a t -> Dim.change -> bool -> 'a t
val remove_dimensions : 'a Manager.t -> 'a t -> Dim.change -> 'a t
val permute_dimensions : 'a Manager.t -> 'a t -> Dim.perm option -> 'a t

```

26.3.4.0.5 Side-effect versions of the previous functions

```

val add_dimensions_with : 'a Manager.t -> 'a t -> Dim.change -> bool -> unit
val remove_dimensions_with : 'a Manager.t -> 'a t -> Dim.change -> unit
val permute_dimensions_with : 'a Manager.t -> 'a t -> Dim.perm option -> unit

```

26.3.5 Expansion and folding of dimensions

These functions allows to expand one dimension into several ones having the same properties with respect to the other dimensions, and to fold several dimensions into one. Formally,

- expand $P(x,y,z) \ z \ w = P(x,y,z) \text{ inter } P(x,y,w)$ if z is expanded in z and w
- fold $Q(x,y,z,w) \ z \ w = \text{exists } w:Q(x,y,z,w) \text{ union } (\text{exist } z:Q(x,y,z,w))(z \leftarrow w)$ if z and w are folded onto z

```

val expand : 'a Manager.t -> 'a t -> Dim.t -> int -> 'a t

```

Expansion: `expand a dim n` expands the dimension `dim` into itself + `n` additional dimensions. It results in $(n+1)$ unrelated dimensions having same relations with other dimensions. The $(n+1)$ dimensions are put as follows:

- original dimension `dim`
- if the dimension is integer, the `n` additional dimensions are put at the end of integer dimensions; if it is real, at the end of the real dimensions.

```

val fold : 'a Manager.t -> 'a t -> Dim.t array -> 'a t

```

Folding: `fold a tdim` fold the dimensions in the array `tdim` of size $n \geq 1$ and put the result in the first dimension of the array. The other dimensions of the array are then removed (using `ap_abstract0_permute_remove_dimensions`).

```

val expand_with : 'a Manager.t -> 'a t -> Dim.t -> int -> unit
val fold_with : 'a Manager.t -> 'a t -> Dim.t array -> unit

```

26.3.6 Widening

```

val widening : 'a Manager.t -> 'a t -> 'a t -> 'a t
    Widening

val widening_threshold :
    'a Manager.t ->
    'a t -> 'a t -> Lincons0.t array -> 'a t

```

26.3.7 Closure operation

```

val closure : 'a Manager.t -> 'a t -> 'a t
    Closure: transform strict constraints into non-strict ones.

val closure_with : 'a Manager.t -> 'a t -> unit
    Side-effect version

```

26.4 Additional operations

```

val of_lincons_array : 'a Manager.t -> int -> int -> Lincons0.t array -> 'a t
val of_tcons_array : 'a Manager.t -> int -> int -> Tcons0.t array -> 'a t
    Abstract a conjunction of constraints

```

```

val assign_linexpr :
    'a Manager.t ->
    'a t ->
    Dim.t -> Linexpr0.t -> 'a t option -> 'a t
val substitute_linexpr :
    'a Manager.t ->
    'a t ->
    Dim.t -> Linexpr0.t -> 'a t option -> 'a t
val assign_texpr :
    'a Manager.t ->
    'a t ->
    Dim.t -> Texpr0.t -> 'a t option -> 'a t
val substitute_texpr :
    'a Manager.t ->
    'a t ->
    Dim.t -> Texpr0.t -> 'a t option -> 'a t
    Assignment/Substitution of a single dimension by a single expression

```

```

val assign_linexpr_with :
    'a Manager.t ->
    'a t -> Dim.t -> Linexpr0.t -> 'a t option -> unit
val substitute_linexpr_with :
    'a Manager.t ->
    'a t -> Dim.t -> Linexpr0.t -> 'a t option -> unit
val assign_texpr_with :

```



