APRON

The APRON library
Edition 0.9.8, October 2007

by Bertrand Jeannet and the APRON team

Table of Contents

1 APRON Copying Conditions (LGPL)

[This is the first released version of the Lesser GPL. It also counts
as the successor of the GNU Library Public License, version 2, hence the
version number 2.1.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software—typically libraries—of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the *Lesser* General Public License because it does *Less* to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

   A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

   The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a. The modified work must itself be a software library.

   b. You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

   c. You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

   d. If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

      (For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

   Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

   Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

   This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

   If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

   However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

   When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

   If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

   Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modifi-

cation of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a. Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b. Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

c. Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

d. If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

e. Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

   a. Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

  b. Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

    If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

    It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

    This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser Gen-

eral Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
one line to give the library's name and an idea of what it does.
Copyright (C) year  name of author

This library is free software; you can redistribute it and/or modify it
under the terms of the GNU Lesser General Public License as published by
the Free Software Foundation; either version 2.1 of the License, or (at
your option) any later version.

This library is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the Free Software
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301,
USA.
```

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the library
'Frob' (a library for tweaking knobs) written by James Random Hacker.

signature of Ty Coon, 1 April 1990
Ty Coon, President of Vice
```

That's all there is to it!

2 Introduction to APRON

The APRON library provides a common interface for *abstract domains of invariants* for numerical variables, in the sense of the Abstract Interpretation theory. It includes a few domains, and provides interfaces to libraries implemented by other teams.

Several libraries already exists, wich implement various abstract domains of invariants. One can cite intervals, linear equalities, octagons, octahedra, convex polyhedra, polynomial equalities, polynomial inequalities. Although they offer a kernel of common functionalities, their API may differ greatly, and some functionalities may lack in some libraries. The aim of the APRON library is to offer a common interface to these libraries. Such a standardized interface offers several advantages: it allows

- to easily substitute a library/abstract domain by another in the same analysis tool; this is useful to compare the efficiency of 2 implementations of the same abstract domain, or the precision of 2 different abstract domains.
- to factorize services which are mostly independant of the abstract domain (variables management, linearization of non-linear expressions, etc...);
- to make easier the combination of abstract domains: the abstract domains to be combined are used through the same interface, as the resulting combination;

As a user, why should I use APRON ?

1. it makes very easy to switch the abstract domain (for numerical variables) in use in an analyzer;
2. it already offers the most used abstract domains, ranging from intervals, octagons, convex polyhedra to linear congruences;
3. its interface should satisfy most needs, as it already satisfies the members of the APRON project working in different contexts (verification of high-level specifications/programs with exact arithmetics for INRIA \& Verimag, static analysis of runtime errors with floating-point arithmetics for ENS Paris, automatic parallelization of programs for ENSMP).
4. the interface, at the level 1, already provides slightly higher-level functionalities than most existing and publicy available abstract domains libraries (with the manipulation of environments); this statement should be reinforced in the near future with the planned addition of a generic non-linear expressions layer and a floating-point arithmetic layer.

As a domain implementor, why should I interface my abstract domain/library to APRON ?

1. to incite existing users of the APRON interface to try your library;
2. to make your users, including yourself, benefit from previous points 1 and 4;
3. to not waste your time implementing environments, variables renaming, OCaml interfaces, and so on; the effort to connect your library to the interface should at minimum be counterbalanced by such gains;

3 APRON Rationale and Functionalities

3.1 General choices

Interface levels

There are two main goals for the APRON interface: efficiency of the implementations, and ease of use for the user. In addition, code duplication between libraries should be avoided. As a consequence, two levels were identified:

Level 0 Choices are guided by the efficiency and the precision of the operations;

Level 1 Choices are guided by ease of use, and code factorization.

The level 0 is directly connected to the underlying (existing) library. It includes all the operations that are specific to an abstract domain and whose code cannot be shared. The interface should be minimal, *unless* there is a strong algorithmical advantage to include a combination of more basic operations.

The higher levels offers additional functionalities that are shared by all the library connected to the level 0. For instance:

- managing correspondance between numerical dimensions and names (characters strings or more generally references);
- abstraction of non linear expressions in interval linear expressions;
- automatic call to redimensioning and permutation operations for computing $P(x,y) \sqcap Q(y,z)$

Combination of abstract domain is possible at the level 0. One can implement for instance the cartesian or reduced product of two different abstract domains, the decomposition of abstract values into a product of values of smaller dimensionality, ...

Programming language

The reference version of the interface is the C version of the interface:

- C can be easily interfaced with most programming languages;
- Most of the existing libraries implementing abstract domains for numerical variables are programmed in C or C++.

An OCaml version is already available. The interface between OCaml and C is even generic and any libraries can benefit from it by providing the glue for just one function (see XX).

Compatibility with threads

In order to ensure compatibility with multithreading programming, a context is explicitly passed to functions in order to ensure the following points:

- the transmission of data specific to each library (non-standard options, workspace, ...);
- the transmission of standard options (selection of algorithms and their precision inside a library);
- the management of exceptions (implemented as error codes in the C interface) (`not_implemented`, `invalid_argument`, `overflow`, `timeout`, `out_of_space`).

Interruptions

Interruptions mechanism is have possible for different cases:

`timeout` if the execution time for an operation exceeds some bound;

`out_of_space`
if the space consumption for an operation exceeds some bound;

`overflow` if the magnitude or the space usage of manipulated numbers exceeds some bound;

`not_implemented`
if the operation is actually not implemented by the underlying library;

`invalid_argument`
if the arguments do not follow the requirements of an operation.

> For instance, in a convex polyhedra library, the `out_of_space` exception allows to abort an operation is the result appears to have too many constraints and/or generators. If this happens, one can redo the operation with another (less precise) algorithm. The `overflow` may be useful when effective overflows are encountered with machine integers or when multiprecision rational numbers have too large numerators and denominators. The `not_implemented` exception allows for a library to be linked to the interface even if it does not provide some operation of the interface.

When an interruption occurs, the function should still return a correct result, in the abstract interpretation sense: it should be a correct approximation, usable for next operations in the program. The top value is always a correct approximation.

Memory management

Memory is managed differently depending on the programming language. Currently:

- No automatic garbage collection in the C interface
- Use of the OCaml runtime garbage collector in the OCaml interface

Programming style

Both functional and imperative (i.e., side-effect) signatures are supported for operations. This allows to optimize the memory allocation and to use whichever version is more convenient for an user and the used programming language.

Number representation

Inside a specific library, any number representation may be used (floating-point numbers, machine integers, multiprecision integers/rationals, ...). Existing libraries often offers the possibility to select different representations.

However, in the interface, this representation should be normalized and independent of underlying libraries, without being restrictive either. As a consequence, the interface offers the choiced between

- GMP multiprecision rationals (which implements exact arithmetic);
- and machine floating-point numbers (`double`).

3.2 Functionalities of the interface at level 0

Representation of an abstract value

At the level 0 of the interface, an abstract value is a structure

```
struct ap_abstract0_t {
  ap_manager_t *manager; /* Explicit context */
```

```
  void          *value;   /* Abstract value representation
                              (only known by the underlying library) */
}
```

The context is allocated by the underlying library, and contains an array of function pointers pointing to the function of the underlying library. Hence, it indicates the effective type of an abstract value.

The validity of the arguments of the functions called through the interface is checked before the call to effective functions. In case of problem, an `invalid_argument` exception is raised.

Semantics of an abstract value

The semantics of an abstract value is a subset

$$X \subseteq \mathcal{N}^p \times \mathcal{R}^q$$

Abstract values are typed according to their dimensionality (p,q).

Dimensions

Dimensions are numbered from 0 to p+q-1 and are typed either as integer or real, depending on their rank w.r.t. the dimensionality of the abstract value.

> **Note:** Taking into account or not the fact that some dimensions are integers is left to underlying libraries. Treating them as real is still a correct approximation. The behaviour of the libraries in this regard may also depend on some options.

Other datatypes

In addition to abstract values, the interface also manipulates the following main datatypes:

scalar (number)
: Either GMP multiprecision rationals or C `double`.

interval
: composed of 2 scalar numbers. With rationals, plus (resp minus) infinity is represented by 1/0 (resp -1/0). With `double`, the IEEE754 is assumed and the corresponding standard representation is used.

coefficient
: which is either a scalar or an interval.

(interval) linear expression
: The term linear is used even if the proper term should rather be affine. A linear expression is a linear expression in the common sense, using only scalar numbers. A quasi-linear expression is a linear expression where the constant coefficient is an interval. An interval linear expression is a linear expression where any coefficient may be an interval. In order to have a unique datatype for these variations, we introduced the notion of coefficient described above.

“linear” constraints
: “Linear” constraints includes proper linear constraints, linear constraints in which the expression can be possibly an interval linear expression, linear equalities modulo a number, and linear disequalities.

generators
: A generator system for a subset of $X \subseteq R^n$ is a finite set of vectors, among which one distinguishes *points* $p_0, \ldots, p_m$ and *rays* $r_0, \ldots, r_n$, that generates X:

$$X = \{\lambda_0 \vec{p_0} + \ldots \lambda_m \vec{p_m} + \mu_0 \vec{r_0} + \ldots + \mu_n \vec{r_n} \mid \sum_i \lambda_i = 1 \ \wedge \ \forall j : \mu_j \geq 0\}$$

The APRON datatype for generators distinguishes points (sum of coefficients equal to one), rays (positive coefficients), lines (or bidirectional rays, with unconstrainted coefficients), integer rays (integer positive coefficients) and integer lines (integer coefficients).

Control of internal representation

We identified several notions:

- Canonical form
- Minimal form (in term of space)
- Approximation notion left to the underlying library (taking into account integers or not, ...).

Printing

There are two printing operations:

- Printing of an abstract value;
- Printing the difference between two abstract values.

The printing format is library dependent. However, the conversion of abstract values to constraints (see below) allows a form of standardized printing for abstract values.

Serializaton/Deserialization

Serialization and deserialization of abstract values to a memory buffer is offered. It is entirely managed by the underlying library. In particular, it is up to it to check that a value read from the memory buffer has the right format and has not been written by a different library.

Serialization is done to a memory buffer instead of to a file descriptor because this mechanism is more general and is needed for interfacing with languages like OCaml.

Constructors

Four basic constructors are offered:

- bottom (empty) and top (universe) values (with a specified dimensionality);
- abstraction of a bounding box;
- abstraction of conjunction of linear constraints (in the broad sense).

Tests

Predicates are offered for testing

- emptiness and universality of an abstract value:
- inclusion and equality of two abstract values;
- inclusion of a dimension into an interval given an abstract value;

$$abs(\vec{x}) \models x_i \in I \quad ?$$

- satisfaction of a linear constraint by the abstract value.

$$abs(\vec{x}) \models cons(\vec{x}) \;\; \texttt{or} \;\; abs(\vec{x}) \Rightarrow cons(\vec{x}) \quad ?$$

Property extraction

Some properties may be inferred given an abstract values:

- Interval of variation of a dimension in an abstract value;

$$\bigcap\{I \mid abs(\vec{x}) \models x_i \in I\}$$

- Interval of variation of a linear expression in an abstract value;

$$\bigcap\{I \mid abs(\vec{x}) \models expr(\vec{x}) \in I\}$$

- Conversion to a bounding box

$$\bigcap\{B \mid abs(\vec{x}) \subseteq B\}$$

- Conversion to a set of linear constraints (in the broad sense).

Notice that the second operation implements linear programming if it is exact. The third operation is not minimal, as it can be implemented using the first one, but it was convenient to include it. But the fourth operation is minimal and cannot be implemented using the second one, as the number of linear expression is infinite.

Lattice operations

- Least upper bound and greatest lower bound of two abstract values, and of arrays of abstract values;
- Intersection with one or several linear constraints;

$$\alpha\left(\gamma(abs(\vec{x})) \cap \bigcap_i cons_i(\vec{x})\right)$$

- Addition of rays (for instance for implement generalized time elapse operator in linear hybrid systems).

$$\alpha\left(\left\{\vec{x} + \sum_i \lambda_i \vec{r}_i \mid \vec{x} \in \gamma(abs), \lambda_i \geq 0\right\}\right)$$

Assignement and Substitutions

- of a dimension by a (interval) linear expression
  Assignement:

$$\alpha\left(\left(\exists x_i : \left(\gamma(abs(\vec{x})) \cap x'_i = expr(\vec{x})\right)\right)[x_i \leftarrow x'_i]\right)$$

  Substitution:

$$\alpha\Big(\exists x'_i : \left(\gamma(abs(\vec{x}))[x'_i \leftarrow x_i] \cap x'_i = expr(\vec{x})\right)\Big)$$

- in parallel of several dimensions by several (interval) linear expressions
  Assignement:

$$\alpha\left(\left(\exists \vec{x} : \left(\gamma(abs(\vec{x})) \cap \bigcap_i x'_i = expr_i(\vec{x})\right)\right)[\vec{x} \leftarrow \vec{x'}]\right)$$

  Substitution:

$$\alpha\Big(\exists \vec{x'} : \left(\gamma(abs(\vec{x'})) \cap \bigcap_i x'_i = expr(\vec{x})\right)\Big)$$

Parallel assignement and substitution ar enot minimal operations, but for some abstract domains implementing them directly results in more efficient or more precise operations.

Operations on dimensions

- Projection/Elimination of one or several dimensions with constant dimensionality;
  Elimination:
  $$\exists x_i : abs(\vec{x})$$
  Projection:
  $$(\exists x_i : abs(\vec{x})) \cap x_i = 0$$
- Addition/Removal/Permutation of dimensions with corresponding change of dimensionality (with the exception of permutation). These operations allows to resize abstract values, and reorganize dimensions.
- Expansion and folding of dimensions. This is useful for the abstraction of arrays, where a dimension may represent several variables.
  Expansion of i into i, j_1, j_2 assuming x_{j_1}, x_{j_2} are new dimensions:
  $$abs(\vec{x}) \sqcap abs(\vec{x})[x_{j_1} \leftarrow x_i] \sqcap abs(\vec{x})[x_{j_2} \leftarrow x_i]...$$
  Folding of j_0 and j_1 into j_0:
  $$(\exists x_{j_1} : abs(\vec{x})) \sqcup (\exists x_{j_0} : abs(\vec{x})[x_{j_0} \leftarrow x_{j_1}]$$

Other operations

Widening, either simple or with threshold, is offered. A generic widening with threshold function is offered in the interface.

Topological closure (i.e., relaxation of strict inequalities) is offered.

3.3 Functionalities of the interface at level 1

We focus on the changes brought by the level 1 w.r.t. the level 0.

Variables

Dimensions are replaced by *variables*.

In the C interface, variables are defined by a generic type (`char*`, structured type, ...), equipped with the operations `compare`, `copy`, `free`, `to_string`. In the OCaml, for technical reasons, the type is just the `string` type.

Environments manages the correspondance between the numerical dimensions of level 0 and the variables of level 1.

Semantics and Representation of an abstract value

The semantics of an abstract value is a subset

$$X \subseteq V \to (\mathcal{N} \cup \mathcal{R})$$

where X is a set of variables. Abstract values are typed according to their environment.

It is represented by a structure

```
struct ap_abstract1_t {
  ap_abstract0_t    *abstract0;
  ap_environment_t  *env;
};
```

Other datatypes of level 0 are extend in the same way. For instance,

```
struct ap_linexpr1_t {
  ap_linexpr0_t    *linexpr0;
  ap_environment_t *env;
};
```

Operations on environments

- creation, merging, destruction
- addition/removal/renaming of variables

Dynamic typing w.r.t. environments

For binary operations on abstract values, the environments should be the same.

For operations involving an abstract value and an other datatype (expression, constraint, ...), one checks that the environment of the expression is a subenvironment of the environment of the abstract value, and one resize if necessary.

Operations on variables in abstract values

Operations on dimensions are lifted to operations on variables:

- Projection/Elimination of one or several variables with constant environment;
- Addition/Removal/Renaming of variables with corresponding change of environment;
- Change of environment (possibly combining removal and addition of variables);
- Expansion and folding of variables.

4 APRON Guidelines

4.1 Installing APRON

You should look at '../README', '../Makefile.config.model' and '../Makefile' files.

4.2 C Programming Guidelines

4.2.1 C Headers and Libraries

Declarations needed to use an underlying library via APRON are collected in the C include files 'ap_global0.h' and 'ap_global1.h'. They respectively refer to the level 0 and the level 1 of the interface. One can also refer to single APRON modules with their corresponding include files 'ap_dimension.h', 'ap_lincons0.h', ... Header files '<stdio.h>', 'stdlib.h' and '<stdarg.h>' will be required.

Then, you should also include the header files of the underlying libraries you want to use it via APRON (for instance, 'box.h', 'pk.h', 'ap_ppl.h').

All programs using APRON must link against the 'libapron', 'libmpfr' and 'libgmp' libraries, and the underlying libraries you want to use it via APRON:

1. If some file 'test.c' uses the POLKA library via APRON, the compilation command should look like 'gcc -I$ITV/include -I$MPFR/include -I$GMP/include -I$APRON/include -L$MPFR/lib -L$GMP/lib -L$APRON/lib -o test test.c -lpolkaMPQ -lapron -lmpfr -lgmp', assuming that the POLKA library is used in its 'MPQ' version (internal number representation is GMP rationals) and resides in '$APRON/include' and '$APRON/lib' directories.

   The 'libpolkaMPQ.a' library is of course needed, 'libapron.a' contains all the code common to all APRON library (manipulation of expressions, environments, ...), as well as ITV functions (quasi)linearisation facilities of APRON,...), last the libraries libmpfr.a and libgmp.a are required both by NewPolka and APRON .

2. If some file 'test.c' uses the PPL library via APRON, the compilation command should look like 'g++ -I$ITV/include -I$MPFR/include -I$GMP/include -I$APRON/include -I$PPL/include -L$ITV/lib -L$MPFR/lib -L$GMP/lib -L$APRON/lib -L$PPL/lib -o test test.c -la_ppl -lppl -lgmpxx -lapron -lmpfr -lgmp', assuming that PPL resides in $PPL and PPL APRON interface in '$APRON/include' and '$APRON/lib' directories.

   Notice that the PPL library ('libppl.a') is a C++ library, you need to use 'g++' instead of 'gcc' for linking. You also need the C++ layer on top of GMP ('libgmpxx.a'). The 'libap_ppl.a' library contains the layer on top of PPL which implements the APRON interface.

You should look at the specific documentation of the libraries for more details.

4.2.2 Naming conventions and Allocation/Deallocation schemes

The general rule is that all type and function names defined by the library are prefixed with ap_, in order to prevent name conflicts with other libraries. Moreover, functions operating on objects of type ap_typ_t are usually prefixed with ap_typ_op.

Given an object of datatype ap_typ_t*, two kinds of allocation/deallocation pairs of functions may be defined:

1. variable declaration: ap_typ_t obj;

- Initialization: `void typ_init(ap_typ_t* arg, ...)` or `ap_typ_t ap_typ_make(...)`
- Finalization: `void ap_typ_clear(ap_typ_t* arg)`

this pair of functions follows the semantics used in the GMP library. The first function initializes the object of type `ap_typ_t` pointed to by *arg*, and fills it with a valid content. The second function deallocates the memory possibly pointed to by fields of the object `*arg`, but do not attempt to deallocate the memory pointed by *arg*.

2. variable declaration: `ap_typ_t* obj;`
   - Allocation `ap_typ_t* ap_typ_alloc(...)`
   - Deallocation `void ap_typ_free(ap_typ_t* arg)`

   the first function allocates an object of type `typ_t` and then calls a `ap_typ_init`-like function on the result; the second functions first call a `ap_typ_clear`-like function and then deallocate the memory pointed by *arg*.