```
'a Manager.t ->
'a t -> Dim.t -> Texpr0.t -> 'a t option -> unit
val substitute_texpr_with :
'a Manager.t ->
'a t -> Dim.t -> Texpr0.t -> 'a t option -> unit
    Side-effect version of the previous functions

val print_array :
?first:(unit, Format.formatter, unit) Pervasives.format ->
?sep:(unit, Format.formatter, unit) Pervasives.format ->
?last:(unit, Format.formatter, unit) Pervasives.format ->
(Format.formatter -> 'a -> unit) -> Format.formatter -> 'a array -> unit
    General use
```

Part V

MLGmpIDL modules

Chapter 27

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`[32].

27.1 Pretty printing

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

27.2 Initialization Functions

```
val init : unit -> t
```

```
val init2 : int -> t
```

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

27.3 Assignment Functions

The first parameter holds the result.

```
val set : t -> t -> unit
```

```
val set_si : t -> int -> unit
```

```
val set_d : t -> float -> unit
```

```
val _set_str : t -> string -> int -> unit
```

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

```
val swap : t -> t -> unit
```

27.4 Combined Initialization and Assignment Functions

```
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
```

27.5 Conversion Functions

```
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
```

27.6 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
```

27.7 Arithmetic Functions

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
```

27.8 Division Functions

`c` stands for ceiling, `f` for floor, and `t` for truncate (rounds toward 0).

27.8.1 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.

27.8.2 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
```

27.8.3 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
```

27.8.4 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

```

27.9 Exponentiation Functions

```

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

```

27.10 Root Extraction Functions

```

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

```

27.11 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
```

27.12 Comparison Functions

```
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
```

27.13 Logical and Bit Manipulation Functions

```
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
```

27.14 Input and Output Functions: not interfaced

27.15 Random Number Functions: see `Gmp_random`[31] module

27.16 Integer Import and Export Functions

```
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
```

27.17 Miscellaneous Functions

```
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 28

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`[33].

`val canonicalize : t -> unit`

28.1 Pretty printing

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

28.2 Initialization and Assignment Functions

`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`

28.3 Additional Initialization and Assignments 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`

28.4 Conversion Functions

```
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
```

28.5 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
val of_frac : int -> int -> t
val of_mpz : Mpz.t -> t
val of_mpz2 : Mpz.t -> Mpz.t -> t
```

28.6 Arithmetic Functions

```
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
```

28.7 Comparison Functions

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

28.8 Applying Integer Functions to Rationals

```
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
```

28.9 Input and Output Functions: not interfaced

Chapter 29

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.

29.1 Pretty printing

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

29.2 Initialization and Assignment 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
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
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
```

29.3 Conversion Functions

```
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
```

29.4 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
val of_mpz : Mpz.t -> t
val of_mpq : Mpq.t -> t
val is_integer : t -> bool
```

29.5 Arithmetic Functions

```
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
```

29.6 Comparison Functions

```
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
```

29.7 Input and Output Functions: not interfaced

29.8 Random Number Functions: see `Gmp_random`[31] module

29.9 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 30

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.

30.1 Pretty printing

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

30.2 Rounding Modes

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

30.3 Exceptions

```
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

```

30.4 Initialization and Assignment 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
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_inf : t -> int -> unit
val set_nan : t -> unit
val swap : t -> t -> unit
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

```

30.5 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

```


30.6 User Conversions

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

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

30.7 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
```

30.8 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

```

30.9 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
```

30.10 Input and Output Functions: not interfaced

30.11 Miscellaneous Float 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
```

Chapter 31

Module Gmp_random : GMP random generation functions

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

31.1 Random State Initialization

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

31.2 Random State Seeding

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

31.3 Random Number Functions

```
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

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

Chapter 32

Module Mpzf : GMP multi-precision integers, functional version

Functions in this module has a functional semantics, unlike the corresponding functions in `Mpz`[27]. These functions are less efficient, due to the additional memory allocation needed 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
```

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
```

Conversion from and to `Mpz.t`.

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

32.1 Constructors

```
val of_string : string -> t
```

```
val of_float : float -> t
```

```
val of_int : int -> t
```

32.2 Conversions and Printing

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

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

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

32.3 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
```

32.4 Comparison Functions

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

Chapter 33

Module Mpqf : GMP multi-precision rationals, functional version

Functions in this module has a functional semantics, unlike the corresponding functions in `Mpq`[28]. These functions are less efficient, due to the additional memory allocation needed for the result.

`type t`

multi-precision rationals

`val to_mpq : t -> Mpq.t`

`val of_mpq : Mpq.t -> t`

`val of_mpz : Mpz.t -> t`

`val of_mpz2 : Mpz.t -> Mpz.t -> t`

Conversion from and to `Mpz.t` and `Mpz.t`.

There is no sharing between the argument and the result.

33.1 Constructors

`val of_string : string -> t`

`val of_float : float -> t`

`val of_int : int -> t`

`val of_frac : int -> int -> t`

`val of_mpzf : Mpzf.t -> t`

`val of_mpzf2 : Mpzf.t -> Mpzf.t -> t`

33.2 Conversions and Printing

`val to_string : t -> string`

`val to_float : t -> float`

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

33.3 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
```

33.4 Comparison Functions

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

33.5 Extraction Functions

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


Index

`_equal`, 83, 87
`_export`, 77
`_gcdext`, 75
`_get_str`, 73, 79, 82, 85
`_import`, 76
`_init_set_str`, 73, 82, 85
`_powm`, 75
`_powm_ui`, 75
`_set_str`, 72, 78, 81, 85
`_sqrtrem`, 75

`abs`, 73, 79, 82, 86, 91, 93
`Abstract0`, 64
`abstract0`, 46
`Abstract1`, 45
`acos`, 87
`acosh`, 87
`add`, 30, 73, 79, 82, 86, 91, 92
`add_dimensions`, 68
`add_dimensions_with`, 68
`add_epsilon`, 20
`add_epsilon_bin`, 21
`add_int`, 91
`add_q`, 86
`add_ray_array`, 48, 67
`add_ray_array_with`, 48, 67
`add_ui`, 73, 82, 86
`add_z`, 86
`addmul`, 73
`addmul_ui`, 73
`agm`, 88
`approximate`, 45, 64
`array_extend_environment`, 36, 38, 44
`array_extend_environment_with`, 36, 38, 44
`array_get`, 36, 38, 44
`array_get_env`, 36, 44
`array_length`, 36, 38, 44
`array_make`, 36, 38, 44
`array_print`, 36, 38, 44
`array_set`, 36, 38, 44
`asin`, 87
`asinh`, 87
`assign_linexpr`, 51, 69
`assign_linexpr_array`, 48, 67
`assign_linexpr_array_with`, 49, 67
`assign_linexpr_with`, 51, 69
`assign_texpr`, 51, 69

`assign_texpr_array`, 48, 67
`assign_texpr_array_with`, 49, 68
`assign_texpr_with`, 51, 70
`atan`, 87
`atan2`, 87
`atanh`, 87

`bin_ui`, 76
`bin_uiui`, 76
`binop`, 40, 41, 60, 61
`bottom`, 11, 46, 65
`bound_dimension`, 66
`bound_linexpr`, 47, 66
`bound_texpr`, 47, 66
`bound_variable`, 47
`Box`, 18
`box1`, 45