4.2.3 Allocating managers and setting options

From the user point of view, the benefit of using the APRON interface is to restrict the place where the user is aware of the real library in use to the code initializing the manager, as illustrated by the following example:

```
#include "ap_global1.h"
#include "pk.h"

/* Allocating a Polka manager, for polyhedra with strict constraints */
manager_t* man = pk_manager_alloc(true);
/* Setting options offered by the common interface,
   but with meaning possibly specific to the library */
manager_set_abort_if_exception(man,EXC_OVERFLOW,true);
{
  funopt_t funopt;
  funopt_init(&funopt);
  funopt.algorithm = 1; /* default value is 0 */
  manager_set_funopt(man,fun_widening,&funopt); /* Setting options for widening */
}
{
  funopt_t funopt = manager_get_funopt(man,fun_widening);
  funopt.timeout = 30;
  manager_set_funopt(man,fun_widening,&funopt);
}
/* Obtaining the internal part of the manager and setting specific options */
pk_internal_t* pk = manager_get_internal(man);
pk_set_max_coeff_size(pk,size);
```

The standard operations can then be used and will have the semantics defined in the interface. Notice however that some generic functions are not formally generic: `abstract_fprint`, `abstract_fdump`, `abstract_approximate`. At any point, options may be modified in the same way as during the initialization.

4.2.4 Sequel of the small example

An environment can be created as follows:

```
/* Create an environment with 6 real variables */
ap_var_t name_of_dim[6] = {
  "x","y","z","u","w","v"
};
ap_environment_t* env = ap_environment_alloc(NULL,0,name_of_dim,6);
```

Then, we build an array of constraints. At level 1, an array of constraints is an abstract datatype, which requires careful manipulation w.r.t. memory management.

```
/* Create an array of constraints of size 2 */
ap_lincons1_array_t array = ap_lincons1_array_make(env,2);

/* 1.a Creation of an inequality constraint */
ap_linexpr1_t expr = ap_linexpr1_make(env,AP_LINEXPR_SPARSE,1);
ap_lincons1_t cons = ap_lincons1_make(AP_CONS_SUP,&expr,NULL);
    /* Now expr is memory-managed by cons */
/* 1.b Fill the constraint */
ap_lincons1_set_list(&cons,
     AP_COEFF_S_INT,"x",
     AP_CST_S_FRAC,1,2,
     AP_END);
/* 1.c Put in the array */
ap_lincons1_array_set(&array,0,&cons);
    /* Now cons is memory-managed by array */

/* 2.a Creation of an inequality constraint */
expr = ap_linexpr1_make(env,AP_LINEXPR_SPARSE,2);
cons = ap_lincons1_make(AP_CONS_SUPEQ,&expr,NULL);
    /* The old cons is not lost, because it is stored in the array.
       It would be an error to clear it (same for expr). */
/* 2.b Fill the constraint */
ap_lincons1_set_list(&cons,
     AP_COEFF_S_INT,1,"x",
     AP_COEFF_S_INT,1,"y",
     AP_COEFF_S_INT,1,"z",
     AP_END);
/* 2.c Put in the array */
ap_lincons1_array_set(&array,1,&cons);
```

Last we can build an abstract value.

```
/* Creation of an abstract value defined by the array of constraints */
ap_abstract1_t abs = ap_abstract1_of_lincons_array(man,env,&array);

fprintf(stdout,"Abstract value:\n");
ap_abstract1_fprint(stdout,man,&abs);
```

We now deallocate everything:

```
/* deallocation */
ap_lincons1_array_clear(&array);
ap_abstract1_clear(&abs);
ap_environment_free(env);
ap_manager_free(man);
```

4.2.5 Typing issue in C

The use of several libraries at the same time via the common interface and the managers associated to each library raises the problem of typing. Look at the following code:

```
ap_manager_t* manpk = pk_manager_alloc(true); /* manager for Polka */
ap_manager_t* manoct = oct_manager_alloc();    /* manager for octagon */

ap_abstract0_t* abs1 = ap_abstract_top(manpk,3,3);
ap_abstract0_t* abs2 = ap_abstract_top(manoct,3,3);
bool b = ap_abstract0_is_eq(manoct,abs1,abs2);
```

```
    /* Problem: the effective function called (octagon_is_eq) expects
       abs1 to be an octagon, and not a polyhedron ! */

  ap_abstract0_t* abs3 = ap_abstract_top(manoct,3,3);
  abstract0_meet_with(manpk,abs2,abs3);
    /* Problem: the effective function called (pk_meet_with) expects
       abs2 and abs3 to be polyhedra, but they are octagons */
```

There is actually no static typing, as `abstract0_t*` and `manager_t` are abstract types shared by the different libraries. Types are thus dynamically checked by the interface. Notice that the use of *C*++ and inheritance would not solve directly the problem, if functions of the interface are methods of the manager; one would have:

```
  ap_manager_t* manpk = pk_manager_alloc(true);
    /* manager for Polka, effective type pk_manager_t* */
  ap_manager_t* manoct = oct_manager_alloc();
    /* manager for octagon, effective type oct_manager_t* */

  ap_abstract0_t* abs1 = manpk->abstract_top(3,3);
    /* effective type: poly_t */
  ap_abstract0_t* abs2 = manoct->abstract_top(3,3);
    /* effective type: oct_t */
  bool b = manoct->abstract0_is_eq(abs1,abs2);
    /* No static typing possible:
       manpk->abstract0_is_eq and manoct->abstract0_is_eq should have the same
       signature (otherwise one cannot interchange manpk and manoct in the code),
       which means that abs1 and abs2 are supposed to be of type abstract0_t* */
  */
```

Currently, only the OCaml polymorphic type system allows to solve elegantly this problem.

4.3 OCaml Programming Guidelines

All modules belonging to the APRON interface itself (`Scalar`, `Interval`, ..., `Manager`, `Linexpr0`, ... `Abstract1`) are included in a big encapsulating module named `Apron`. In addition, there are modules like `Box` (intervals), `Oct` (octagons), `Polka` (linear equalities and convex polyhedra) and `Ppl` (convex polyhedra and linear congruences) not included in `Apron`.

Generic abstract values have the type `'a Abstract1.t`, generic managers the type `'a Manager.t`. A typical operation like the emptiness test has the type `val is_bottom : 'a Manager.t -> 'a Abstract1.t -> bool`.

Octagons of Octagon have the type `Oct.t Apron.Abstract1.t`. The corresponding managers have thus the type `Oct.t Manager.t`.

See OCaml interface for the documentation.

4.4 How to make an existing library conformant to APRON ?

We briefly describe here how to connect an existing library to the common interface.

First, the library has to expose an interface which conforms to the level 0 of the interface (module `abstract0`). All the functions described in this module should be defined. If a function is not really implemented, at least it shoulld contain the code raising the exception `EXC_NOT_IMPLEMENTED`. The implementor may use any functions of the files '`ap_coeff.h`', 'ap_linexpr0.h', 'ap_lincons0.h', 'ap_generator0.h' and 'ap_manager.h' to help the job of converting datatypes of the interface to internal datatypes used inside the library.

Second and last, the library should expose an initialization function that allocates and initializes properly an object of type `manager_t`. For this purpose, the module `manager` offers the utility functions `manager_alloc`. As an example, we give the definition of the function allocating a manager as implemented in the *NewPolka*.

1. Header of the function:

   ```
   manager_t* pk_manager_alloc(
     bool strict /* specific parameter: do we allow strict constaints ? */
   )
   ```

2. Allocation and initialisation of global data specific to *NewPolka*:

   ```
   {
     pk_internal_t* pk = pk_internal_alloc(strict); /* allocation */
     pk_set_approximate_max_coeff_size(pk, 1);
       /* initialization of specific functions
          (not offered in the common interface) */
   }
   ```

3. Allocation of the manager itself:

   ```
     manager_t* man = ap_manager_alloc("polka","2.0",
       pk, (void (*)(void*))pk_internal_free);
   ```

   We provide resp. name, version, internal specific manager, and the function to free it.

   The function `manager_alloc` sets the options of the common interface to their default value (see documentation).

4. Initialization of the "virtual" table: we need to connect the generic functions of the interface (eg, `abstract_meet`, \ldots) to the actual functions of the library.

   ```
     funptr = man->funptr;

     funptr[fun_minimize] = &poly_minimize;
     funptr[fun_canonicalize] = &poly_canonicalize;
     funptr[fun_hash] = &poly_hash;
     funptr[fun_approximate] = &poly_approximate;
     funptr[fun_fprint] = &poly_fprint;
     funptr[fun_fprintdiff] = &poly_fprintdiff;
     funptr[fun_fdump] = &poly_fdump;
     ...
   ```

5. Last, we return the allocated manager:

   ```
     return man;
   }
   ```

That's all for the implementor side.

5 Managers and Abstract Domains

APRON makes use of a global manager for:

- selecting an effective underlying library/abstract domain;
- controlling various options;
- managing exceptions and flags;
- and also managing internal workspace needed for some library.

In a multithreaded program, both managers and abstract values should not be shared between threads (make copies to transmit information).

Managers are allocated by the underlying libraries/abstract domains, but are freed via an APRON function.

5.1 Managers ('ap_manager.h')

5.1.1 Datatypes

tbool_t [datatype]

```
typedef enum tbool_t {
  tbool_false=0,
  tbool_true=1,
  tbool_top=2,   /* don't know */
} tbool_t;
static inline tbool_t tbool_of_bool(bool a);
static inline tbool_t tbool_or(tbool_t a, tbool_t b);
static inline tbool_t tbool_and(tbool_t a, tbool_t b);
```

Booleans with a third unknown value.

ap_membuf_t [datatype]

```
typedef struct ap_membuf_t {
  void* ptr;
  size_t size;
} ap_membuf_t;
```

For serialization.

ap_manager_t [datatype]

APRON managers (opaque type).

ap_funid_t [datatype]

For identifying functions in excpetions, and when reading/setting options attached to them.

```
typedef enum ap_funid_t {
  AP_FUNID_UNKNOWN,
  AP_FUNID_COPY,
  AP_FUNID_FREE,
  AP_FUNID_ASIZE, /* For avoiding name conflict with AP_FUNID_SIZE */
  AP_FUNID_MINIMIZE,
  AP_FUNID_CANONICALIZE,
  AP_FUNID_HASH,
  AP_FUNID_APPROXIMATE,
```

```
  AP_FUNID_FPRINT,
  AP_FUNID_FPRINTDIFF,
  AP_FUNID_FDUMP,
  AP_FUNID_SERIALIZE_RAW,
  AP_FUNID_DESERIALIZE_RAW,
  AP_FUNID_BOTTOM,
  AP_FUNID_TOP,
  AP_FUNID_OF_BOX,
  AP_FUNID_DIMENSION,
  AP_FUNID_IS_BOTTOM,
  AP_FUNID_IS_TOP,
  AP_FUNID_IS_LEQ,
  AP_FUNID_IS_EQ,
  AP_FUNID_IS_DIMENSION_UNCONSTRAINED,
  AP_FUNID_SAT_INTERVAL,
  AP_FUNID_SAT_LINCONS,
  AP_FUNID_SAT_TCONS,
  AP_FUNID_BOUND_DIMENSION,
  AP_FUNID_BOUND_LINEXPR,
  AP_FUNID_BOUND_TEXPR,
  AP_FUNID_TO_BOX,
  AP_FUNID_TO_LINCONS_ARRAY,
  AP_FUNID_TO_TCONS_ARRAY,
  AP_FUNID_TO_GENERATOR_ARRAY,
  AP_FUNID_MEET,
  AP_FUNID_MEET_ARRAY,
  AP_FUNID_MEET_LINCONS_ARRAY,
  AP_FUNID_MEET_TCONS_ARRAY,
  AP_FUNID_JOIN,
  AP_FUNID_JOIN_ARRAY,
  AP_FUNID_ADD_RAY_ARRAY,
  AP_FUNID_ASSIGN_LINEXPR_ARRAY,
  AP_FUNID_SUBSTITUTE_LINEXPR_ARRAY,
  AP_FUNID_ASSIGN_TEXPR_ARRAY,
  AP_FUNID_SUBSTITUTE_TEXPR_ARRAY,
  AP_FUNID_ADD_DIMENSIONS,
  AP_FUNID_REMOVE_DIMENSIONS,
  AP_FUNID_PERMUTE_DIMENSIONS,
  AP_FUNID_FORGET_ARRAY,
  AP_FUNID_EXPAND,
  AP_FUNID_FOLD,
  AP_FUNID_WIDENING,
  AP_FUNID_CLOSURE,
  AP_FUNID_SIZE,
  AP_FUNID_CHANGE_ENVIRONMENT,
  AP_FUNID_RENAME_ARRAY,
  AP_FUNID_SIZE2
} ap_funid_t;
```

```
extern const char* ap_name_of_funid[AP_FUNID_SIZE2];
/* give the name of a function identifier */
```

`ap_exc_t` [datatype]
`ap_exc_log_t` [datatype]

Exceptions and exception logs (chained in a list, the first one being the last one).

```
typedef enum ap_exc_t {
  AP_EXC_NONE,           /* no exception detected */
  AP_EXC_TIMEOUT,        /* timeout detected */
  AP_EXC_OUT_OF_SPACE,   /* out of space detected */
  AP_EXC_OVERFLOW,        /* magnitude overflow detected */
  AP_EXC_INVALID_ARGUMENT, /* invalid arguments */
  AP_EXC_NOT_IMPLEMENTED, /* not implemented */
  AP_EXC_SIZE
} ap_exc_t;
extern const char* ap_name_of_exception[AP_EXC_SIZE];
typedef struct ap_exclog_t {
  ap_exc_t exn;
  ap_funid_t funid;
  char* msg;                  /* dynamically allocated */
  struct ap_exclog_t* tail;
} ap_exclog_t;
```

`ap_funopt_t` [datatype]

Options attached to functions.

```
typedef struct ap_funopt_t {
  int algorithm;
  /* Algorithm selection:
     - 0 is default algorithm;
     - MAX_INT is most accurate available;
     - MIN_INT is most efficient available;
     - otherwise, no accuracy or speed meaning
  */
  size_t timeout; /* unit !? */
  /* Above the given computation time, the function may abort with the
     exception flag flag_time_out on.
  */
  size_t max_object_size; /* in abstract object size unit. */
  /* If during the computation, the size of some object reach this limit, the
     function may abort with the exception flag flag_out_of_space on.
  */
  bool flag_exact_wanted;
  /* return information about exactitude if possible
  */
  bool flag_best_wanted;
  /* return information about best correct approximation if possible
  */
} ap_funopt_t;
```

5.1.2 Functions related to managers

`void ap_manager_free (ap_manager_t* man)` [Function]
Free a manager (dereference a counter, and possibly deallocate).

`const char* ap_manager_get_library (ap_manager_t* man)` [Function]
`const char* ap_manager_get_version (ap_manager_t* man)` [Function]
Reading the name and the version of the attached underlying library.

`bool ap_manager_get_flag_exact (ap_manager_t* man)` [Function]
`bool ap_manager_get_flag_best (ap_manager_t* man)` [Function]
Return `true` if the last called APRON function returned an exact (resp. a best approximation) result.

Options

See [ap_funopt_t], page 27.

`ap_funopt_t ap_manager_get_funopt (ap_manager_t* man, ap_funid_t funid)` [Function]
Getting the option attached to the specified function in the manager. *funid* should be less than `AP_FUNID_SIZE` (no option associated to other identifiers). Otherwise, abort with a message.

`void ap_manager_set_funopt (ap_manager_t* man, ap_funid_t funid, ap_funopt_t* fopt)` [Function]
Setting the option attached to the specified function in the manager. *fopt* is copied (and not only referenced). *funid* should be less than `AP_FUNID_SIZE` (no option associated to other identifiers). Otherwise, do nothing.

`void ap_funopt_init (ap_funopt_t* fopt)` [Function]
Initialize *fopt* with default values.

Exceptions

`bool ap_manager_get_abort_if_exception (ap_manager_t* man, ap_exc_t exn)` [Function]
Return true if the program abort when the exception *exn* is raised by some function. Otherwise, in such a case, a valid (but dummy) value should be returned by the function that raises the exception.

`void ap_manager_set_abort_if_exception (ap_manager_t* man, ap_exc_t exn, bool flag)` [Function]
Position the above-described option.

`ap_exc_t ap_manager_get_exception (ap_manager_t* man)` [Function]
Get the last exception raised.

`ap_exclog_t ap_manager_get_exclog (ap_manager_t* man)` [Function]
Get the full log of exceptions. The first one in the list is the last raised one.

5.2 Box ('box.h'): intervals abstract domain

The Box interval library is aimed to be used through the APRON interface.

5.2.1 Use of Box

To use Box in C, add

```
#include "box.h"
```

in your source file(s) and add '-I$(BOX_PREFIX)/include' in the command line in your Makefile.

You should also link your object files with the Box library to produce an executable, by adding something like '-L$(APRON_PREFIX)/lib -lboxmpq -litvmpq' in the command line in your Makefile (followed by the standard '-lapron -litvmpq -litvdbl -L$(MPFR_PREFIX)/lib -lmpfr -L$(GMP_PREFIX)/lib -lgmp').

There are actually several variants of the library:

'libboxllr.a'
: The underlying representation for numbers is rationals based on `long long int` integers. This may cause overflows. These are currently not detected. It requires also the 'libitvllr.a' library.

'libboxdbl.a'
: The underlying representations for numbers is `double`. Overflows are not possible (we use infinite floating numbers), but currently the soundness is not ensured for all operations. It requires also the 'libitvdbl.a' library.

'libboxmpq.a'
: The underlying representations for rationams is `mpq_t`, the multi-precision rationals from the GNU GMP library. Overflows are not possible any more, but huge numbers may appear. It requires also the 'libitvmpq.a' library.

Also, all library are available in debug mode ('libboxmpq_debug.a', ...).

5.2.2 Allocating Box managers

`ap_manager_t* box_manager_alloc ()` [Function]

Allocate a APRON manager linked to the Box library.

5.3 Oct: octagon abstract domain

oct_doc.html

5.4 NewPolka ('pk.h'): convex polyhedra and linear equalities abstract domains

The NewPolka convex polyhedra and linear equalities library is aimed to be used through the APRON interface. However some specific points should be precised. First, NewPolka can use several underlying representations for numbers, which lead to several library variants. Second, some specific functions are needed, typically to allocate managers, and to specify special options.

5.4.1 Use of NewPolka

To use NewPolka in C, add

```
#include "pk.h"
```

```
#include "pkeq.h"
  /* if you want linear equalities */
```

in your source file(s) and add '-I$(APRON_PREFIX)/include' in the command line in your Makefile.

You should also link your object files with the NewPolka library to produce an executable, by adding something like '-L$(APRON_PREFIX)/lib -lpolkag' in the command line in your Makefile (followed by the standard '-lapron -litvmpq -litvdbl -L$(MPFR_PREFIX)/lib -lmpfr -L$(GMP_PREFIX)/lib -lgmp').

There are actually several variants of the library:

'libpolkai.a'
: The underlying representation for integers is `long int`. This may easily cause overflows, especially with many dimensions or variables. Overflows are not detected but usually result in infinite looping. The underlying representation for integers is `long long int`. This may (less) easily cause overflows.

'libpolkag.a'
: The underlying representation for integers is `mpz_t`, the multi-precision integers from the GNU GMP library. Overflows are not possible any more, but huge numbers may appear.

All scalars of type `double` are converted to scalars of type `mpq_t` inside NewPolka, as NewPolka works internally with exact rational arithmetics. So when possible it is better for the user (in term of efficiency) to convert already `double` scalars to `mpq_t` scalars.

There is a way to prevent overflow and/or huge numbers, which is to position the options `max_coeff_size` and `approximate_max_coeff_size`, see Section 5.4.2 [Allocating NewPolka managers and setting specific options], page 30.