`canonicalize`, 45, 64, 78
`cdiv_q`, 74, 91
`cdiv_q_2exp`, 74
`cdiv_q_ui`, 74
`cdiv_qr`, 74, 91
`cdiv_qr_ui`, 74
`cdiv_r`, 74, 91
`cdiv_r_2exp`, 74
`cdiv_r_ui`, 74
`cdiv_ui`, 74
`ceil`, 83, 88
`change`, 55
`change_environment`, 49
`change_environment_with`, 50
`check_range`, 84
`clear_flags`, 85
`clear_inexflag`, 85
`clear_nanflag`, 85
`clear_overflow`, 85
`clear_underflow`, 84
`closure`, 50, 69
`closure_with`, 50, 69
`clrbt`, 76
`cmp`, 8, 11, 13, 76, 79, 83, 86, 91, 93
`cmp_d`, 76, 83
`cmp_frac`, 93
`cmp_int`, 8, 91, 93
`cmp_si`, 76, 79, 83, 86
`cmp_si_2exp`, 86

cmpabs, 76
 cmpabs_d, 76
 cmpabs_ui, 76
 Coeff, 12
 com, 76
 compare, 29, 30, 56
 congruent_2exp_p, 75
 congruent_p, 75
 congruent_ui_p, 75
 const_catalan, 88
 const_euler, 88
 const_log2, 88
 const_pi, 88
 copy, 32, 34, 37, 41, 43, 45, 56, 58, 59, 61, 63, 64
 cos, 87
 cosh, 87
 cot, 87
 coth, 87
 csc, 87
 csch, 87
 cst, 41, 61

 Dim, 55
 dim, 61
 dim_of_var, 31
 dimension, 30, 55, 65
 div, 79, 82, 86, 93
 div_2exp, 79, 82, 86
 div_2si, 86
 div_2ui, 86
 div_q, 86
 div_ui, 82, 86
 div_z, 86
 divexact, 75, 91
 divexact_ui, 75
 divisible_2exp_p, 75
 divisible_p, 75
 divisible_ui_p, 75

 earray, 34, 37, 43
 eint, 87
 env, 46
 Environment, 30
 equal, 9, 11, 13, 30, 79, 83, 87, 93
 equal_int, 9
 equalities, 22
 erf, 87
 erfc, 87
 Error, 17, 52
 even_p, 77
 exc, 16
 exclog, 16
 exp, 87
 exp10, 87
 exp2, 87
 expand, 50, 68
 expand_with, 50, 69
 expm1, 87
 export, 77
 expr, 41, 61
 extend_environment, 33, 35, 38, 41, 44
 extend_environment_with, 33, 35, 38, 41, 44

 fac_ui, 76, 87
 fdiv_q, 74, 91
 fdiv_q_2exp, 74
 fdiv_q_ui, 74
 fdiv_qr, 74, 91
 fdiv_qr_ui, 74
 fdiv_r, 74, 91
 fdiv_r_2exp, 74
 fdiv_r_ui, 74
 fdiv_ui, 74
 fdump, 46, 65
 fib_ui, 76
 fib2_ui, 76
 fits_int_p, 77, 83
 fits_sint_p, 77, 83
 fits_slong_p, 77, 83
 fits_sshort_p, 77, 83
 fits_uint_p, 77, 83
 fits_ulong_p, 77, 83
 fits_ushort_p, 77, 83
 floor, 83, 88
 fma, 88
 fold, 50, 68
 fold_with, 50, 69
 forget_array, 49, 68
 forget_array_with, 49, 68
 funid, 16
 funopt, 16
 funopt_make, 17

 gamma, 87
 gand, 76
 gcd, 75, 91
 gcd_ui, 75
 gcdext, 75
 Generator0, 59
 Generator1, 37
 generator1_of_lexbuf, 52
 generator1_of_lstring, 53
 generator1_of_string, 53
 get_approximate_max_coeff_size, 23
 get_coeff, 33, 35, 38, 56
 get_cst, 32, 35, 56
 get_d, 73, 79, 82, 85
 get_d_2exp, 73, 82
 get_d1, 85
 get_default_prec, 81, 85
 get_den, 79, 93
 get_deserialize, 17

get_emax, 84
 get_emin, 84
 get_env, 33, 35, 39, 42, 44
 get_flag_best, 17
 get_flag_exact, 17
 get_funopt, 17
 get_generator0, 39
 get_int, 73, 82
 get_library, 16
 get_lincons0, 35
 get_linexpr0, 33
 get_linexpr1, 35, 39
 get_max_coeff_size, 23
 get_num, 79, 93
 get_prec, 81, 85
 get_q, 82
 get_si, 73, 82
 get_size, 56
 get_str, 73, 79, 82, 85
 get_tcons0, 44
 get_texpr0, 41
 get_texpr1, 44
 get_typ, 34, 37, 43
 get_version, 17
 get_z, 79, 82, 85
 get_z_exp, 85
 gmod, 75, 91
 gmod_ui, 75
 Gmp_random, 89
 grid, 24

 hamdist, 76
 hash, 29, 56
 hypot, 88

 i_of_float, 12
 i_of_frac, 12
 i_of_int, 12
 i_of_mpq, 12
 i_of_mpqf, 12
 i_of_scalar, 12
 import, 77
 inxflag_p, 85
 inf_p, 87
 init, 72, 78, 81, 85
 init_default, 89
 init_lc_2exp, 89
 init_lc_2exp_size, 89
 init_set, 72, 78, 81, 85
 init_set_d, 73, 78, 82, 85
 init_set_f, 85
 init_set_q, 85
 init_set_si, 73, 78, 81, 85
 init_set_str, 73, 78, 82, 85
 init_set_z, 78, 85
 init2, 72, 81, 85

 integer_p, 83, 88
 internal, 20, 22
 Interval, 10
 inv, 79, 93
 invert, 75
 ior, 76
 is_bottom, 10, 46, 65
 is_dimension_unconstrained, 66
 is_eq, 47, 65
 is_infty, 8
 is_integer, 33, 82
 is_interval, 13
 is_interval_cst, 41, 61
 is_interval_linear, 41, 61
 is_interval_polyfrac, 41, 61
 is_interval_polynomial, 41, 61
 is_leq, 10, 46, 65
 is_real, 33
 is_scalar, 13, 41, 61
 is_top, 10, 46, 65
 is_unsat, 35
 is_variable_unconstrained, 47
 is_zero, 11, 13
 iter, 32, 34, 37, 57

 jacobi, 75
 join, 48, 67
 join_array, 48, 67
 join_with, 48, 67

 kronecker, 75
 kronecker_si, 75

 lce, 30
 lcm, 75, 91
 lcm_ui, 75
 legendre, 75
 Lincons0, 58
 Lincons1, 34
 lincons1_of_lexbuf, 52
 lincons1_of_lstring, 53
 lincons1_of_string, 53
 Linexpr0, 56
 Linexpr1, 32
 linexpr1_of_lexbuf, 52
 linexpr1_of_string, 52
 lngamma, 87
 log, 87
 log10, 87
 log1p, 87
 log2, 87
 loose, 22, 24
 lucnum_ui, 76
 lucnum2_ui, 76

 make, 30, 32, 34, 37, 43, 56, 58, 59, 63

-
- make_unsat, 35
 - Manager, 15
 - manager, 46, 65
 - manager_alloc, 18, 20
 - manager_alloc_equalities, 23
 - manager_alloc_grid, 24
 - manager_alloc_loose, 22, 24, 26
 - manager_alloc_strict, 22, 24, 26
 - manager_get_internal, 20, 23
 - meet, 48, 66
 - meet_array, 48, 66
 - meet_lincons_array, 48, 66
 - meet_lincons_array_with, 48, 67
 - meet_tcons_array, 48, 66
 - meet_tcons_array_with, 48, 67
 - meet_with, 48, 67
 - mem_var, 31
 - minimize, 32, 45, 56, 64
 - minimize_environment, 49
 - minimize_environment_with, 50
 - Mpf, 81, 89
 - Mpfr, 84
 - Mpq, 78
 - Mpqf, 92
 - Mpz, 72, 89
 - mpz, 90
 - Mpzf, 90
 - mpzf, 90
 - mul, 73, 79, 82, 86, 91, 93
 - mul_2exp, 73, 79, 82, 86
 - mul_2si, 86
 - mul_2ui, 86
 - mul_int, 91
 - mul_q, 86
 - mul_si, 73
 - mul_ui, 82, 86
 - mul_z, 86