Also, all library are available in debug mode ('libpolkai_debug.a', ...

5.4.2 Allocating NewPolka managers and setting specific options

`pk_internal_t` [datatype]

NewPolka type for internal managers (specific to NewPolka, and specific to each execution thread in multithreaded programs).

Allocating managers

`ap_manager_t* pk_manager_alloc (`*bool* `strict)` [Function]

Allocate an APRON manager for convex polyhedra, linked to the NewPolka library.

The *strict* option, when true, enables strict constraints in polyhedra (like `x>0`). Managers in strict mode or in loose mode (strict constraints disabled) are not compatible, and so are corresponding abstract values.

`ap_manager_t* pkeq_manager_alloc ()` [Function]

Allocate an APRON manager for linear equalities, linked to the NewPolka library.

Most options which makes sense for convex polyhedra are meaningless for linear equalities. It is better to set the standard options associated to functions so that abstract values are in canonical form (see Section 5.4.3 [NewPolka standard options], page 31). This is the default anyway.

Setting options

Options specific to NewPolka are set directly on the internal manager. It can be extracted with the `pk_manager_get_internal` function.

`pk_internal_t* pk_manager_get_internal` (*ap_manager_t* man*) [Function]
: Return the internal submanager. If *man* has not been created by `pk_manager_alloc` or `pkeq_manager_alloc`, return `NULL`.

`void pk_set_max_coeff_size` (*pk_internal_t* pk, size_t size*) [Function]
: If *size* is not 0, try to raise an `AP_EXC_OVERFLOW` exception as soon as the size of an integer exceed *size*.

  Very incomplete implementation. Currently, used only in '`libpolkag`' variant, where the size is the number of limbs as returned by the function `mpz_size` of the GMP library. This allows to detect huge numbers.

`void pk_set_approximate_max_coeff_size` (*pk_internal_t* pk, size_t size*) [Function]
: This is the parameter to the `poly_approximate`/`ap_abstractX_approximate` functions.

`size_t pk_get_max_coeff_size` (*pk_internal_t* pk*) [Function]
`size_t pk_get_approximate_max_coeff_size` (*pk_internal_t* pk*) [Function]
: Reading the previous parameters.

5.4.3 NewPolka standard options

This section describes the NewPolka options which are selected using the standard mechanism offered by APRON (see [Manager options], page 28).

Modes

Most functions of NewPolka has two modes. In the lazy mode the canonicalization (computation of the dual representation and minimisation of both representations) of the argument polyhedra is performed only when the needed representation is not available. The resulting polyhedra is in general not in the canonical representation. In the strict mode, argument polyhedra are canonicalized (if they are not yet in canonical form) and the result is (in general) in canonical form.

The strict mode exploits the incremental propery of the Chernikova algorithm and maintain in parallel the constraints and the generators representations. The lazy mode delays computations as much as possible.

Be cautious, in the following table, canonical means minimized constraints and generators representation, but nothing more. In particular, the function `canonicalize` performs further normalization by normalizing strict constraints (when they exist) and ordering constraints and generators.

Function	algo	Comments
copy		Identical representation
free		

size		Return the number of coefficients. Their size (when using multi-precision integers) is not taken into account.
minimize		Require canonicalization. Keep only the smallest representation among the constraints and the generators representation.
canonicalize		
approximate		Require constraints. algo here refers to the explicit parameter of the function. A negative number indicates a possibly smaller result, a positive one a possibly greater one. The effects of the function may be different for 2 identical polyhedra defined by different systems of (non minimal) constraints. Equalities are never modified.
	-1	Normalize integer minimal constraints. This results in a smaller polyhedra.
	1	Remove constraints with coefficients of size (in bits) greater than the approximate_max_coeff_size parameter.
	2	Idem, but preserve interval constraints.
	3	Idem, but preserve octagonal constraints (+/- xi +/- xj >= cst).
	10	Simplify constraints such that the coefficients size (in bits) are less or equal than the approximate_max_coeff_size parameter. The constant coefficients are recomputed by linear programming and are not involved in the reduction process.
	–	Do nothing
fprint		Require canonicalization.
fprintdiff		not implemented
fdump		Print raw representations of any of the constraints, generators and saturation matrices that are available.
serialize_raw, deserialize_raw		not implemented
bottom,top		Return canonical form.
of_box		Return constraints.
of_lincons_array		Return constraints.

	>=0	Take into account interval-linear constraints, after having minimized the quasi-linear constraints
	<0	Ignore interval-linear constraints
dimension		
is_bottom	<0	If generators not available, return `tbool_top`
	>=0	If generators not available, canonicalize and return `tbool_false` or `tbool_true`.
is_top	<0	If not in canonical form, return `tbool_top`
	>=0	Require canonical form.
is_leq	<=0	Require generators of first argument and constraints of second argument.
	>0	Require canonical form for both arguments.
is_eq		Require canonical form for both arguments.
is_dimension_unconstrained		Require canonical form
sat_interval, sat_lincons, bound_dimension, bound_linexpr	<=0	Require generators.
	>0	Require canonical form.
to_box	<0	Require generators.
	>=0	Require canonical form.
to_lincons_array, to_generator_array		Require canonical form.
meet, meet_array, meet_lincons_array	<0	Require constraints. Return non-minimized constraints.
	>=0	Require canonical form. Return canonical form.
join, join_array, add_ray_array	<0	Require generators. Return non-minimized generators.
	>=0	Require canonical form. Return canonical form.
assign_linexpr		1. If the optional argument is NULL,
	<=0	If the expr. is deterministic and invertible, require any representation and return the transformed one. If in canonical form, return canonical form. If the expr. is deterministic and non-invertible, require generators and return generators If the expr. is non-deterministic, require constraints and return generators.

	>0	Require canonical form, return canonical form. If the expr. is deterministic,(and even more, invertible), the operation is more efficient. 2. If the optional argument is not NULL, first the assignement is performed, and then the meet function is applied with its corresponding option.
substitute_linexpr		1. If the optional argument is NULL,
	<=0	If the expr. is deterministic and invertible, require any representation and return the transformed one. If in canonical form, return canonical form. If the expr. is deterministic and non-invertible, require constraints and return constraints If the expr. is non-deterministic, require constraints and return generators.
	>0	Require canonical form, return canonical form. If the expr. is deterministic (and even more, invertible), the operation is more efficient. 2. If the optional argument is not NULL, first the substitution is performed, and then the meet function is applied with its corresponding option.
assign_linexpr_array		1. If the optional argument is NULL,
	<=0	If the expr. are deterministic, require generators and return generators Otherwise, require canonical form and return generators.
	>0	Require canonical form, return canonical form. 2. If the optional argument is not NULL, first the assignement is performed, and then the meet function is applied with its corresponding option.
substitute_linexpr_array		1. If the optional argument is NULL,
	<=0	If the expr. are deterministic, require constraints and return constraints Otherwise, require canonical form and return generators.
	>0	Require canonical form, return canonical form. 2. If the optional argument is not NULL, first the substitution is performed, and then the meet function is applied with its corresponding option.
forget_array	<=0	Require generators and return generators.
	>0	Require canonical form and return canonical form.
add_dimensions, permute_dimensions	<=0	Require any representation and return the updated one. If in canonical form, return canonical form.

	>0	Require canonical form, return canonical form.
remove_dimensions	<=0	Require generators, return generators.
	>0	Require canonical form, return canonical form.
expand	<0	Require constraints, return constraints.
	>=0	Require canonical form, return canonical form.
fold	<0	Require generators, return generators.
	>=0	Require canonical form, return canonical form.
widening		Require canonical form.
closure		1. If pk_manager_alloc() has been given a false Boolean (no strict constraints), same as copy.
		2. Otherwise,
	<0	Require constraints, return constraints.
	>=0	Require canonical form, return constraints.

5.5 PPL (‘`ap_ppl.h`’): convex polyhedra and linear congruences abstract domains

The APRON PPL library is an APRON wrapper around the Parma Polyhedra Library (PPL). The wrapper offers the convex polyhedra and linear congruences abstract domains.

5.5.1 Use of APRON PPL

To use APRON PPL in C, you need of course to install PPL, *after having patched it* following the recommendations of the ‘`README`’ file. You need also to add

```
#include "apron_ppl.h"
```

in your source file(s) and add ‘`-I$(APRON_PREFIX)/include`’ in the command line in your Makefile.

You should also link your object files with the APRON PPL library to produce an executable, *using* ‘`g++`’ (instead of ‘`gcc`’, because ‘`libppl.a`’ is a C++ library), and adding something like ‘`-L$(APRON_PREFIX)/lib -lapron_ppl -L$(PPL_PREFIX)/lib -lppl -L$(GMP_PREFIX)/lib -lgmpxx`’ in the command line in your Makefile (followed by the standard ‘`-lapron -litvmpq -litvdbl -L$(MPFR_PREFIX)/lib -lmpfr -L$(GMP_PREFIX)/lib -lgmp`’). The ‘`libgmpxx.a`’ library is the C++ wrapper on top of the GMP library. Ensure that your GMP installation contains it, as it is not always installed by default.

All scalars of type `double` are converted to scalars of type `mpq_t` inside APRON PPL, as APRON PPL works internally with exact rational arithmetics. So when possible it is better for the user (in term of efficiency) to convert already `double` scalars to `mpq_t` scalars.

The wrapper library is available in debug mode (‘`libapron_ppl_debug.a`’).

5.5.2 Allocating APRON PPL managers

`ap_manager_t* ap_ppl_poly_manager_alloc` (*bool* `strict`) [Function]

Allocate a APRON manager for convex polyhedra, linked to the PPL library.

The *strict* option, when true, enables strict constraints in polyhedra (like `x>0`). Managers in strict mode or in loose mode (strict constraints disabled) are not compatible, and so are corresponding abstract values.

`ap_manager_t* ap_ppl_grid_manager_alloc ()` [Function]

Allocate an APRON manager for linear equalities, linked to the PPL library.

5.5.3 APRON PPL standard options

Currently, the only options available are related to the widening operators.

Function	algo	Comments
widening	<=0	CH78 standard widening (Cousot & Halbwachs, POPL'1978).
	>0	BHRZ03 widening (Bagnara, Hill, Ricci & Zafanella, SAS'2003)
widening_threshold	<=0	standard widening with threshold
	=1	standard widening with threshold, intersected by the bounding box of the convex hull pof the two arguments
	<=0	standard widening with threshold
	=1	standard widening with threshold, intersected by the bounding box of the convex hull of the second argument. This is actually an extrapolation rather than a widening (termination is not guaranteed)
	=2	BHRZ03 widening with threshold
	=3	BHRZ03 widening with threshold, intersected by the bounding box of the convex hull of the second argument. This is actually an extrapolation rather than a widening (termination is not guaranteed)

5.6 pkgrid ('`ap_pkgrid.h`'): reduced product of NewPolka convex polyhedra and PPL linear congruences abstract domains

The PKGRID library is aimed to be used through the APRON interface. It implements the reduced product of NewPolka convex polyhedra and the PPL linear congruences abstract domains and implementations. It exploits for this the features offered by the module '`ap_reducedproduct`' contained in the '`apron`' core library.

5.6.1 Use of pkgrid

To use PKGRID in C, add

```
#include "ap_pkgrid.h"
```

in your source file(s) and add '`-I$(APRON_PREFIX)/include`' in the command line in your Makefile.

You should also link your object files with the PKGRID library to produce an executable, by adding something like '`-L$(APRON_PREFIX)/lib -lap_pkgrid`' in the command line in your Makefile, followed by the flags and libraries needed for the NewPolka library (see Section 5.4.1 [Use of NewPolka], page 29) and the PPL library (see Section 5.5.1 [Use of APRON PPL], page 35). Be cautious: because of the use of the PPL library, you '`g++`' (C++ compiler) instead of '`gcc`' (C compiler) for the linking.

Also, the library is available in debug mode ('`libap_pkgrid_debug.a`', '`libap_pkgrid_debug.so`').

5.6.2 Allocating pkgrid managers

`ap_manager_t* ap_pkgrid_manager_alloc` (*ap_manager_t** ***manpk***, *ap_manager_t** ***manpplgrid***) [Function]

Allocate a APRON manager linked to the pkgrid library, using the (loose or strict) polka manager *manpk* and the PPL grid manager *manpplgrid*. If one of the argulment manager is not of the right type, returns `NULL`.

Available standard options are the one offered by the generic reduced product module '`ap_reducedproduct`' contained in the '`apron`' core library (see Chapter 9 [Functions for implementors], page 89).

6 Scalars & Intervals & coefficients

Scalars are scalar numbers, implemented either as an (inexact) floating point type or an (exact) rational type. *Intervals* are intervals built on scalars. *Coefficients* are either scalars or intervals.

We sum up the involved types below (numbers denotes sizes in bytes on a typical 32 bits computer.

```
       ap_scalar_t     12       ap_interval_t  8
   |-----------------|        |--------------|
   | ap_scalar_discr |  4     | ap_scalar_t* | 4
   |-----------------|        |--------------|
   | double | mpq_t* |  8     | ap_scalar_t* | 4
   |-----------------|        |--------------|


                   ap_coeff_t            8
        |-------------------------------|
        |        ap_coeff_discr         | 4
        |-------------------------------|
        | ap_scalar_t* | ap_interval_t* | 4
        |-------------------------------|
```

These types are manipulated using pointers, with creator `X_t* X_alloc()` and destructors `void X_free(X_t*)`.

6.1 Scalars ('ap_scalar.h')

`ap_scalar_discr_t` [datatype]

```
typedef enum ap_scalar_discr_t {
  AP_SCALAR_DOUBLE, /* floating-point with double */
  AP_SCALAR_MPQ     /* rational with multi-precision GMP */
} ap_scalar_discr_t;
```

Discriminant indicating the underlying type of a scalar number.

`ap_scalar_t` [datatype]

```
typedef struct ap_scalar_t {
  ap_scalar_discr_t discr;
  union {
    double dbl;
    mpq_ptr mpq; /* +infty coded by 1/0, -infty coded by -1/0 */
  } val;
} ap_scalar_t;
```

A scalar number is either a double, or a multi-precision rational, as implemented by GMP.

6.1.1 Initializing scalars

`ap_scalar_t* ap_scalar_alloc ()` [Function]

Allocate a scalar, of default type DOUBLE (the most economical)

`void ap_scalar_free (`*ap_scalar_t* op*`)` [Function]

Deallocate a scalar.

`void ap_scalar_reinit (ap_scalar_t* op, ap_scalar_discr_t discr)` [Function]

Change the type of an already allocated scalar (mainly for internal use)

`void ap_scalar_init (ap_scalar_t* op, ap_scalar_discr_t discr)` [Function]
`void ap_scalar_clear (ap_scalar_t* op)` [Function]

Initialize and clear a scalar \'a la GMP (internal use).

6.1.2 Assigning scalars

`void ap_scalar_set (ap_scalar_t* rop, ap_scalar_t* op)` [Function]

Set the value of *rop* from *op*.

`void ap_scalar_set_mpq (ap_scalar_t* rop, mpq_t mpq)` [Function]
`void ap_scalar_set_int (ap_scalar_t* rop, long int i)` [Function]
`void ap_scalar_set_frac (ap_scalar_t* rop, long int i, unsigned long int j)` [Function]

Change the type of *rop* to MPQ and set its value to resp. *mpq*, *i*, and *i/j*.

`void ap_scalar_set_double (ap_scalar_t* rop, double k)` [Function]

Change the type of *rop* to DOUBLE and set its value to *k*.

`void ap_scalar_set_infty (ap_scalar_t* rop, int sgn)` [Function]

Set the value of *rop* to *sgn**infinity. Keep the type of the *rop*.

`ap_scalar_t* ap_scalar_alloc_set (ap_scalar_t* op)` [Function]
`ap_scalar_t* ap_scalar_alloc_set_mpq (mpq_t mpq)` [Function]
`ap_scalar_t* ap_scalar_alloc_set_double (double k)` [Function]

Combined allocation and assignement.

6.1.3 Converting scalars

`void ap_mpq_set_scalar (mpq_t mpq, ap_scalar_t* op, int round)` [Function]

Set *mpq* with the value of *op*, possibly converting from DOUBLE type.

round currently unused.

`double ap_scalar_get_double (ap_scalar_t* op, int round)` [Function]

Return the value of *op* in DOUBLE type, possibly converting from MPQ type.

Conversion may be not exact. *round* currently unused.

6.1.4 Comparing scalars

`int ap_scalar_infty (ap_scalar_t* op)` [Function]

Return `-1` if *op* is set to +infty, `-1` if set to -infty, and `0` otherwise.

`int ap_scalar_sgn (ap_scalar_t* op)` [Function]

Return the sign of *op* (`+1`, `0` or `-1`).

`int ap_scalar_cmp (ap_scalar_t* op1, ap_scalar_t* op2)` [Function]
`int ap_scalar_cmp_int (ap_scalar_t* op1, int op2)` [Function]

Exact comparison between two scalars (resp. a scalar and an integer).

Return `-1` if *op1* is less than *op2*, `0` if they are equal, and `+1` if `op1` is greater than *op2*.

`bool ap_scalar_equal (ap_scalar_t* op1, ap_scalar_t* op2);` [Function] `bool ap_scalar_equal_int` (*ap_scalar_t* op1, int op2*); [Function]

Equality test between two scalars (resp. a scalar and an integer).

Return `true` if equality.

6.1.5 Other operations on scalars

`void ap_scalar_neg` (*ap_scalar_t* rop, ap_scalar_t* op*) [Function]

Negation.

`void ap_scalar_inv` (*ap_scalar_t* rop, ap_scalar_t* op*) [Function]

Inversion. Not exact for DOUBLE type.

`void ap_scalar_swap` (*ap_scalar_t* op1, ap_scalar_t* op2*) [Function]

Exchange the values of *op1* and *op2*.

`int ap_scalar_hash` (*ap_scalar_t* op*) [Function]

Return an hash code (for instance for OCaml interface).

`void ap_scalar_fprint` (*FILE* stream, ap_scalar_t* op*) [Function]

Print *op* on the stream *stream*.

6.2 Intervals (‘ap_interval.h’)

`ap_interval_t` [datatype]

```
typedef struct ap_interval_t {
  ap_scalar_t* inf;
  ap_scalar_t* sup;
} ap_interval_t;
```

Intervals on scalars.

6.2.1 Initializing intervals

`void ap_interval_alloc` () [Function]

Allocate an interval (with scalars of default type DOUBLE, the most economical).

`void ap_interval_free` (*ap_interval_t* op*) [Function]

Deallocate an interval.

`void ap_interval_reinit` (*ap_interval_t* op, ap_scalar_discr_t discr*) [Function]

Change the type of the bounds of the interval (mainly for internal use).

6.2.2 Assigning intervals

`void ap_interval_set` (*ap_interval_t* rop, ap_interval_t* op*) [Function]

Set the value of *rop* from *op*.

`void ap_interval_set_scalar` (*ap_interval_t* rop, ap_scalar_t* inf, ap_scalar_t* sup*) [Function]

Set the value of *rop* from the interval [*inf*,*sup*].

`void ap_interval_set_mpq` (*ap_interval_t* rop, mpq_t inf, mpq_t sup*) [Function]

`void ap_interval_set_int` (*ap_interval_t* rop, int inf, int sup*) [Function] `void ap_interval_set_frac` (*ap_interval_t* **rop**, int **numinf**, int **deninf**, int **numsup**, int **densup***) [Function]

Set the value of *rop* from the interval [*inf*,*sup*] or [*numinf*/*deninf*,*numsup*/*densup*]. The scalars are of type MPQ.