 - nan_p, 87
 - nanflag_p, 85
 - narrowing, 20
 - neg, 9, 11, 13, 73, 79, 82, 86, 91, 93
 - nextprime, 75
 - number_p, 87

 - Oct, 20
 - odd_p, 77
 - of_box, 46, 65
 - of_expr, 41, 61
 - of_float, 8, 10, 73, 79, 82, 90, 92
 - of_frac, 8, 10, 79, 92
 - of_generator_array, 20
 - of_infsup, 10
 - of_infty, 8
 - of_int, 8, 10, 73, 79, 82, 90, 92
 - of_lincons_array, 51, 69
 - of_linexpr, 41, 61
 - of_lstring, 53
 - of_mpq, 8, 10, 82, 92
 - of_mpqf, 8, 10
 - of_mpz, 79, 82, 90, 92
 - of_mpz2, 79, 92
 - of_mpzf, 92
 - of_mpzf2, 92
 - of_scalar, 10
 - of_string, 29, 73, 79, 82, 90, 92
 - of_tcons_array, 51, 69
 - overflow_p, 85

 - Parser, 52
 - perfect_power_p, 75
 - perfect_square_p, 75
 - perm, 55
 - permute_dimensions, 68
 - permute_dimensions_with, 68
 - Polka, 22
 - PolkaGrid, 26
 - popcount, 76
 - pow, 86
 - pow_si, 86
 - pow_ui, 75, 82, 86
 - powm, 75
 - powm_ui, 75
 - Ppl, 24
 - pre_widening, 21
 - print, 9, 11, 13, 29, 31, 32, 34, 37, 42, 43, 46, 57–59, 61, 63, 65, 72, 78, 81, 84, 90, 92
 - print_array, 70
 - print_binop, 42, 61
 - print_exc, 17
 - print_exclog, 17
 - print_expr, 42, 61
 - print_funid, 17
 - print_funopt, 17
 - print_precedence_of_binop, 62
 - print_precedence_of_unop, 62
 - print_round, 42, 61, 84
 - print_sprint_binop, 62
 - print_sprint_unop, 62
 - print_tbool, 17
 - print_typ, 42, 61
 - print_unop, 42, 61
 - probab_prime_p, 75

 - realloc2, 72
 - reduce, 13
 - reldiff, 83, 87
 - remove, 30, 76
 - remove_dimensions, 68
 - remove_dimensions_with, 68
 - rename_array, 49
 - rename_array_with, 50

-
- rint, 88
 - root, 75
 - round, 40, 60, 84, 88
 - round_prec, 84
 - rrandomb, 89
 -
 - s_of_float, 12
 - s_of_frac, 12
 - s_of_int, 12
 - s_of_mpq, 12
 - s_of_mpqf, 12
 - sat_interval, 47, 66
 - sat_lincons, 47, 65
 - sat_tcons, 47, 65
 - Scalar, 8
 - scan0, 76
 - scan1, 76
 - sec, 87
 - sech, 87
 - seed, 89
 - seed_ui, 89
 - set, 72, 78, 81, 85
 - set_approximate_max_coeff_size, 23
 - set_array, 32, 35, 38, 56
 - set_bottom, 11
 - set_coeff, 33, 35, 38, 57
 - set_cst, 32, 35, 57
 - set_d, 72, 79, 81, 85
 - set_default_prec, 81, 85
 - set_default_rounding_mode, 84
 - set_den, 79
 - set_deserialize, 17
 - set_emax, 84
 - set_emin, 84
 - set_f, 85
 - set_funopt, 17
 - set_gc, 64
 - set_inf, 85
 - set_infsup, 11
 - set_list, 32, 35, 38, 56
 - set_max_coeff_size, 23
 - set_nan, 85
 - set_num, 79
 - set_prec, 81, 85
 - set_prec_raw, 81, 85
 - set_q, 81, 85
 - set_si, 72, 78, 81, 85
 - set_str, 72, 78, 81, 85
 - set_top, 11
 - set_typ, 35, 37, 43
 - set_z, 78, 81, 85
 - setbit, 76
 - sgn, 8, 76, 79, 83, 87, 91, 93
 - si_kronecker, 76
 - sin, 87
 - sin_cos, 87
 - sinh, 87
 - size, 31, 45, 64, 77
 - sizeinbase, 77
 - sqrt, 75, 82, 86
 - sqrt_ui, 86
 - sqrtrem, 75
 - state, 89
 - strict, 22, 24
 - string_of_binop, 42, 61
 - string_of_exc, 17
 - string_of_funid, 17
 - string_of_round, 42, 61, 84
 - string_of_tbool, 17
 - string_of_ttyp, 34, 42, 43, 58, 59, 61, 63
 - string_of_unop, 42, 61
 - sub, 73, 79, 82, 86, 91, 93
 - sub_int, 91
 - sub_q, 86
 - sub_ui, 73, 82, 86
 - sub_z, 86
 - submul, 73
 - submul_ui, 73
 - substitute_linexpr, 51, 69
 - substitute_linexpr_array, 48, 67
 - substitute_linexpr_array_with, 49, 68
 - substitute_linexpr_with, 51, 69
 - substitute_texpr, 51, 69
 - substitute_texpr_array, 49, 67
 - substitute_texpr_array_with, 49, 68
 - substitute_texpr_with, 51, 70
 - swap, 72, 78, 81, 85
 -
 - t, 8, 10, 12, 16, 18, 20, 22, 24, 26, 29, 30, 32, 34, 37, 40, 43, 45, 55, 56, 58–60, 63, 64, 72, 78, 81, 84, 90, 92
 -
 - tan, 87
 - tanh, 87
 - tbool, 15
 - Tcons0, 63
 - Tcons1, 43
 - tcons1_of_lexbuf, 52
 - tcons1_of_lstring, 53
 - tcons1_of_string, 53
 - tdiv_q, 74, 91
 - tdiv_q_2exp, 74
 - tdiv_q_ui, 74
 - tdiv_qr, 74, 91
 - tdiv_qr_ui, 74
 - tdiv_r, 74, 91
 - tdiv_r_2exp, 74
 - tdiv_r_ui, 74
 - tdiv_ui, 74
 - Texpr0, 60
 - Texpr1, 40
 - texpr1_of_lexbuf, 52
 - texpr1_of_string, 53

texpr1expr_of_lexbuf, 52
texpr1expr_of_string, 53
to_box, 47, 66
to_expr, 41, 61
to_float, 73, 79, 82, 90, 92
to_generator_array, 47, 66
to_lincons_array, 47, 66
to_mpq, 92
to_mpz, 90
to_string, 9, 29, 73, 79, 82, 86, 90, 92
to_tcons_array, 47, 66
top, 11, 46, 65
trunc, 83, 88
tstbit, 76
typ, 34, 37, 40, 43, 58–60, 63
typ_of_var, 31
typvar, 30

ui_div, 82, 86
ui_pow, 86
ui_pow_ui, 75, 86
ui_sub, 73, 82, 86
underflow_p, 85
unify, 51
unify_with, 51
union_5, 12
unop, 40, 41, 60, 61
urandomb, 89
urandomm, 89

Var, 29
var, 41
var_of_dim, 31
vars, 31

widening, 50, 69
widening_threshold, 50, 69
widening_thresholds, 20

xor, 76

zeta, 87