`void ap_interval_set_double` (*ap_interval_t* **rop**, double **inf**, double **sup***) [Function]

Set the value of *rop* from the interval [*inf*,*sup*]. The scalars are of type DOUBLE.

`void ap_interval_set_top` (*ap_interval_t* **op***) [Function]
`void ap_interval_set_bottom` (*ap_interval_t* **op***) [Function]

Set the value of *rop* resp. to the top interval [-oo,+oo] or to the empty interval [+1,-1].

`ap_interval_t* ap_interval_alloc_set` (*ap_interval_t* **op***) [Function]

Combined allocation and assignement.

6.2.3 Comparing intervals

`bool ap_interval_is_top` (*ap_interval_t* **op***) [Function]
`bool ap_interval_is_bottom` (*ap_interval_t* **op***) [Function]

Return `true` if the interval is resp. the universe interval ([-oo,+oo]) or an empty interval.

`bool ap_interval_is_leq` (*ap_interval_t* **op1**, ap_interval_t* **op2***) [Function]

Inclusion test.

Return true if the interval *op1* is included in *op2*.

`bool ap_interval_equal` (*ap_interval_t* **op1**, ap_interval_t* **op2***) [Function]

Equality test.

Return true if the interval *op1* is included in *op2*.

`int ap_interval_cmp` (*ap_interval_t* **op1**, ap_interval_t* **op2***) [Function]

Non-total comparison.

`0` equality
`-1` *op1* included in *op2*
`+1` *op2* included in *op1*
`-2` *op1.inf* less than *op2.inf*
`+2` *op1.inf* greater than *op2.inf*

6.2.4 Other operations on intervals

`void ap_interval_neg` (*ap_interval_t* **rop**, ap_interval_t* **op***) [Function]

Negation.

`void ap_interval_swap` (*ap_interval_t* **op1**, ap_interval_t* **op2***) [Function]

Exchange the values of *op1* and *op2*.

`int ap_interval_hash` (*ap_interval_t* **op***) [Function]

Return an hash code (for instance for OCaml interface).

`void ap_interval_fprint` (*FILE* **stream**, ap_interval_t* **op***) [Function]

Print *op* on the stream *stream*.

6.2.5 Array of intervals

`ap_interval_t** ap_interval_array_alloc` (*size_t* **size**) [Function]

Allocate an array of intervals, initialized with [0,0] values.

`void ap_interval_array_free` (*ap_interval_t*** **array**, *size_t* **size**) [Function]

Clearing and deallocating an array of intervals.

6.3 Coefficients ('ap_coeff.h')

`ap_coeff_discr_t` [datatype]

```
typedef enum ap_coeff_discr_t { AP_COEFF_SCALAR, AP_COEFF_INTERVAL }
ap_coeff_discr_t;
```

Discriminant indicating the underlying type of a coefficient.

`ap_coeff_t` [datatype]

```
typedef struct ap_coeff_t {
  ap_coeff_discr_t discr;
  union {
    ap_scalar_t* scalar;
    ap_interval_t* interval;
  } val;
} ap_coeff_t;
```

A coefficient is either a scalar or an interval.

6.3.1 Initializing coefficients

`void ap_coeff_alloc` (*ap_coeff_discr_t* **discr**) [Function]

Allocate a coefficient, using *discr* to specify the type of coefficient (scalar or interval).

`void ap_coeff_free` (*ap_coeff_t** **op**) [Function]

Deallocate a coefficient.

`void ap_coeff_reinit` (*ap_coeff_t** **op**, *ap_coeff_discr_t* **discr1**, *ap_scalar_discr_t* **discr2**) [Function]

Changing the type of the coefficient and also the type of the underlting scalar(s).

`void ap_coeff_reduce` (*ap_coeff_t** **op**) [Function]

If the coefficient is an interval [a;a], convert it to a scalar. */

`void ap_coeff_init` (*ap_coeff_t** **rop**, *ap_coeff_discr_t* **discr**) [Function]
`void ap_coeff_init_set` (*ap_coeff_t** **rop**, *ap_coeff_t** **op**) [Function]
`void ap_coeff_clear` (*ap_coeff_t** **rop**) [Function]

Initialize, initialize and assign, and clear a scalar \'a la GMP (internal use).

6.3.2 Assigning coefficients

`void ap_coeff_set` (*ap_coeff_t** **rop**, *ap_coeff_t** **op**) [Function]

Set the value of *rop* from *op*.

`void ap_coeff_set_scalar` (*ap_coeff_t** **rop**, *ap_scalar_t** **op**) [Function]
`void ap_coeff_set_scalar_mpq` (*ap_coeff_t** **rop**, *mpq_t* **mpq**) [Function]
`void ap_coeff_set_scalar_int` (*ap_coeff_t** **rop**, *long int* **i**) [Function]

`void ap_coeff_set_scalar_frac` (*ap_coeff_t* rop*, *long int i*, *unsigned long int j*) [Function]

`void ap_coeff_set_scalar_double` (*ap_coeff_t* rop*, *double k*) [Function]

Set the type of *rop* to scalar, and sets its value as the functions `ap_scalar_set_XXX`.

`void ap_coeff_set_interval` (*ap_coeff_t* rop*, *ap_interval_t* op*) [Function]

`void ap_coeff_set_interval_scalar` (*ap_coeff_t* rop*, *ap_scalar_t* inf*, *ap_scalar_t* sup*) [Function]

`void ap_coeff_set_interval_mpq` (*ap_coeff_t* rop*, *mpq_t inf*, *mpq_t sup*) [Function]

`void ap_coeff_set_interval_int` (*ap_coeff_t* rop*, *int inf*, *int sup*) [Function]

`void ap_coeff_set_interval_frac` (*ap_coeff_t* rop*, *int numinf*, *int deninf*, *int numsup*, *int densup*) [Function]

`void ap_coeff_set_interval_double` (*ap_coeff_t* rop*, *double inf*, *double sup*) [Function]

Set the type of *rop* to interval, and sets its value as the functions `ap_interval_set_XXX`.

`ap_coeff_t* ap_coeff_alloc_set` (*ap_coeff_t* op*) [Function]

`ap_coeff_t* ap_coeff_alloc_set_scalar` (*ap_scalar_t* scalar*) [Function]

`ap_coeff_t* ap_coeff_alloc_set_interval` (*ap_interval_t* interval*) [Function]

Combined allocation and assignement.

6.3.3 Comparing coefficients

`int ap_coeff_cmp` (*ap_coeff_t* op1*, *ap_coeff_t* op2*) [Function]

Non-total comparison.

- If *op1* and *op2* are scalars, corresponds to `ap_scalar_cmp`.
- If *op1* and *op2* are intervals, corresponds to `ap_interval_cmp`.
- otherwise, -3 if the first is a scalar, 3 otherwise

`bool ap_coeff_equal` (*ap_coeff_t* op1*, *ap_coeff_t* op2*) [Function]

Equality test.

`bool ap_coeff_zero` (*ap_coeff_t* op*) [Function]

Return true iff coeff is a zero scalar or an interval with zero bounds.

6.3.4 Other operations on coefficients

`void ap_coeff_neg` (*ap_coeff_t* rop*, *ap_coeff_t* op*) [Function]

Negation.

`void ap_coeff_swap` (*ap_coeff_t* op1*, *ap_coeff_t* op2*) [Function]

Exchange the values of *op1* and *op2*.

`int ap_coeff_hash` (*ap_coeff_t* op*) [Function]

Return an hash code (for instance for OCaml interface).

`void ap_coeff_fprint` (*FILE* stream*, *ap_coeff_t* op*) [Function]

Print *op* on the stream *stream*.

7 Level 1 of the interface

This interface of level 1 is defined in ‘ap_global1.h’.

The main functions brought by level 1 are

- to convert variables to dimensions, thanks to the addition of environments to objects;
- to redimension (abstract values), expressions, constraints and generators defined on different environments.

The policy for redimensioning is the following one:

- For functions taking one abstract value and one expression (or constraint or generator, or array of ...), the environment of the expression should be a sub-environment of the environment of the abstract value. The environment of the result is the environment of the argument abstract value.
- For functions taking several abstract values, their environments should be the same. Otherwise, it is up to the user to move them to a common super-environment (see Section 7.2 [Environments], page 46 and Section 7.8.12 [Change of environments of abstract values of level 1], page 68).

For information only (as these types are considered as abstract), we sum up the involved types below.

```
   ap_var_t               ap_var_t      ap_environment_t
  |-------|              |-------|    |----------------------|
  | void* | by default   | char* |    | ap_var_t* var_of_dim |
  |-------|              |-------|    | size_t    intdim     |
                                      | size_t    realdim    |
                                      | size_t    count      |
                                      |----------------------|

     ap_linexpr1_t
  |-------------------|
  | ap_linexpr0_t*    |
  | ap_environment_t* |
  |-------------------|

     ap_lincons1_t          ap_lincons1_array_t
  |-------------------|   |----------------------|
  | ap_lincons0_t*    |   | ap_lincons0_array_t* |
  | ap_environment_t* |   | ap_environment_t*    |
  |-------------------|   |----------------------|

     ap_generator1_t        ap_generator1_array_t
  |-------------------|   |------------------------|
  | ap_generator0_t*  |   | ap_generator0_array_t* |
  | ap_environment_t* |   | ap_environment_t*      |
  |-------------------|   |------------------------|

     ap_texpr1_t
  |-------------------|
  | ap_texpr0_t*      |
  | ap_environment_t* |
```

```
|------------------|

   ap_tcons1_t          ap_tcons1_array_t
|------------------|  |---------------------|
| ap_tcons0_t*     |  | ap_tcons0_array_t*  |
| ap_environment_t* |  | ap_environment_t*   |
|------------------|  |---------------------|

   ap_abstract1_t
|------------------|
| ap_abstract0_t*   |
| ap_environment_t* |
|------------------|
```

7.1 Variables and related operations ('ap_var.h')

A variable is not necessarily a name, it can be a more complex structured datatype, depending on the application. That is the motivation to make it a parameter of the interface.

The abstract type `ap_var_t` is equipped with a total ordering function, a hashing function, a copy function, and a free function. The parametrization of the interface is performed via a global variable pointing to a `ap_var_operations_t` structure, containing the above-mentione doperations on `ap_var_t` objects. This means that this type should be fixed once, and that in a multitreaded application all threads should share the same `ap_var_t` type.

By default, `ap_var_t` is a C string (`char*`), and the global variable `ap_var_operations` is properly initialized.

`ap_var_t` [datatype]

```
typedef void* ap_var_t;
```

Datatype for "variables". It is assumed to be of size `sizeof(void*)`.

`ap_var_operations_t` [datatype]

```
typedef struct ap_var_operations_t {
  int (*compare)(ap_var_t v1, ap_var_t v2); /* Total ordering function */
  int (*hash)(ap_var_t v);                  /* Hash function */
  ap_var_t (*copy)(ap_var_t var);           /* Duplication function */
  void (*free)(ap_var_t var);               /* Deallocation function */
  char* (*to_string)(ap_var_t var); /* Conversion to a dynamically allocated string,
       which should be deallocated with free after use */
} ap_var_operations_t;
```

Datatype for defining the operations on "variables".

`ap_var_operations_t var_operations_default` [Variable]

Default manager, where `ap_var_t` is assumed to be `char*`.

`ap_var_operations_t* var_operations` [Variable]

Global pointer to the manager in use, by default points to `ap_var_operations_default`.

7.2 Environments ('ap_environment.h')

Environments bind variables (of level 1) to dimensions (of level 0).

`ap_environment_t` [datatype]

Internal datatype for environments.

For information, the definition is:

```
typedef struct ap_environment_t {
  ap_var_t* var_of_dim;
  /*
    Array of size intdim+realdim, indexed by dimensions.
    - It should not contain identical strings..
    - Slice [0..intdim-1] is lexicographically sorted,
      and denotes integer variables.
    - Slice [intdim..intdim+realdim-1] is lexicographically sorted,
      and denotes real variables.
    - The memory allocated for the variables are attached to the structure
      (they are freed when the structure is no longer in use)
  */
  size_t intdim; /* Number of integer variables */
  size_t realdim;/* Number of real variables */
  size_t count; /* For reference counting */
} ap_environment_t;
```

`void ap_environment_free` (*ap_environment_t** ***env***) [Function]

`ap_environment_t* ap_environment_copy` (*ap_environment_t** ***env***) [Function]

Respectively free and duplicate an environment.

(`copy` is cheap, as environments are managed with reference counters).

`void ap_environment_fdump` (*FILE** ***stream***, *ap_environment_t** ***env***) [Function]

Print an environment under the form:

```
environment: dim = (..,..), count = ..
0: name0
1: name1
...
```

`ap_environment_t* ap_environment_alloc_empty` () [Function]

Build an empty environment.

`ap_environment_t* ap_environment_alloc` (*ap_var_t** ***var_of_intdim***, *size_t* ***intdim***, *ap_var_t** ***var_of_realdim***, *size_t* ***realdim***) [Function]

Build an environment from an array of integer and an array of real variables.

var_of_intdim is an array of variables of size *intdim*, *var_of_realdim* is an array of variables of size *realdim*. Pointers to arrays may be `NULL` if their size is 0.

Variables are duplicated in the result, so it is the responsability of the user to free the variables he provides.

If some variables are duplicated, return `NULL`.

`ap_environment_t* ap_environment_add` (*ap_environment_t** ***env***, *ap_var_t** ***var_of_intdim***, *size_t* ***intdim***, *ap_var_t** ***var_of_realdim***, *size_t* ***realdim***) [Function] `ap_environment_t* ap_environment_remove` (*ap_environment_t** `env`, *ap_var_t** `tvar`, *size_t* `size`) [Function]

Resp. add or remove new variables to an existing environment, with a functional semantics. Same conventions as for `ap_environment_alloc` function apply. If the result is non-sense (or in case of attempt to remove an unknwon variable), return `NULL`.

`ap_dim_t ap_environment_dim_of_var` (*ap_environment_t** `env`, *ap_var_t* `var`) [Function]

Convert a variable in its corresponding dimension in the environment *env*. If *var* is unknown in *env*, return `AP_DIM_MAX`.

`ap_dim_t ap_environment_var_of_dim` (*ap_environment_t** `env`, *ap_dim_t* `dim`) [Function]

Return the variable associated to the dimension *dim* in the environment *env*. There is no bound check here.

The remaining functions are much less useful for normal user.

`bool ap_environment_is_eq` (*ap_environment_t** `env1`, *ap_environment_t** `env2`) [Function]

`bool ap_environment_is_leq` (*ap_environment_t** `env1`, *ap_environment_t** `env2`) [Function]

Resp. test the equality and the inclusion of two environments.

`int ap_environment_compare` (*ap_environment_t** `env1`, *ap_environment_t** `env2`) [Function]

Return:

`-2` if the environments are not compatible (a variable has a different type in the 2 environments);

`-1` if *env1* is a subset of (included in) *env2*;

`0` if they are equal;

`+1` if *env1* is a superset of *env2*;

`+2` otherwise: the least common environment exists and is a strict superset of both environments.

`int ap_environment_hash` (*ap_environment_t** `env`) [Function]

Return an hash code for an environment.

`ap_dimchange_t* ap_environment_dimchange` (*ap_environment_t** `env1`, *ap_environment_t** `env`) [Function]

Compute the transformation for converting from an environment *env1* to a superenvironment *env*. Return NULL if *env* is not a superenvironment.

`ap_dimchange2_t* ap_environment_dimchange2` (*ap_environment_t** `env1`, *ap_environment_t** `env2`) [Function]

Compute the transformation for switching from an environment *env1* to an *env2*, by first adding (some) variables of *env2*, and then removing (some) variables of *env1*. Return NULL if *env1* and *env2* ar enot compatible environments.

`ap_environment_t* ap_environment_lce` (*ap_environment_t** ***env1***, *ap_environment_t** ***env2***, *ap_dimchange_t*** ***dimchange1***, *ap_dimchange_t*** ***dimchange2***) [Function]

Least common environment to two enviroenments.

- Assume `ap_environment_is_eq(env1,env2)==false`
- If environments are not compatible (a variable has different types in the 2 environments), return NULL
- Compute also in *dimchange1* and *dimchange2* the conversion transformations to the lce.
- If no dimensions to add to *env1*, this implies that *env* is actually *env1*. In this case, `*dimchange1==NULL`. Otherwise, the function allocates the `*dimchange1` with `ap_dimchange_alloc`.

`ap_environment_t* ap_environment_lce_array` (*ap_environment_t*** ***tenv***, *size_t* ***size***, *ap_dimchange_t**** ***ptdimchange***) [Function]

Least common environment to an array of environments.

- Assume the size *size* of the array *tenv* is at least one;
- If all input environments are the same, `*ptdimchange==NULL`. Otherwise, compute in `*ptdimchange` the conversion permutations
- If no dimensions to add to `tenv[i]`, this implies that the result is actually `tenv[i]`. In this case, `(*ptdimchange)[i]==NULL`. Otherwise, the function allocates the `(*ptdimchange)[i]` with `ap_dimchange_alloc`.

`ap_environment_t* ap_environment_rename` (*ap_environment_t** ***env***, *ap_var_t** ***tvar1***, *ap_var_t** ***tvar2***, *size_t* ***size***, *ap_dimperm_t** ***perm***) [Function]

Rename the variables in the environment. *size* is the common size of arrays *tvar1* and *tvar2*, and *perm* is a result-parameter pointing to an *existing but not initialized* object of type `ap_dimperm_t`.

The function applies the variable substitution `tvar1[i]->tvar2[i]` to the environment, and returns the resulting environment and the allocated transformation permutation in `*perm`.

If the parameters are not valid, return `NULL` with `perm->dim==NULL`.

7.3 Linear expressions of level 1 (‘`ap_linexpr1.h`’)

We manipulate here expressions of the form

$$a_0.x_0 + ... + a_n.x_n + b$$

where the coefficients $a_0, ..., a_n, b$ are of `ap_coeff_t` type (either scalars or intervals) and the variables $x_0, ..., x_n$ are of type `ap_var_t`.

The semantics of linear expressions is exact, in the sense that the arithmetic operations are interpreted in the real (or rational) numbers. However, abstract domains are free to overapproximate this exact semantics (this may occur when converting rational scalars to `double` type for instance).

A special remark concerns integer variables. Abstract domains are assumed to perform the operations involving linear expressions using a real/rational semantics, and then possibly to reduce the result using the knowledge that integer variables may only take integer values.

> This semantics *coincides* with the natural integer semantics of expressions involving only integer variables *only if* the involved coefficients are all integers.
>
> A typical counter-example to this is an assignement $y := 1/3x$ where x and y are integer variables. If this assignement is applied to the Box abstract domain value

xin[1; 1], it may lead to the bottom value, because one will first obtain *yin*[1/3; 1/3] by real/rational computations, and this may be reduced to the empty interval because *y* is integer and the interval contains no integer values.

If you need expressions with a less simple semantics (mixing integer, real/rational and floating-point semantics with casts), you should use tree expressions (see Section 7.6 [Tree expressions of level 1], page 58).

`ap_linexpr1_t` [datatype]

(Internal) type of interval linear expressions.

Linear expressions of level 1 are created as objects of type `ap_linexpr1_t`, not as pointers of type `ap_linexpr1_t*`.

For information:

```
typedef struct ap_linexpr1_t {
  ap_linexpr0_t* linexpr0;
  ap_environment_t* env;
} ap_linexpr1_t;
```

7.3.1 Allocating linear expressions of level 1

`ap_linexpr1_t ap_linexpr1_make` (*ap_environment_t** **env**, *ap_linexpr_discr_t* **lin_discr**, *size_t* **size**) [Function]

Build a linear expressions on the environment *env*, with by default coefficients of type SCALAR and DOUBLE.

If *lin_discr* selects a dense representation, the size of the expression is the size of the environment. Otherwise, the initial size is given by *size* and the expression may be resized lazily.

`void ap_linexpr1_minimize` (*ap_linexpr1_t** **expr**) [Function]

Reduce the coefficients (transform intervals into scalars when possible). In case of sparse representation, also remove zero coefficients.

`ap_linexpr1_t ap_linexpr1_copy` (*ap_linexpr1_t** **expr**) [Function]

Duplication.

`void ap_linexpr1_clear` (*ap_linexpr1_t* **expr**) [Function]

Clear the linear expression.

`void ap_linexpr1_fprint` (*FILE** **stream**, *ap_linexpr1_t** **expr**) [Function]

Print the linear expression on stream *stream*.

7.3.2 Tests on linear expressions of level 1

`bool ap_linexpr1_is_integer` (*ap_linexpr1_t** **expr**) [Function]

Does the expression depends only on integer variables ?

`bool ap_linexpr1_is_real` (*ap_linexpr1_t** **expr**) [Function]

Does the expression depends only on real variables ?

`bool ap_linexpr1_is_linear` (*ap_linexpr1_t** **expr**) [Function]

Return true iff all involved coefficients are scalars.

`bool ap_linexpr1_is_quasilinear` (*ap_linexpr1_t** **expr**) [Function]

Return true iff all involved coefficients but the constant are scalars.

7.3.3 Access to linear expressions of level 1

`ap_environment_t* ap_linexpr1_envref` (*ap_linexpr1_t* expr*) [Function]

Get a reference to the underlying environment. Do not free it.

`size_t ap_linexpr1_linexpr0ref` (*ap_linexpr1_t* expr*) [Function]

Get a reference to the underlying linear expression of level 0. Do not free it.

7.3.3.1 Getting references

`ap_coefft* ap_linexpr1_cstref` (*ap_linexpr1_t* e*) [Function]

Get a reference to the constant. Do not free it.

`ap_coefft* ap_linexpr1_coeffref` (*ap_linexpr1_t* e, ap_var_t var*) [Function]

Get a reference to the coefficient associated to the variable *var* in expression *e*.

Do not free it. In case of sparse representation, possibly induce the addition of a new linear term.

Return NULL if *var* is unknown in the environment of *e*.

7.3.3.2 Getting values

`void ap_linexpr1_get_cst` (*ap_coefft* coeff, ap_linexpr1_t* e*) [Function]

Assign to *coeff* the constant coefficient of *e*.

`bool ap_linexpr1_get_coeff` (*ap_coefft* coeff, ap_linexpr1_t* e, ap_var_t var*) [Function]

Assign to *coeff* the coefficient of variable *var* in the expression *e*.

Return true in case `ap_linexpr1_coeffref(e,dim)` returns NULL.

`ap_linexpr1_ForeachLinterm` (*ap_linexpr1_t* e, size_t i, ap_ap_var_t var, ap_coeff_t* coeff*) [Macro]

Iterator on the coefficients associated to variables.

`ap_linexpr1_ForeachLinterm(E,I,VAR,COEFF){ body }` executes the body for each pair (*coeff*,*var*) in the expression *e*. *coeff* is a reference to the coefficient associated to variable *var* in *e*. *i* is an auxiliary variable used internally by the macro.

7.3.3.3 Assigning values with a list of arguments

`bool ap_linexpr1_set_list` (*ap_linexpr1_t* e, ...*) [Function]

This function assign the linear expression *e* from a list of tags of type `ap_coefftag_t`, each followed by a number of arguments as specified in the definition of the type `ap_coefftag_t` (see Section 8.2.3 [Access to linear expressions of level 0], page 75). The list should end with the tag `AP_COEFF_END`. The only difference with level 0 is that variables replace dimensions in the list.

Return `true` in case `ap_linexpr1_coeffref (e,dim)` returns NULL for one of the variables involved.

Here is a typical example, in the case where `ap_var_t` is actually `char*` (the default):

```
ap_linexpr1_set_list(e,
     AP_COEFF_S_INT, 3, "x",
     AP_COEFF_S_FRAC, 3,2, "y",
     AP_COEFF_S_DOUBLE, 4.1, "z",
```

```
          AP_CST_I_DOUBLE, -2.4, 3.6,
          AP_END); /* Do not forget the last tatg ! */
```

which transforms an null expression into `3 x + 3/2 y + 4.1 z + [-2.4,3.6]` and is equivalent to:

```
ap_linexpr1_set_coeff_scalar_int(e, "x", 3);
ap_linexpr1_set_coeff_scalar_frac(e, "y", 3,2);
ap_linexpr1_set_coeff_scalar_double(e, "z", 4.1);
ap_linexpr1_set_cst_interval_double(e, -2.4, 3.6);
```

7.3.3.4 Assigning values

void **ap_linexpr1_set_cst** (*ap_linexpr1_t* e, ap_coefft* coeff*) [Function]
void **ap_linexpr1_set_cst_scalar** (*ap_linexpr1_t* e, ap_scalar_t* scalar*) [Function]
void **ap_linexpr1_set_cst_scalar_int** (*ap_linexpr1_t* e, int num*) [Function]
void **ap_linexpr1_set_cst_scalar_frac** (*ap_linexpr1_t* e, int num, unsigned int den*) [Function]
void **ap_linexpr1_set_cst_scalar_double** (*ap_linexpr1_t* e, double num*) [Function]
void **ap_linexpr1_set_cst_interval** (*ap_linexpr1_t* e, ap_interval_t* itv*) [Function]
void **ap_linexpr1_set_cst_interval_scalar** (*ap_linexpr1_t* e, ap_scalar_t* inf, ap_scalar_t* sup*) [Function]
void **ap_linexpr1_set_cst_interval_int** (*ap_linexpr1_t* e, int inf, int sup*) [Function]
void **ap_linexpr1_set_cst_interval_frac** (*ap_linexpr1_t* e, int numinf, unsigned int deninf, int numsup, unsigned int densup*) [Function]
void **ap_linexpr1_set_cst_interval_double** (*ap_linexpr1_t* e, double inf, double sup*) [Function]

Set the constant coefficient of expression *e*.

bool **ap_linexpr1_set_coeff** (*ap_linexpr1_t* e, ap_var_t var, ap_coefft* coeff*) [Function]
bool **ap_linexpr1_set_coeff_scalar** (*ap_linexpr1_t* e, ap_var_t var, ap_scalar_t* scalar*) [Function]
bool **ap_linexpr1_set_coeff_scalar_int** (*ap_linexpr1_t* e, ap_var_t var, int num*) [Function]
bool **ap_linexpr1_set_coeff_scalar_frac** (*ap_linexpr1_t* e, ap_var_t var, int num, unsigned int den*) [Function]
bool **ap_linexpr1_set_coeff_scalar_double** (*ap_linexpr1_t* e, ap_var_t var, double num*) [Function]
bool **ap_linexpr1_set_coeff_interval** (*ap_linexpr1_t* e, ap_var_t var, ap_interval_t* itv*) [Function]
bool **ap_linexpr1_set_coeff_interval_scalar** (*ap_linexpr1_t* e, ap_var_t var, ap_scalar_t* inf, ap_scalar_t* sup*) [Function]
bool **ap_linexpr1_set_coeff_interval_int** (*ap_linexpr1_t* e, ap_var_t var, int inf, int sup*) [Function]
bool **ap_linexpr1_set_coeff_interval_frac** (*ap_linexpr1_t* e, ap_var_t var,int numinf, unsigned int deninf, int numsup, unsigned int densup*) [Function]

void ap_linexpr1_set_coeff_interval_double (*ap_linexpr1_t** **e**, *ap_var_t* **var**, *double* **inf**, *double* **sup**) [Function]

Set the coefficient of the variable *var* of expression *e*.

Return `true` in case `ap_linexpr1_coeffref(e,var)` returns NULL.

7.3.4 Change of dimensions and permutations of linear expressions of level 1

bool ap_linexpr1_extend_environment (*ap_linexpr1_t** **nexpr**, *ap_linexpr1_t** **expr**, *ap_environment_t** **nenv**) [Function]

bool ap_linexpr1_extend_environment_with (*ap_linexpr1_t** **expr**, *ap_environment_t** **nenv**) [Function]

Change the current environment of the expression *expr* with a super-environment *nenv*. Return `true` if *nenv* is not a superenvironment.

The first version store the result in the uninitialized `*nexpr`, the second one updates in-place its argument.

7.4 Linear constraints of level 1 (‘ap_lincons1.h’)

ap_lincons1_t [datatype]

Datatype for constraints.

For information:

```
typedef struct ap_lincons1_t {
  ap_lincons0_t lincons0;
  ap_environment_t* env;
} ap_lincons1_t;
```

Constraints are meant to be manipulated freely via their components. Creating the constraint `[1,2]x + 5/2 y >=0` and then freeing it can be done with

```
ap_lincons1_t cons = ap_lincons1_make(AP_CONS_SUPEQ,
      ap_linexpr1_alloc(env,AP_LINEXPR_SPARSE,2),
                                NULL);
ap_lincons1_set_list(&cons,
     AP_COEFF_I_INT, 1,2, "x",
     AP_COEFF_S_FRAC, 5,2, "y",
     AP_END);
ap_lincons1_clear(&cons);
```

ap_lincons1_array_t [datatype]

```
typedef struct ap_lincons1_array_t {
  ap_lincons0_array_t lincons0_array;
  ap_environment_t* env;
} ap_lincons1_array_t;
```

Datatype for arrays of constraints.

Arrays at level 1 cannot be accessed directly, for example by writing `array->p[i]`, but should instead be accessed with functions `ap_lincons1_array_get` and `ap_lincons1_array_set`.

7.4.1 Allocating linear constraints of level 1

`ap_lincons1_t ap_lincons1_make` (*ap_constyp_t* **`constyp`**, *ap_linexpr1_t** **`linexpr`**, *ap_scalar_t* mod*) [Function]

Create a constraint of type *constyp* with the expression *linexpr*, and the modulo *mod* in case of a congruence constraint (`constyp==AP_CONS_EQMOD`).

The expression is not duplicated, just pointed to, so it becomes managed via the constraint.

`ap_lincons1_t ap_lincons1_make_unsat` (*ap_environment_t** **`env`**) [Function]

Create the constraint `-1>=0`.

`ap_lincons1_t ap_lincons1_copy` (*ap_lincons1_t** **`cons`**) [Function]

Duplication

`void ap_lincons1_clear` (*ap_lincons1_t** **`cons`**) [Function]

Clear the constraint and set pointers to NULL.

`void ap_lincons1_fprint` (*FILE** **`stream`**, *ap_lincons1_t** **`cons`**); [Function]

Print the linear constraint on stream *stream*.

7.4.2 Tests on linear constraints of level 1

`bool ap_lincons1_is_unsat` (*ap_lincons1_t** **`cons`**) [Function]

Return `true` if the constraint is not satisfiable (`b>=0` or `[a,b]>=0` with b negative).

7.4.3 Access to linear constraints of level 1

`ap_environment_t* ap_lincons1_envref` (*ap_lincons1_t** **`cons`**) [Function]

Get a reference to the environment. Do not free it.

`ap_constyp_t* ap_lincons1_constypref` (*ap_lincons1_t** **`cons`**) [Function]

Get a reference to the type of constraint. You may use the reference to modify the constraint type.

`ap_linexpr1_t ap_lincons1_linexpr1ref` (*ap_lincons1_t** **`cons`**) [Function]

Get a reference to the underlying expression of the constraint. Do not free it: nothing is duplicated. Modifying the argument or the result is equivalent, except for change of dimensions/environment.

`void ap_lincons1_get_cst` (*ap_coeff_t** **`coeff`**, *ap_lincons1_t** **`cons`**) [Function]
`void ap_lincons1_set_cst` (*ap_lincons1_t** **`cons`**, *ap_coeff_t* cst*) [Function]
`bool ap_lincons1_get_coeff` (*ap_coeff_t** **`coeff`**, *ap_lincons1_t* cons*, *ap_var_t* **`var`**) [Function]
`bool ap_lincons1_set_coeff` (*ap_lincons1_t** **`cons`**, *ap_var_t* **`var`**, *ap_coeff_t** **`coeff`**) [Function]
`bool ap_lincons1_set_list` (*ap_lincons1_t** **`cons`**, ...) [Function]
`ap_coeff_t* ap_lincons1_cstref` (*ap_lincons1_t** **`cons`**) [Function]
`ap_coeff_t* ap_lincons1_coeffref` (*ap_lincons1_t** **`cons`**, *ap_var_t* **`var`**) [Function]

Identical to corresponding `ap_linexpr1_XXX` functions (see Section 7.3.3 [Access to linear expressions of level 1], page 51).

`ap_lincons0_t* ap_lincons1_lincons0ref` (*ap_lincons1_t** **`cons`**) [Function]

Return underlying constraint of level 0. Do not free it: nothing is duplicated. Modifying the argument or the result is equivalent, except for change of dimensions/envionment.

7.4.4 Change of dimensions and permutations of linear constraints of level 1

`bool ap_lincons1_extend_environment` (*ap_lincons1_t** **ncons**, *ap_lincons1_t** **cons**, *ap_environment_t** **nenv**) [Function]

`bool ap_lincons1_extend_environment_with` (*ap_lincons1_t** **cons**, *ap_environment_t** **nenv**) [Function]

Identical to corresponding `ap_linexpr1_XXX` functions (see Section 7.3.4 [Change of dimensions and permutations of linear expressions of level 1], page 53).

7.4.5 Arrays of linear constraints of level 1

`ap_lincons1_array_t ap_lincons1_array_make` (*ap_environment_t** **env**, *size_t* **size**) [Function]

Allocate an array of constraints of size *size*, defined on the environment *env*.

The constraints are initialized with NULL pointers for underlying expressions.

`void ap_lincons1_array_clear` (*ap_lincons1_array_t** **array**) [Function]

Clear the constraints of the array, and then the array itself.

`void ap_lincons1_array_fprint` (*FILE** **stream**, *ap_lincons1_array_t** **array**) [Function]

Print the array on the stream.

`size_t ap_lincons1_array_size` (*ap_lincons1_array_t** **array**) [Function]

Return the size of the array.

`ap_environment_t* ap_lincons1_array_envref` (*ap_lincons1_array_t** **array**) [Function]

Get a reference to the environment. Do not free it.

`ap_lincons1_t ap_lincons1_array_get` (*ap_lincons1_array_t** **array**, *size_t* **index**) [Function]

Return the linear constraint of the given index. Nothing is duplicated, and the result should never be cleared. Modifying the argument or the result is equivalent, except for change of environments

`bool ap_lincons1_array_set` (*ap_lincons1_array_t** **array**, *size_t* **index**, *ap_lincons1_t** **cons**) [Function]

Fill the index of the array with the constraint. Assumes `ap_environment_is_eq(array->env,cons->env)`. Nothing is duplicated. The argument should never be cleared (its environment is dereferenced). If a constraint was already stored, it is first cleared. Return true iff problem (index or `array->env!=cons->env`)

`void ap_lincons1_array_clear_index` (*ap_lincons1_array_t** **array**, *size_t* **index**) [Function]

Clear the constraint at index index.

`bool ap_lincons1_array_extend_environment_with` (*ap_lincons1_array_t** **array**, *ap_environment_t** **nenv**) [Function]

`bool ap_lincons1_array_extend_environment` (*ap_lincons1_array_t** *narray*, *ap_lincons1_array_t** **array**, *ap_environment_t** **nenv**) [Function]

Identical to corresponding `ap_linexpr1_XXX` functions (see Section 7.3.4 [Change of dimensions and permutations of linear expressions of level 1], page 53).

7.5 generators of level 1 (‘ap_generator1.h’)

`ap_generator1_t` [datatype]

Datatype for generators.

For information:

```
typedef struct ap_generator1_t {
  ap_generator0_t generator0;
  ap_environment_t* env;
} ap_generator1_t;
```

Generators are meant to be manipulated freely via their components. Creating the ray generator `x+2/3y` and then freeing it can be done with

```
ap_generator1_t gen = ap_generator1_make(AP_GEN_RAY,
      ap_linexpr1_alloc(env,AP_LINEXPR_SPARSE,2));
ap_generator1_set_list(&gen,
     AP_COEFF_S_INT, 1, "x",
     AP_COEFF_S_FRAC, 2,3, "y",
     AP_END);
ap_generator1_clear(&gen);
```

`ap_generator1_array_t` [datatype]

```
typedef struct ap_generator1_array_t {
  ap_generator0_array_t generator0_array;
  ap_environment_t* env;
} ap_generator1_array_t;
```

Datatype for arrays of generators.

Arrays at level 1 cannot be accessed directly, for example by writing `array->p[i]`, but should instead be accessed with functions `ap_generator1_array_get` and `ap_generator1_array_set`.

7.5.1 Allocating generators of level 1

`ap_generator1_t ap_generator1_make` (*ap_gentyp_t* ***gentyp***, *ap_linexpr1_t** ***linexpr***) [Function]

Create a generator of type *gentyp* with the expression *linexpr*.

The expression is not duplicated, just pointed to, so it becomes managed via the generator.

`ap_generator1_t ap_generator1_copy` (*ap_generator1_t** ***gen***) [Function]

Duplication

`void ap_generator1_clear` (*ap_generator1_t** ***gen***) [Function]

Clear the generator and set pointers to NULL.

`void ap_generator1_fprint` (*FILE** ***stream***, *ap_generator1_t** ***gen***)*;* [Function]

Print the linear generator on stream *stream*.

7.5.2 Access to generators of level 1

`ap_environment_t* ap_generator1_envref` (*ap_generator1_t** ***gen***) [Function]

Get a reference to the environment. Do not free it.

`ap_gentyp_t* ap_generator1_gentypref` (*ap_generator1_t** `gen`) [Function]
Get a reference to the type of generator. You may use the reference to modify the generator type.

`ap_linexpr1_t ap_generator1_linexpr1ref` (*ap_generator1_t** `gen`) [Function]
Get a reference to the underlying expression of the generator. Do not free it: nothing is duplicated. Modifying the argument or the result is equivalent, except for change of dimensions/environment.

`void ap_generator1_get_cst` (*ap_coeff_t** `coeff`, *ap_generator1_t** `gen`) [Function]
`void ap_generator1_set_cst` (*ap_generator1_t** `gen`, *ap_coeff_t** *cst*) [Function]
`bool ap_generator1_get_coeff` (*ap_coeff_t** `coeff`, *ap_generator1_t** *gen*, *ap_var_t* `var`) [Function]
`bool ap_generator1_set_coeff` (*ap_generator1_t** `gen`, *ap_var_t* `var`, *ap_coeff_t** `coeff`) [Function]
`bool ap_generator1_set_list` (*ap_generator1_t** `gen`, ...) [Function]
`ap_coeff_t* ap_generator1_cstref` (*ap_generator1_t** `gen`) [Function]
`ap_coeff_t* ap_generator1_coeffref` (*ap_generator1_t** `gen`, *ap_var_t* `var`) [Function]
Identical to corresponding `ap_linexpr1_XXX` functions (see Section 7.3.3 [Access to linear expressions of level 1], page 51).

`ap_generator0_t* ap_generator1_generator0ref` (*ap_generator1_t** `gen`) [Function]
Return underlying generator of level 0. Do not free it: nothing is duplicated. Modifying the argument or the result is equivalent, except for change of dimensions/envionment.

7.5.3 Change of dimensions and permutations of generators of level 1

`bool ap_generator1_extend_environment` (*ap_generator1_t** `ngen`, *ap_generator1_t** `gen`, *ap_environment_t** `nenv`) [Function]
`bool ap_generator1_extend_environment_with` (*ap_generator1_t** `gen`, *ap_environment_t** `nenv`) [Function]
Identical to corresponding `ap_linexpr1_XXX` functions (see Section 7.3.4 [Change of dimensions and permutations of linear expressions of level 1], page 53).

7.5.4 Arrays of generators of level 1

`ap_generator1_array_t ap_generator1_array_make` (*ap_environment_t** `env`, *size_t* `size`) [Function]
Allocate an array of generators of size *size*, defined on the environment *env*.

The generators are initialized with NULL pointers for underlying expressions.

`void ap_generator1_array_clear` (*ap_generator1_array_t** `array`) [Function]
Clear the generators of the array, and then the array itself.

`void ap_generator1_array_fprint` (*FILE** `stream`, *ap_generator1_array_t** `array`) [Function]
Print the array on the stream.

`size_t ap_generator1_array_size` (*ap_generator1_array_t** `array`) [Function]
Return the size of the array.

`ap_environment_t* ap_generator1_array_envref` *(ap_generator1_array_t* array)* [Function]

Get a reference to the environment. Do not free it.

`ap_generator1_t ap_generator1_array_get` *(ap_generator1_array_t* array, size_t index)* [Function]

Return the linear generator of the given index Nothing is duplicated, and the result should never be cleared. Modifying the argument or the result is equivalent, except for change of environments

`bool ap_generator1_array_set` *(ap_generator1_array_t* array, size_t index, ap_generator1_t* gen)* [Function]

Fill the index of the array with the generator. Assumes `array->env==gen->env`. Nothing is duplicated. The argument should never be cleared. (its environment is dereferenced). If a generator was already stored, it is first cleared. Return true iff problem (index or `array->env!=gen->env`)

`void ap_generator1_array_clear_index` *(ap_generator1_array_t* array, size_t index)* [Function]

Clear the generator at index index.

`bool ap_generator1_array_extend_environment_with` *(ap_generator1_array_t* array, ap_environment_t* nenv)* [Function]

`bool ap_generator1_array_extend_environment` *(ap_generator1_array_t* narray, ap_generator1_array_t* array, ap_environment_t* nenv)* [Function]

Identical to corresponding `ap_linexpr1_XXX` functions (see Section 7.3.4 [Change of dimensions and permutations of linear expressions of level 1], page 53).

7.6 Tree expressions of level 1 ('`ap_texpr1.h`')

We manipulate here general expressions described by the grammar

$$expr ::= cst|var|unop e|e1 binop e2$$

Such tree expressions generalize linear expressions(see Section 7.3 [Linear expressions of level 1], page 49) in two ways:

- Non-linear operations are possible (multiplication, division, casts, ...)
- Semantics of operators is no longer restricted to real/rational semantics. Each operation is parameterized by two parameters:
  - a rounding type parameter, which indicates the destination type of the operation, and influences how the rounding is performed;
  - a rounding direction parameter, which defines the rounding mode.

7.6.1 Datatypes for tree expressions of level 1

`ap_texpr1_t` [datatype]

Type of tree expressions.

Tree expressions of level 1 are created as objects of type `ap_texpr1_t*`. They are manipulated in a functional way (except a few operations), unlike linear expressions on which most operations acts by side-effects.

`ap_texpr_op_t` [datatype]

Operators (actually defined in 'ap_texpr0.h')

```
typedef enum ap_texpr_op_t {
  /* Binary operators */
  AP_TEXPR_ADD, AP_TEXPR_SUB, AP_TEXPR_MUL, AP_TEXPR_DIV,
  AP_TEXPR_MOD,  /* either integer or real, no rounding */
  /* Unary operators */
  AP_TEXPR_NEG   /* no rounding */,
  AP_TEXPR_CAST, AP_TEXPR_SQRT,
} ap_texpr_op_t;
```

`ap_texpr_rtype_t` [datatype]

Destination type of the operation for rounding (actually defined in 'ap_texpr0.h')

```
typedef enum ap_texpr_rtype_t {
  AP_RTYPE_REAL,     /* real (no rounding) */
  AP_RTYPE_INT,      /* integer */
  AP_RTYPE_SINGLE,   /* IEEE 754 32-bit single precision, e.g.: C's float */
  AP_RTYPE_DOUBLE,   /* IEEE 754 64-bit double precision, e.g.: C's double */
  AP_RTYPE_EXTENDED, /* non-standard 80-bit double extended, e.g.: Intel's long doub
  AP_RTYPE_QUAD,     /* non-standard 128-bit quadruple precision, e.g.: Motorola's l
} ap_texpr_rtype_t;
```

`ap_texpr_rdir_t` [datatype]

Rounding mode (actually defined in 'ap_texpr0.h')

```
typedef enum ap_texpr_rdir_t {
  AP_RDIR_NEAREST /* Nearest */
  AP_RDIR_ZERO    /* Zero (truncation for integers) */
  AP_RDIR_UP      /* + Infinity */
  AP_RDIR_DOWN    /* - Infinity */
  AP_RDIR_RND,    /* All possible mode, non deterministically */
  AP_RDIR_SIZE    /* Not to be used ! */
} ap_texpr_rdir_t;
```

7.6.2 Constructors/Destructors for tree expressions of level 1

Parameters of constructors are not memory-managed by the constructed expression, with the important exception of expressions parameters (type `ap_texpr1.h`) are, which means that they should not be freed afterwards.

`ap_texpr1_t* ap_texpr1_cst` (*ap_environment_t* env*, *ap_coeff_t* coeff*) [Function]
`ap_texpr1_t* ap_texpr1_cst_scalar` (*ap_environment_t* env*, *ap_scalar_t* scalar*) [Function]
`ap_texpr1_t* ap_texpr1_cst_scalar_mpq` (*ap_environment_t* env*, *mpq_t mpq*) [Function]
`ap_texpr1_t* ap_texpr1_cst_scalar_int` (*ap_environment_t* env*, *long int num*) [Function]
`ap_texpr1_t* ap_texpr1_cst_scalar_frac` (*ap_environment_t* env*, *long int num*, *unsigned long int den*) [Function]
`ap_texpr1_t* ap_texpr1_cst_scalar_double` (*ap_environment_t* env*, *double num*) [Function]

`ap_texpr1_t* ap_texpr1_cst_interval` (*ap_environment_t* env*, *ap_interval_t* itv*) [Function]

`ap_texpr1_t* ap_texpr1_cst_interval_scalar` (*ap_environment_t* env*, *ap_scalar_t* inf*, *ap_scalar_t* sup*) [Function]

`ap_texpr1_t* ap_texpr1_cst_interval_mpq` (*ap_environment_t* env*, *mpq_t inf*, *mpq_t sup*) [Function]

`ap_texpr1_t* ap_texpr1_cst_interval_int` (*ap_environment_t* env*, *long int inf*, *long int sup*) [Function]

`ap_texpr1_t* ap_texpr1_cst_interval_frac` (*ap_environment_t* env*, *long int numinf*, *unsigned long int deninf*, *long int numsup*, *unsigned long int densup*) [Function]

`ap_texpr1_t* ap_texpr1_cst_interval_double` (*ap_environment_t* env*, *double inf*, *double sup*) [Function]

`ap_texpr1_t* ap_texpr1_cst_top` (*ap_environment_t* env*) [Function]

Create a constant expression, on the environment *env*.

`ap_texpr1_t* ap_texpr1_var` (*ap_environment_t* env*, *ap_var_t var*) [Function]

Create a variable expression. Return `NULL` in the case *var* is unknown in *env*.

`ap_texpr1_t* ap_texpr1_unop` (*ap_texpr_op_t op*, *ap_texpr1_t* opA*, *ap_texpr_rtype_t type*, *ap_texpr_rdir_t dir*) [Function]

`ap_texpr1_t* ap_texpr1_binop` (*ap_texpr_op_t op*, *ap_texpr1_t* opA*, *ap_texpr1_t* opB*, *ap_texpr_rtype_t type*, *ap_texpr_rdir_t dir*) [Function]

Create an expression from an operator and expression operand(s). Be aware that *opA* and *opB* are memroy-managed by the result upon return.

`ap_texpr1_t* ap_texpr1_copy` (*ap_texpr1_t* expr*) [Function]

(Deep) copy of a tree expression.

`ap_texpr1_t* ap_texpr1_from_linexpr1` (*ap_linexpr1_t* linexpr*) [Function]

Creation from a linear expression.

`void ap_texpr1_free` (*ap_texpr1_t* expr*) [Function]

Free (recursively) a tree expression.

`void ap_texpr1_fprint` (*FILE* stream*, *ap_texpr1_t* e*) [Function]

`void ap_texpr1_print` (*ap_texpr1_t* e*) [Function]

Print the expression

7.6.3 Tests on tree expressions of level 1

`bool ap_texpr1_equal` (*ap_texpr1_t* e1*, *ap_texpr1_t* e2*) [Function]

Structural (recursive) equality

`bool ap_texpr1_has_var` (*ap_texpr1_t* e*, *ap_var_t var*) [Function]

Return true if variable *var* appears in the expression.

The next functions classifies tree expressions.

`bool ap_texpr1_is_interval_cst` (*ap_texpr1_t* e*) [Function]

No variable, only constant leaves

`bool ap_texpr1_is_interval_linear` (*ap_texpr1_t* e*) [Function]

Linear with possibly interval coefficients, no rounding

`bool ap_texpr1_is_interval_polynomial` (*ap_texpr1_t* e*) [Function]
Polynomial with possibly interval coefficients, no rounding

`bool ap_texpr1_is_interval_polyfrac` (*ap_texpr1_t* e*) [Function]
Polynomial fraction with possibly interval coefficients, no rounding

`bool ap_texpr1_is_scalar` (*ap_texpr1_t* e*) [Function]
All coefficients are scalar (non-interval)

7.6.4 Operations on tree expressions of level 1

`ap_texpr1_t* ap_texpr1_substitute` (*ap_texpr1_t* e, ap_var_t var, ap_texpr1_t* dst*) [Function]
Substitute every occurence of variable var with a copy of dst. Return NULL in case of incorrect argument (w.r.t. var and/or environment compatibility).

`ap_texpr1_t* ap_texpr1_extend_environment` (*ap_texpr1_t* e, ap_environment_t* nenv*) [Function]
Change current environment with a super-environment. Return NULL if *nenv* is not a superenvironment of `e->env`.

`bool ap_texpr1_substitute_with` (*ap_texpr1_t* e, ap_var_t var, ap_texpr1_t* dst*) [Function]
`bool ap_texpr1_extend_environment_with` (*ap_texpr1_t* e, ap_environment_t* nenv*) [Function]
Side-effect versions of the previous functions. Return `true` instead of `NULL` in case of problem.

7.7 Tree constraints of level 1 ('ap_tcons1.h')

Tree constraints are constraints built on tree expressions.

7.7.1 Datatypes for tree constraints of level 1

`ap_tcons1_t` [datatype]
Datatype for constraints.

For information:

```
typedef struct ap_tcons1_t {
  ap_tcons0_t tcons0;
  ap_environment_t* env;
} ap_tcons1_t;
```

`ap_tcons1_array_t` [datatype]

```
typedef struct ap_tcons1_array_t {
  ap_tcons0_array_t tcons0_array;
  ap_environment_t* env;
} ap_tcons1_array_t;
```

Datatype for arrays of constraints.

Arrays at level 1 cannot be accessed directly, for example by writing `array->p[i]`, but should instead be accessed with functions `ap_tcons1_array_get` and `ap_tcons1_array_set`.

7.7.2 Constructors/Destructors for tree constraints of level 1

`ap_tcons1_t ap_tcons1_make` (*ap_constyp_t* `constyp`, *ap_texpr1_t** `expr`, *ap_scalar_t** `scalar`) [Function]

Create a constraint of given type with the given expression. The expression and the optional coefficient are not duplicated, just pointed to.

`ap_tcons1_t ap_tcons1_from_lincons1` (*ap_tcons1_t** `cons`) [Function]

Create a tree constraint from a linear constraint.

`ap_tcons1_t ap_tcons1_copy` (*ap_tcons1_t** `cons`) [Function]

Duplication.

`void ap_tcons1_clear` (*ap_tcons1_t** `cons`) [Function]

Clear the constraint and set pointers to NULL.

`void ap_tcons1_fprint` (*FILE** `stream`, *ap_tcons1_t** `cons`) [Function]
`void ap_tcons1_print` (*ap_tcons1_t** `cons`) [Function]

Printing

`ap_environment_t* ap_tcons1_envref` (*ap_tcons1_t** `cons`) [Function]

Get a reference to the environment. Do not free it.

`ap_constyp_t* ap_tcons1_constypref` (*ap_tcons1_t** `cons`) [Function]

Get a reference to the type of constraint.

`ap_scalar_t* ap_tcons1_scalarref` (*ap_tcons1_t** `cons`) [Function]

Get a reference to the auxiliary coefficient of the constraint.

`ap_texpr1_t ap_tcons1_texpr1ref` (*ap_tcons1_t** `cons`) [Function]

Get a reference to the underlying expression of the constraint. Do not free it: nothing is duplicated. Modifying the argument or the result is equivalent, except for change of dimensions/environment.

`ap_tcons0_t* ap_tcons1_tcons0ref` (*ap_tcons1_t** `cons`) [Function]

Return underlying constraint of level 0. Do not free it: nothing is duplicated. Modifying the argument or the result is equivalent, except for change of dimensions/envionment.

7.7.3 Operations on tree constraints of level 1

`bool ap_tcons1_extend_environment` (*ap_tcons1_t** `ncons`, *ap_tcons1_t** `cons`, *ap_environment_t** `nenv`) [Function]
`bool ap_tcons1_extend_environment_with` (*ap_tcons1_t** `cons`, *ap_environment_t** `nenv`) [Function]

Change current environment with a super-environment. Return `true` if *nenv* is not a superenvironment of `e->env`.

7.7.4 Arrays of tree constraints of level 1

`ap_tcons1_array_t ap_tcons1_array_make` (*ap_environment_t** `env`, *size_t* `size`) [Function]

Allocate an array of size constraints. The constraints are initialized with NULL pointers, so that `ap_tcons1_array_free` may be safely called.

`void ap_tcons1_array_clear` (*ap_tcons1_array_t* array*) [Function]

Clear the constraints of the array, and then the array itself.

`void ap_tcons1_array_fprint` (*FILE* stream, ap_tcons1_array_t* array*) [Function]
`void ap_tcons1_array_print` (*ap_tcons1_array_t* array*) [Function]

Printing.

`size_t ap_tcons1_array_size` (*ap_tcons1_array_t* array*) [Function]

Return the size of the array.

`ap_environment_t* ap_tcons1_array_envref` (*ap_tcons1_array_t* array*) [Function]

Return a reference to the environment of the array. Do not free it.

`void ap_tcons1_array_clear_index` (*ap_tcons1_array_t* array, size_t index*) [Function]

Clear the constraint at index index and set pointers to `NULL`.

`ap_tcons1_t ap_tcons1_array_get` (*ap_tcons1_array_t* array, size_t index*) [Function]

Return the linear constraint of the given index Nothing is duplicated, and the result should never be cleared. Modifying the argument or the result is equivalent, except for change of environments.

`bool ap_tcons1_array_set` (*ap_tcons1_array_t* array, size_t index, ap_tcons1_t* cons*) [Function]

Fill the index of the array with the constraint. Assumes `ap_environment_is_eq(array->env,cons->env)`. Nothing is duplicated. The argument should never be cleared (its environment is dereferenced). If a constraint was already stored, it is first cleared. Return `true` iff problem (index or `array->env!=cons->env`)

`bool ap_tcons1_array_extend_environment_with` (*ap_tcons1_array_t* array, ap_environment_t* nenv*) [Function]
`bool ap_tcons1_array_extend_environment` (*ap_tcons1_array_t* narray, ap_tcons1_array_t* array, ap_environment_t* nenv*) [Function]

Change current environment with a super-environment. Return `true` if *nenv* is not a superenvironment of `array->env`.

7.8 Abstract values and operations of level 1 (‘ap_abstract1.h’)

`ap_abstract1_t` [datatype]

Datatype for abstract values at level 1.

For information:

```
typedef struct ap_abstract1_t {
  ap_abstract0_t* abstract0;
  ap_environment_t* env;
} ap_abstract1_t;
  /* data structure invariant:
     ap_abstract0_integer_dimension(man,abstract0)== env->intdim &&
     ap_abstract0_real_dimension(man,abstract0)== env->realdim */
```

ap_box1_t [datatype]

```
typedef struct ap_box1_t {
  ap_interval_t** p;
  ap_environment_t* env;
} ap_box1_t;
void ap_box1_fprint(FILE* stream, ap_box1_t* box);
void ap_box1_clear(ap_box1_t* box);
```

Most operations are offered in 2 versions: *functional* or *destructive* See Section 8.7 [Abstract values and operations of level 0], page 82.

We remind the policy for redimensioning (see Chapter 7 [Level 1 of the interface], page 45):

- For functions taking one abstract value and one expression (or constraint or generator, or array of ...), the environment of the expression should be a sub-environment of the environment of the abstract value. The environment of the result is the environment of the argument abstract value.
- For functions taking several abstract values, their environments should be the same. Otherwise, it is up to the user to move them to a common super-environment (see Section 7.2 [Environments], page 46).

7.8.1 Allocating abstract values of level 1

`ap_abstract1_t ap_abstract1_copy` (*ap_manager_t* man*, *ap_abstract1_t* a*) [Function]

Return a copy of *a*, on which destructive update does not affect *a*.

`void ap_abstract1_clear` (*ap_manager_t* man*, *ap_abstract1_t* a*) [Function]

Free all the memory used by *a*.

`size_t ap_abstract1_size` (*ap_manager_t* man*, *ap_abstract1_t* a*) [Function]

Return the abstract size of *a*.

7.8.2 Control of internal representation of abstract values of level 1

`void ap_abstract1_minimize` (*ap_manager_t* man*, *ap_abstract1_t* a*) [Function]

Minimize the size of the representation of *a*. This may result in a later recomputation of internal information.

`void ap_abstract1_canonicalize` (*ap_manager_t* man*, *ap_abstract1_t* a*) [Function]

Put *a* in canonical form. (not yet clear definition).

`int ap_abstract1_hash` (*ap_manager_t* man*, *ap_abstract1_t* a*) [Function]

Return an hash value for *a*. Two abstract values in canonical from (according to `ap_abstract1_canonicalize`) and considered as equal by the function `ap_abstract1_is_eq` are given the same hash value.

`void ap_abstract1_approximate` (*ap_manager_t* man*, *ap_abstract1_t* a*, *int algorithm*) [Function]

Perform some transformation on *a*, guided by the field algorithm.

The transformation may lose information. The argument *algorithm* overrides the field algorithm of the structure of type `ap_funopt_t` associated to `ap_abstract1_approximate`.

7.8.3 Printing abstract values of level 1

`void ap_abstract1_fprint` (*FILE* stream*, *ap_manager_t* man*, *ap_abstract1_t* a*) [Function]

Print *a* in a pretty way on the stream.

`void ap_abstract1_fprintdiff` (*FILE* stream*, *ap_manager_t* man*, *ap_abstract1_t* a1*, *ap_abstract1_t* a2*) [Function]

Print the difference between *a1* (old value) and *a2* (new value). The meaning of difference is library dependent.

`void ap_abstract1_fdump` (*FILE* stream*, *ap_manager_t* man*, *ap_abstract1_t* a*) [Function]

Dump the internal representation of *a* for debugging purposes.

7.8.4 Serialization of abstract values of level 1

`ap_membuf_t ap_abstract1_serialize_raw` (*ap_manager_t* man*, *ap_abstract1_t* a*) [Function]

Allocate a memory buffer (with `malloc`), output *a* in raw binary format to it and return a pointer on the memory buffer and the number of bytes written. It is the user responsability to free the memory afterwards (with `free`).

`ap_abstract1_t ap_abstract1_deserialize_raw` (*ap_manager_t* man*, *void* ptr*, *size_t* size*) [Function]

Return the abstract value read in raw binary format from the buffer pointed by *ptr* and store in size the number of bytes read.

7.8.5 Constructors for abstract values of level 1

`ap_abstract1_t ap_abstract1_bottom` (*ap_manager_t* man*, *ap_environment_t* env*) [Function]

`ap_abstract1_t ap_abstract1_top` (*ap_manager_t* man*, *ap_environment_t* env*) [Function]

Create resp. a bottom (empty) value and a top (universe) value defined on the environment *env*.

`ap_abstract1_t ap_abstract1_of_box` (*ap_manager_t* man*, *ap_environment_t* env*, *ap_var_t* tvar*, *ap_interval_t** tinterval*, *size_t size*) [Function]

Abstract an hypercube defined by the arrays *tvar* and *tintnerval* of size *size*.

If no inclusion is specified for a variable in the environment, its value is no constrained in the resulting abstract value.

7.8.6 Accessors for abstract values of level 1

`ap_dimension_t ap_abstract1_environment` (*ap_manager_t* man*, *ap_abstract1_t* a*) [Function]

Get a reference to the environment of *a*. Do not free it.

`ap_manager_t* ap_abstract1_manager` (*ap_abstract1_t* a*) [Function]

Get a reference to the manager contained in *a*. Do not free it.

`ap_dimension_t ap_abstract1_abstract0` (*ap_manager_t** `man`, *ap_abstract1_t** `a`) [Function]

Get a reference to the underlying abstract value of level 0 in *a*. Do not free it.

7.8.7 Tests on abstract values of level 1

In abstract tests,

- true means that the predicate is certainly true;
- false means false *or* don't know (an exception has occurred, or the exact computation was considered too expensive to be performed, according to the options).

`bool ap_abstract1_is_bottom` (*ap_manager_t** `man`, *ap_abstract1_t** `a`) [Function]
`bool ap_abstract1_is_top` (*ap_manager_t** `man`, *ap_abstract1_t** `a`) [Function]

Emtpiness and universality tests.

`bool ap_abstract1_is_leq` (*ap_manager_t** `man`, *ap_abstract1_t** `a1`, *ap_abstract1_t** `a2`) [Function]
`bool ap_abstract1_is_eq` (*ap_manager_t** `man`, *ap_abstract1_t** `a1`, *ap_abstract1_t** `a2`) [Function]

Inclusion and equality tests.

`bool ap_abstract1_sat_interval` (*ap_manager_t** `man`, *ap_abstract1_t** `a`, *ap_var_t* `var`, *ap_interval_t** `interval`) [Function]

Is the variable *var* included in the interval *interval* in the abstract value *a* ?

`bool ap_abstract1_sat_lincons` (*ap_manager_t** `man`, *ap_abstract1_t** `a`, *ap_lincons1_t** `cons`) [Function]
`bool ap_abstract1_sat_tcons` (*ap_manager_t** `man`, *ap_abstract1_t** `a`, *ap_tcons1_t** `cons`) [Function]

Does the abstract value *a* satisfy the constraint *cons* ?

`bool ap_abstract1_is_variable_unconstrained` (*ap_manager_t** `man`, *ap_abstract1_t** `a`, *ap_var_t* `var`) [Function]

Is the dimension *dim* unconstrained in the abstract value *a* ? If it is the case, we have `forget(man,a,dim) == a`.

7.8.8 Extraction of properties of abstract values of level 1

`ap_interval_t* ap_abstract1_bound_variable` (*ap_manager_t** `man`, *ap_abstract1_t** `a`, *ap_var_t* `var`) [Function]

Return the interval taken by the variable *var* over the abstract value *a*.

`ap_interval_t* ap_abstract1_bound_linexpr` (*ap_manager_t** `man`, *ap_abstract1_t** `a`, *ap_linexpr1_t** `expr`) [Function]
`ap_interval_t* ap_abstract1_bound_texpr` (*ap_manager_t** `man`, *ap_abstract1_t** `a`, *ap_texpr1_t** `expr`) [Function]

Return the interval taken by the expression *expr* over the abstract value *a*.

In the case of truly linear expression, this function allows to solve a Linear Programming (LP) problem, but depending on the underlying domain the solution may be not optimal.

`ap_box1_t ap_abstract1_to_box` (*ap_manager_t** `man`, *ap_abstract1_t** `a`) [Function]

Convert *a* to an interval/hypercube.

`ap_lincons1_array_t ap_abstract1_to_lincons_array` (*ap_manager_t** `man`, *ap_abstract1_t** `a`) [Function]

`ap_tcons1_array_t ap_abstract1_to_tcons_array` (*ap_manager_t** `man`, *ap_abstract1_t** `a`) [Function]

Convert *a* to a conjunction of linear (resp. tree) constraints.

The constraints are normally guaranteed to be without intervals.

`ap_generator1_array_t ap_abstract1_to_generator_array` (*ap_manager_t** `man`, *ap_abstract1_t** `a`) [Function]

Convert *a* to an array of generators.

7.8.9 Meet and Join of abstract values of level 1

`ap_abstract1_t ap_abstract1_meet` (*ap_manager_t** `man`, *bool* `destructive`, *ap_abstract1_t** `a1`, *ap_abstract1_t** `a2`) [Function]

`ap_abstract1_t ap_abstract1_join` (*ap_manager_t** `man`, *bool* `destructive`, *ap_abstract1_t** `a1`, *ap_abstract1_t** `a2`) [Function]

Meet and Join of 2 abstract values

`ap_abstract1_t ap_abstract1_meet_array` (*ap_manager_t** `man`, *ap_abstract1_t** `array`, *size_t* `size`) [Function]

`ap_abstract1_t ap_abstract1_join_array` (*ap_manager_t** `man`, *ap_abstract1_t** `array`, *size_t* `size`) [Function]

Meet and Join of the array *array* of abstract values of size *size*.

Raise an `AP_EXC_INVALID_ARGUMENT` exception if `size==1` (no way to define the environment of the result in such a case).

`ap_abstract1_t ap_abstract1_meet_lincons_array` (*ap_manager_t** `man`, *bool* `destructive`, *ap_abstract1_t** `a`, *ap_lincons1_array_t** `array`) [Function]

`ap_abstract1_t ap_abstract1_meet_tcons_array` (*ap_manager_t** `man`, *bool* `destructive`, *ap_abstract1_t** `a`, *ap_tcons1_array_t** `array`) [Function]

Meet of the abstract value a with the set of constraints *array*.

`ap_abstract1_t ap_abstract1_add_ray_array` (*ap_manager_t** `man`, *bool* `destructive`, *ap_abstract1_t** `a`, *ap_generator1_array_t** `array`) [Function]

Generalized time elapse operator.

array is supposed to contain only rays or lines, no vertices.

7.8.10 Assignements and Substitutions of abstract values of level 1

`ap_abstract1_t ap_abstract1_assign_linexpr_array` (*ap_manager_t** `man`, *bool* `destructive`, *ap_abstract1_t** `org`, *ap_var_t** `tvar`, *ap_linexpr1_t** `texpr`, *size_t* `size`, *ap_abstract1_t** `dest`) [Function]

`ap_abstract1_t ap_abstract1_substitute_linexpr_array` (*ap_manager_t** `man`, *bool* `destructive`, *ap_abstract1_t** `org`, *ap_var_t** `tvar`, *ap_linexpr1_t** `texpr`, *size_t* `size`, *ap_abstract1_t** `dest`) [Function]

`ap_abstract1_t ap_abstract1_assign_texpr_array` (*ap_manager_t** `man`, *bool* `destructive`, *ap_abstract1_t** `org`, *ap_var_t** `tvar`, *ap_texpr1_t** `texpr`, *size_t* `size`, *ap_abstract1_t** `dest`) [Function] `ap_abstract1_t ap_abstract1_substitute_texpr_array` (*ap_manager_t** **`man`**, *bool* **`destructive`**, *ap_abstract1_t** **`org`**, *ap_var_t** **`tvar`**, *ap_texpr1_t** **`texpr`**, *size_t* **`size`**, *ap_abstract1_t** **`dest`**) [Function]

Parallel Assignement and Substitution of several variables by expressions in abstract value *org*.

dest is an optional argument. If not NULL, semantically speaking, the result of the transformation is intersected with *dest*. This is useful for precise backward transformations in lattices like intervals or octagons.

7.8.11 Existential quantification of abstract values of level 1

`ap_abstract1_t ap_abstract1_forget_array` (*ap_manager_t** **`man`**, *bool* **`destructive`**, *ap_abstract1_t** **`a`**, *ap_var_t** **`tvar`**, *size_t* **`size`**, *bool* **`project`**) [Function]

Forget (`project=false`) or Project (`project=true`) the array of variables *tvar* of size *size* in the abstract value *a*.

7.8.12 Change of environments of abstract values of level 1

`ap_abstract1_t ap_abstract1_change_environment` (*ap_manager_t** **`man`**, *bool* **`destructive`**, *ap_abstract1_t** **`a`**, *ap_environment_t** **`nenv`**, *bool* **`project`**) [Function]

Change the environment of the abstract values. Variables that are removed are first existentially quantified, and variables that are introduced are either unconstrained (`project==false`) or initialized to 0 (`project==false`).

`ap_abstract1_t ap_abstract1_minimize_environment` (*ap_manager_t** **`man`**, *bool* **`destructive`**, *ap_abstract1_t** **`a`**) [Function]

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

`ap_abstract1_t ap_abstract1_rename_array` (*ap_manager_t** **`man`**, *bool* **`destructive`**, *ap_abstract1_t** **`a`**, *ap_var_t** **`tvar`**, *ap_var_t** **`ntvar`**, *size_t* **`size`**) [Function]

Parallel renaming of the environment of the abstract value. The new variables should not interfere with the variables that are not renamed.

7.8.13 Expansion and Folding of dimensions of abstract values of level 1

Formally, expanding `z` into `z` and `w` in abstract value (predicate) `P` is defined by $expand(P(x,y,z),z,w) = P(x,y,z) and P(x,y,w)$.

Conversely, folding `z` and `w` into `z` in abstract value (predicate) `Q` is defined by $fold(Q(x,y,z,w),z,w) = (exists w : Q(x,y,z,w)) or (exists z : Q(x,y,z,w)[z < -w])$.

`ap_abstract1_t ap_abstract1_expand` (*ap_manager_t** **`man`**, *bool* **`destructive`**, *ap_abstract1_t** **`a`**, *ap_var_t* **`var`**, *ap_var_t** **`tvar`**, *size_t* **`size`**) [Function]

Expand the variable *var* into itself + the *size* additional variables of the array *tvar*, which are given the same type as *var*. 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 environment.

It results in `size+1` unrelated variables having same relations with other dimensions.

`ap_abstract1_t ap_abstract1_fold` (*ap_manager_t* man, bool destructive, ap_abstract1_t* a, ap_var_t* tvar, size_t size*) [Function]

Fold the variables in the array *tvar* of size *size*>=1 and put the result in the first variable in the array. The other variables of the array are then forgot and removed from the environment.

7.8.14 Widening of abstract values of level 1

`ap_abstract1_t ap_abstract1_widening` (*ap_manager_t* man, ap_abstract1_t* a1, ap_abstract1_t* a2*) [Function]

Widening of *a1* with *a2*. *a1* is supposed to be included in *a2*.

`ap_abstract1_t ap_abstract1_widening_threshold` (*ap_manager_t* man, ap_abstract1_t* a1, ap_abstract1_t* a2, ap_lincons1_array_t* array*) [Function]

Widening with threshold.

Intersect the result of the standard widening with all the constraints in *array* that are satisfied by both *a1* and *a2*.

7.8.15 Topological closure of abstract values of level 1

`ap_abstract1_t* ap_abstract1_closure` (*ap_manager_t* man, bool destructive, ap_abstract1_t* a*) [Function]

Relax strict constraints into non strict constraints.

7.8.16 Additional functions on abstract values of level 1

`ap_abstract1_t ap_abstract1_of_lincons_array` (*ap_manager_t* man, ap_environment_t* env, ap_lincons1_array_t* array*) [Function]

`ap_abstract1_t ap_abstract1_of_tcons_array` (*ap_manager_t* man, ap_environment_t* env, ap_tcons1_array_t* array*) [Function]

Abstract a conjunction of constraints. The environment of the array should be a subset of the environment *env*.

`ap_abstract1_t ap_abstract1_assign_linexpr` (*ap_manager_t* man, bool destructive, ap_abstract1_t* org, ap_var_t var, ap_linexpr1_t* expr, ap_abstract1_t* dest*) [Function]

`ap_abstract1_t ap_abstract1_substitute_linexpr` (*ap_manager_t* man, bool destructive, ap_abstract1_t* org, ap_var_t var, ap_linexpr1_t* expr, ap_abstract1_t* dest*) [Function]

`ap_abstract1_t ap_abstract1_assign_texpr` (*ap_manager_t* man, bool destructive, ap_abstract1_t* org, ap_var_t var, ap_texpr1_t* expr, ap_abstract1_t* dest*) [Function]

`ap_abstract1_t ap_abstract1_substitute_texpr` (*ap_manager_t* man, bool destructive, ap_abstract1_t* org, ap_var_t var, ap_texpr1_t* expr, ap_abstract1_t* dest*) [Function]

Assignement and Substitution of the dimension *dim* by the expression *expr* in abstract value *org*.

dest is an optional argument. If not NULL, semantically speaking, the result of the transformation is intersected with *dest*. This is useful for precise backward transformations in lattices like intervals or octagons.

`ap_abstract1_t ap_abstract1_unify` (*ap_manager_t* **man***, *bool **destructive***, *ap_abstract1_t** **a1**, *ap_abstract1_t** **a2**) [Function]

Unify two abstract values on their common variables, that is, embed them on the least common environment and then compute their meet. The result is defined on the least common environment.

For instance, the unification of `1<=x<=3 and x=y` defined on `{ x, y }` and `2<=z<=4 and z=y` defined on `{y,z }` results in `2<=x<=3 and x=y=z` defined on `{x,y,z}`.

`ap_linexpr1_t ap_abstract1_quasilinear_of_intlinear` (*ap_manager_t* **man***, *ap_abstract1_t** **a**, *ap_linexpr1_t* **expr***) [Function]

Evaluate the interval linear expression *expr* on the abstract value *a* and approximate it by a quasilinear expression.

This implies calls to `ap_abstract0_bound_dimension`.

`ap_linexpr1_t ap_abstract1_intlinear_of_tree` (*ap_manager_t* **man***, *ap_abstract1_t** **a**, *ap_texpr1_t* **expr***, *bool quasilinear*) [Function]

Evaluate the tree expression *expr* on the abstract value *a* and approximate it by an interval linear (resp. quasilinear if *quasilinear* is true) expression.

This implies calls to `ap_abstract0_bound_dimension`.

8 Level 0 of the interface

This interface of level 0 is defined in 'ap_global0.h'.

Unless there exists specific reasons for not doing so, we advise the user to use the level 1 of the interface (see Chapter 7 [Level 1 of the interface], page 45). The level 0 is intented for implementors who wants to connect a new library/abstract domain, or who want to build a composite domain from existing ones.

For information only (as most of these types are considered as abstract) and for implementors, we sum up the involved types below.

```
    ap_dim_t          ap_dimension_t
|--------------|   |----------------|
| unsigned int |   | size_t intdim  |
|--------------|   | size_t realdim |
                   |----------------|

  ap_dimchange_t      ap_dimperm_t
|----------------|   |-----------|
| ap_dim_t* dim  |   | ap_dim_t* |
| size_t intdim  |   | size_t    |
| size_t realdim |   |-----------|
|----------------|

        ap_linexpr0_t              ap_linterm_t
|-----------------------------|   |------------|
|      ap_coeff_t       cst   |   | ap_dim_t   |
|      ap_linexpr_discr discr |   | ap_coeff_t |
|      size_t           size  |   |------------|
|-----------------------------|
| ap_coeff_t* | ap_linterm_t* |
|-----------------------------|

   ap_lincons0_t        ap_generator0_t
|------------------|   |----------------|
| ap_linexpr0_t*   |   | ap_linexpr0_t* |
| ap_constyp_t     |   | ap_gentyp_t    |
| ap_scalar_t* mod |   |----------------|
|------------------|

 ap_abstract0_t
|---------------|
| void*         |
| ap_manager_t* |
|---------------|
```

8.1 Dimensions and related operations ('ap_dimension.h')

ap_dim_t [datatype]

```
typedef unsigned int ap_dim_t;
```

Datatype for dimensions.

`AP_DIM_MAX` [Macro]

Special value used for sparse representations, means: "to be ignored". Also used as a result when an error occurs.

`ap_dimension_t` [datatype]

```
typedef struct ap_dimension_t {
  size_t intdim;  /* Number of integer dimensions */
  size_t realdim; /* Number of real dimensions */
} ap_dimension_t;
```

Datatype for specifying the dimensionality of an abstract value.

`ap_dimchange_t` [datatype]

```
typedef struct ap_dimchange_t {
  ap_dim_t* dim;  /* Assumed to be an array of size intdim+realdim */
  size_t intdim ; /* Number of integer dimensions to add/remove */
  size_t realdim; /* Number of real dimensions to add/remove */
} ap_dimchange_t;
```

Datatype for specifying change of dimension.

The semantics is the following:

Addition of dimensions

`dimchange.dim[k]` means: add one dimension at dimension k and shift the already existing dimensions greater than or equal to `k` one step on the right (or increment them).

if `k` is equal to the size of the vector, then it means: add a dimension at the end.

Repetion are allowed, and means that one inserts more than one dimensions.

Example: `linexpr0_add_dimensions([i0 i1 r0 r1], { [0 1 2 2 4],3,1 })` returns `[0 i0 0 i1 0 0 r0 r1 0]`, considered as a vector with 5 integer dimensions and 4 real dimensions.

Removal of dimensions

`dimchange.dim[k]`: remove the dimension `k` and shift the dimensions greater than `k` one step on the left (or decrement them).

Repetitions are meaningless (and are not correct specification).

Example: `linexpr0_remove_dimensions([i0 i1 i2 r0 r1 r2], { [0 2 4],2,1 })` returns `[i1 r0 r2]`, considered as a vector with 1 integer dimensions and 2 real dimensions.

`ap_dimchange2_t` [datatype]

```
typedef struct ap_dimchange_2t {
  ap_dimchange_t* add;    /* If not NULL, specifies the adding new dimensions */
  ap_dimchange_t* remove; /* If not NULL, specifies the removal of dimensions */
} ap_dimchange2_t;
```

Datatype for specifying a transformation composed of the addition and the removal of dimensions. Used by functions `ap_abstract0_apply_dimchange2`, `ap_environment_dimchange2`, and `ap_abstract1_change_environment`..

`ap_dimperm_t` [datatype]

```
typedef struct ap_dimperm_t {
```

```
    ap_dim_t* dim; /* Array assumed to be of size size */
    size_t size;
  } ap_dimperm_t;
```

Datatype for permutations.

Represents the permutation `i -> dimperm.p[i]` for `0<=i<dimperm.size`.

8.1.1 Manipulating changes of dimensions

`void ap_dimchange_init` (*ap_dimchange_t** ***dimchange***, *size_t* ***intdim***, *size_t* ***realdim***) [Function]

`void ap_dimchange_clear` (*ap_dimchange_t** ***dimchange***) [Function]

Initialize and clear a dimchange structure.

`ap_dimchange_t* ap_dimchange_alloc` (*size_t* ***intdim***, *size_t* ***realdim***) [Function]

`void ap_dimchange_free` (*ap_dimchange_t** ***dimchange***) [Function]

Allocate and free a dimchange structure.

`void ap_dimchange_fprint` (*FILE** ***stream***, *ap_dimchange_t** ***dimchange***) [Function]

Print the change of dimension.

`void ap_dimchange_add_invert` (*ap_dimchange_t** ***dimchange***) [Function]

Assuming that dimchange is a transformation for the addition of dimensions, invert it to obtain the inverse transformation for removing dimensions.

`void ap_dimchange2_init` (*ap_dimchange2_t** ***dimchange2***, *ap_dimchange_t** ***add***, *ap_dimchange_t** ***remove***) [Function]

`void ap_dimchange2_clear` (*ap_dimchange2_t** ***dimchange2***) [Function]

Initialize (with *add* and *remove*) and clear a dimchange2 structure.

`ap_dimchange2_t* ap_dimchange2_alloc` (*ap_dimchange_t** ***add***, *ap_dimchange_t** ***remove***) [Function]

`void ap_dimchange2_free` (*ap_dimchange2_t** ***dimchange2***) [Function]

Allocate and free a dimchange2 structure.

`void ap_dimchange2_fprint` (*FILE** ***stream***, *ap_dimchange2_t** ***dimchange2***) [Function]

Print the change of dimension.

8.1.2 Manipulating permutations of dimensions

`void ap_dimperm_init` (*ap_dimperm_t** ***perm***, *size_t* ***size***) [Function]

`void ap_dimperm_clear` (*ap_dimperm_t** ***perm***) [Function]

Initialize and clear a dimperm structure.

`ap_dimperm_t* ap_dimperm_alloc` (*size_t* ***size***) [Function]

`void ap_dimperm_free` (*ap_dimperm_t** ***perm***) [Function]

Allocate and free a dimperm structure.

`void ap_dimperm_fprint` (*FILE** ***stream***, *ap_dimperm_t** ***perm***) [Function]

Print the permutation.

`void ap_dimperm_set_id` (*ap_dimperm_t** ***perm***) [Function]

Fill the already allocated *perm* with the identity permutation.

`void ap_dimperm_compose` (*ap_dimperm_t* **perm**, ap_dimperm_t* **perm1**, ap_dimperm_t* **perm2***) [Function]

Compose the 2 permutations *perm1* and *perm2* (in this order) and store the result the already allocated perm. The sizes of permutations are supposed to be equal. At exit, we have `perm.dim[i] = perm2.dim[perm1.dim[i]]`.

`void ap_dimperm_invert` (*ap_dimperm_t* **nperm**, ap_dimperm_t* **perm***) [Function]

Invert the permutation *perm* and store it in the already allocated *nperm*. The sizes of permutations are supposed to be equal.

8.2 Linear expressions of level 0 ('`ap_linexpr0.h`')

`ap_linexpr_discr_t` [datatype]

```
typedef enum ap_linexpr_discr_t {
  LINEXPR_DENSE,
  LINEXPR_SPARSE
} ap_linexpr_discr_t;
```

Type of representation of linear expressions: either dense or sparse.

`ap_linexpr0_t` [datatype]

Type of interval linear expressions. Coefficients in such expressions are of type `coeff_t`.

8.2.1 Allocating linear expressions of level 0

`ap_linexpr0_t* ap_linexpr0_alloc` (*ap_linexpr_discr_t **lin_discr**, size_t **size**);* [Function]

Allocate a linear expressions with coefficients by default of type SCALAR and DOUBLE. If sparse representation, corresponding new dimensions are initialized with `AP_DIM_MAX`.

`void ap_linexpr0_realloc` (*ap_linexpr0_t* **e**, size_t **size***) [Function]

Change the dimensions of the array in *e*. If new coefficients are added, their type is of type SCALAR and DOUBLE. If sparse representation, corresponding new dimensions are initialized with `AP_DIM_MAX`.

`void ap_linexpr0_minimize` (*ap_linexpr0_t* **e***) [Function]

Reduce the coefficients (transform intervals into scalars when possible). In case of sparse representation, also remove zero coefficients.

`void ap_linexpr0_free` (*ap_linexpr0_t* **e**);* [Function]

Deallocate the linear expression.

`ap_linexpr0_t* ap_linexpr0_copy` (*ap_linexpr0_t* **e***) [Function]

Duplication.

`void ap_linexpr0_fprint` (*FILE* **stream**, ap_linexpr0_t* **e**, char** **name_of_dim**);* [Function]

Print the linear expression on stream *stream*, using the array *name_of_dim* to convert dimensions to variable names. If *name_of_dim* is `NULL`, the dimensions are named `x0,x1,...`.

8.2.2 Tests on linear expressions of level 0

`bool ap_linexpr0_is_integer` (*ap_linexpr0_t* **e**, size_t **intdim***) [Function]

Does the expression depends only on integer variables ? assuming that the first intdim dimensions are integer.

`bool ap_linexpr0_is_real` (*ap_linexpr0_t** **e**, *size_t* **intdim**) [Function]

Does the expression depends only on real variables ? assuming that the first intdim dimensions are integer .

`bool ap_linexpr0_is_linear` (*ap_linexpr0_t** **e**) [Function]

Return true iff all involved coefficients are scalars.

`bool ap_linexpr0_is_quasilinear` (*ap_linexpr0_t** **e**) [Function]

Return true iff all involved coefficients but the constant are scalars.

8.2.3 Access to linear expressions of level 0

`size_t ap_linexpr0_size` (*ap_linexpr0_t** **e**) [Function]

Get the size of the linear expression

8.2.3.1 Getting references

`ap_coefft* ap_linexpr0_cstref` (*ap_linexpr0_t** **e**) [Function]

Get a reference to the constant. Do not free it.

`ap_coefft* ap_linexpr0_coeffref` (*ap_linexpr0_t** **e**, *ap_dim_t* **dim**) [Function]

Get a reference to the coefficient associated to the dimension *dim* in expression *e*.

Do not free it. In case of sparse representation, possibly induce the addition of a new linear term.

Return NULL if:

- In case of dense representation, `dim>=e->size`.
- In case of sparse representation, `dim==AP_DIM_MAX`.

8.2.3.2 Getting values

`void ap_linexpr0_get_cst` (*ap_coefft** **coeff**, *ap_linexpr0_t** **e**) [Function]

Assign to *coeff* the constant coefficient of *e*.

`bool ap_linexpr0_get_coeff` (*ap_coefft* coeff*, *ap_linexpr0_t** **e**, *ap_dim_t* **dim**) [Function]

Assign to *coeff* the coefficient of dimension *dim* in the expression *e*.

Return true in case `ap_linexpr0_coeffref(e,dim)` returns NULL.

`ap_linexpr0_ForeachLinterm` (*ap_linexpr0_t** **e**, *size_t* **i**, *ap_dim_t* **dim**, *ap_coeff_t** **coeff**) [Macro]

Iterator on the coefficients associated to dimensions.

`ap_linexpr0_ForeachLinterm(E,I,DIM,COEFF){ body }` executes the body for each pair (*coeff*,*dim*) in the expression *e*. *coeff* is a reference to the coefficient associated to dimension *dim* in *e*. *i* is an auxiliary variable used internally by the macro.

8.2.3.3 Assigning values with a list of arguments

`ap_coefftag_t` [datatype]

```
typedef enum ap_coefftag_t {
  AP_COEFF,          /* waiting for a coeff_t* object and a dimension */
  AP_COEFF_S,        /* waiting for a scalar_t* object and a dimension */
  AP_COEFF_S_MPQ,    /* waiting for a mpq_t object and a dimension */
```

```
    AP_COEFF_S_INT,    /* waiting for a int object and a dimension */
    AP_COEFF_S_FRAC,   /* waiting for 2 int objects and a dimension */
    AP_COEFF_S_DOUBLE, /* waiting for a double object and a dimension */
    AP_COEFF_I,        /* waiting for a interval_t* object and a dimension */
    AP_COEFF_I_SCALAR, /* waiting for 2 scalar_t* objects and a dimension */
    AP_COEFF_I_MPQ,    /* waiting for 2 mpq_t objects and a dimension */
    AP_COEFF_I_INT,    /* waiting for 2 int objects and a dimension */
    AP_COEFF_I_FRAC,   /* waiting for 4 int objects and a dimension */
    AP_COEFF_I_DOUBLE, /* waiting for 2 double objects and a dimension */
    AP_CST,            /* waiting for a coeff_t* object */
    AP_CST_S,          /* waiting for a scalar_t* object */
    AP_CST_S_MPQ,      /* waiting for a mpq_t object */
    AP_CST_S_INT,      /* waiting for a int object */
    AP_CST_S_FRAC,     /* waiting for 2 int objects */
    AP_CST_S_DOUBLE,   /* waiting for a double object */
    AP_CST_I,          /* waiting for a interval_t* object */
    AP_CST_I_SCALAR,   /* waiting for 2 scalar_t* objects */
    AP_CST_I_MPQ,      /* waiting for 2 mpq_t objects */
    AP_CST_I_INT,      /* waiting for 2 int objects */
    AP_CST_I_FRAC,     /* waiting for 4 int objects */
    AP_CST_I_DOUBLE,   /* waiting for 2 double objects */
    AP_END             /* indicating end of the list */
  } ap_coefftag_t;
```

Tags for `ap_linexpr0_set_list` function.

`bool ap_linexpr0_set_list` (*ap_linexpr0_t** `e`, *...*) [Function]

This function assign the linear expression E from a list of tags of type `ap_coefftag_t`, each followed by a number of arguments as specified in the definition of the tye `ap_coefftag_t`. The list should end with the tag `AP_COEFF_END`.

Return `true` in case `ap_linexpr0_coeffref(e,dim)` returns NULL for one of the dimensions involved.

Here is a typical example:

```
ap_linexpr0_set_list(e,
     AP_COEFF_S_INT, 3, 0,
     AP_COEFF_S_FRAC, 3,2, 1,
     AP_COEFF_S_DOUBLE, 4.1, 2,
     AP_CST_I_DOUBLE, -2.4, 3.6,
     AP_END); /* Do not forget the last tatg ! */
```

which transforms an null expression into `3 x0 + 3/2 x1 + 4.1 x2 + [-2.4,3.6]` and is equivalent to:

```
ap_linexpr0_set_coeff_scalar_int(e,0, 3);
ap_linexpr0_set_coeff_scalar_frac(e,1, 3,2);
ap_linexpr0_set_coeff_scalar_double(e,2, 4.1);
ap_linexpr0_set_cst_interval_double(e, -2.4, 3.6);
```

8.2.3.4 Assigning values

`void ap_linexpr0_set_cst` (*ap_linexpr0_t** `e`, *ap_coefft** `coeff`) [Function]

void ap_linexpr0_set_cst_scalar (*ap_linexpr0_t* e, ap_scalar_t* scalar*) [Function]
void ap_linexpr0_set_cst_scalar_int (*ap_linexpr0_t* e, int num*) [Function]
void ap_linexpr0_set_cst_scalar_frac (*ap_linexpr0_t* e, int num, unsigned int den*) [Function]
void ap_linexpr0_set_cst_scalar_double (*ap_linexpr0_t* e, double num*) [Function]
void ap_linexpr0_set_cst_interval (*ap_linexpr0_t* e, ap_interval_t* itv*) [Function]
void ap_linexpr0_set_cst_interval_scalar (*ap_linexpr0_t* e, ap_scalar_t* inf, ap_scalar_t* sup*) [Function]
void ap_linexpr0_set_cst_interval_int (*ap_linexpr0_t* e, int inf, int sup*) [Function]
void ap_linexpr0_set_cst_interval_frac (*ap_linexpr0_t* e, int numinf, unsigned int deninf, int numsup, unsigned int densup*) [Function]
void ap_linexpr0_set_cst_interval_double (*ap_linexpr0_t* e, double inf, double sup*) [Function]

Set the constant coefficient of expression *e*.

bool ap_linexpr0_set_coeff (*ap_linexpr0_t* e, ap_dim_t dim, ap_coefft* coeff*) [Function]
bool ap_linexpr0_set_coeff_scalar (*ap_linexpr0_t* e, ap_dim_t dim, ap_scalar_t* scalar*) [Function]
bool ap_linexpr0_set_coeff_scalar_int (*ap_linexpr0_t* e, ap_dim_t dim, int num*) [Function]
bool ap_linexpr0_set_coeff_scalar_frac (*ap_linexpr0_t* e, ap_dim_t dim, int num, unsigned int den*) [Function]
bool ap_linexpr0_set_coeff_scalar_double (*ap_linexpr0_t* e, ap_dim_t dim, double num*) [Function]
bool ap_linexpr0_set_coeff_interval (*ap_linexpr0_t* e, ap_dim_t dim, ap_interval_t* itv*) [Function]
bool ap_linexpr0_set_coeff_interval_scalar (*ap_linexpr0_t* e, ap_dim_t dim, ap_scalar_t* inf, ap_scalar_t* sup*) [Function]
bool ap_linexpr0_set_coeff_interval_int (*ap_linexpr0_t* e, ap_dim_t dim, int inf, int sup*) [Function]
bool ap_linexpr0_set_coeff_interval_frac (*ap_linexpr0_t* e, ap_dim_t dim, int numinf, unsigned int deninf, int numsup, unsigned int densup*) [Function]
void ap_linexpr0_set_coeff_interval_double (*ap_linexpr0_t* e, ap_dim_t dim, double inf, double sup*) [Function]

Set the coefficient of the dimension *dim* of expression *e*.

Return `true` in case `ap_linexpr0_coeffref(e,dim)` returns NULL.

8.2.4 Change of dimensions and permutations of linear expressions of level 0

void ap_linexpr0_add_dimensions_with (*ap_linexpr0_t* e, ap_dimchange_t* dimchange*) [Function] `ap_linexpr0_t* ap_linexpr0_add_dimensions` (*ap_linexpr0_t* e, ap_dimchange_t* dimchange*) [Function]

These two functions add dimensions to the expressions, following the semantics of dimchange (see the type definition of `ap_dimchange_t`).

`void ap_linexpr0_permute_dimensions_with` (*ap_linexpr0_t* e, ap_dimperm_t* perm*) [Function]

`ap_linexpr0_t* ap_linexpr0_permute_dimensions` (*ap_linexpr0_t* e, ap_dimperm_t* perm*) [Function]

These two functions apply the given permutation to the dimensions of *e*. If dense representation, the size of the permutation should be `e->size`. If sparse representation, the dimensions present in the expression should just be less than the size of the permutation.

8.2.5 Other functions on linear expressions of level 0

All these functions induces a reduction of the coefficients of the linear expression.

`int ap_linexpr0_hash` (*ap_linexpr0_t* e*) [Function]

Return a hash code.

`bool ap_linexpr0_equal` (*ap_linexpr0_t* e1, ap_linexpr0_t* e2*) [Function]

Equality test.

`int ap_linexpr0_compare` (*ap_linexpr0_t* e1, ap_linexpr0_t* e2*) [Function]

Lexicographic ordering, terminating by constant coefficients.

Use the (partial order) comparison function on coefficients `coeff_cmp`.

8.3 Linear constraints of level 0 (‘`ap_lincons0.h`’)

`ap_constyp_t` [datatype]

```
typedef enum ap_constyp_t {
  AP_CONS_EQ,    /* equality constraint */
  AP_CONS_SUPEQ, /* >= constraint */
  AP_CONS_SUP,   /* > constraint */
  AP_CONS_EQMOD, /* congruence equality constraint */
  AP_CONS_DISEQ  /* disequality constraint */
} ap_constyp_t;
```

Datatype for type of constraints.

`ap_lincons0_t` [datatype]

```
typedef struct ap_lincons0_t {
  ap_linexpr0_t* linexpr0;   /* expression */
  ap_constyp_t constyp;      /* type of constraint */
  ap_scalar_t* scalar;       /* maybe NULL.

For EQMOD constraint, indicates the
modulo */
} ap_lincons0_t;
```

Datatype for constraints.

Constraints are meant to be manipulated freely via their components. Creating the constraint `[1,2]x0 + 5/2x1 >=0` and then freeing it can be done with

```
ap_lincons0_t cons = ap_lincons0_make(AP_CONS_SUPEQ,
      ap_linexpr0_alloc(AP_LINEXPR_SPARSE,2),
                                  NULL);
ap_linexpr0_set_list(cons.linexpr0,
     AP_COEFF_I_INT, 1,2, 0,
     AP_COEFF_S_FRAC, 5,2, 1,
     AP_END);
ap_lincons0_clear(&cons);
```

`ap_lincons0_array_t` [datatype]

```
typedef struct ap_lincons0_array_t {
  ap_lincons0_t* p;
  size_t size;
} ap_lincons0_array_t;
```

Datatype for arrays of constraints.

Arrays are accessed directly, for example by writing `array->p[i]` (of type `ap_lincons0_t`), `array->p[i].constyp` and `array->p[i].linexpr0`.

One can assign a constraint to the index *index* by writing: `array->p[index] = ap_lincons0_make(constyp,expr)`.

8.3.1 Allocating linear constraints of level 0

`ap_lincons0_t ap_lincons0_make` (*ap_constyp_t* ***constyp***, *ap_linexpr0_t** ***linexpr***, *ap_scalar_t** ***mod***) [Function]

Create a constraint of type *constyp* with the expression *linexpr*, and the modulo *mod* in case of a congruence constraint (`constyp==AP_CONS_EQMOD`).

The expression is not duplicated, just pointed to, so it becomes managed via the constraint.

`ap_lincons0_t ap_lincons0_make_unsat` () [Function]

Create the constraint `-1>=0`.

`ap_lincons0_t ap_lincons0_copy` (*ap_lincons0_t** ***cons***) [Function]

Duplication

`void ap_lincons0_clear` (*ap_lincons0_t** ***cons***) [Function]

Clear the constraint.

`void ap_lincons0_fprint` (*FILE** ***stream***, *ap_lincons0_t** ***cons***, *char*** ***name_of_dim***); [Function]

Print the linear constraint on stream *stream*, using the array *name_of_dim* to convert dimensions to variable names. If *name_of_dim* is `NULL`, the dimensions are named `x0,x1,...`.

8.3.2 Tests on linear constraints of level 0

`bool ap_lincons0_is_unsat` (*ap_lincons0_t** ***cons***) [Function]

Return `true` if the constraint is not satisfiable.

8.3.3 Arrays of linear constraints of level 0

`ap_lincons0_array_t ap_lincons0_array_make` (*size_t* ***size***) [Function]

Allocate an array of size constraints.

The constraints are initialized with NULL pointers for underlying expressions.

`void ap_lincons0_array_clear` (*ap_lincons0_array_t* array*) [Function]

Clear the constraints of the array, and then the array itself.

`void ap_lincons0_array_fprint` (*FILE* stream, ap_lincons0_array_t* array, char** name_of_dim*) [Function]

Print the array on the stream.

8.3.4 Change of dimensions and permutations of linear constraints of level 0

`void ap_lincons0_add_dimensions_with` (*ap_lincons0_t* cons, ap_dimchange_t* dimchange*) [Function]

`ap_lincons0_t ap_lincons0_add_dimensions` (*ap_lincons0_t* cons, ap_dimchange_t* dimchange*) [Function]

These two functions add dimensions to the constraint, following the semantics of dimchange (see the type definition of `ap_dimchange_t`).

`void ap_lincons0_permute_dimensions_with` (*ap_lincons0_t* cons, ap_dimperm_t* perm*) [Function]

`ap_lincons0_t ap_lincons0_permute_dimensions` (*ap_lincons0_t* cons, ap_dimperm_t* perm*) [Function]

These two functions apply the given permutation to the dimensions of *cons*.

`void ap_lincons0_array_add_dimensions_with` (*ap_lincons0_array_t* cons, ap_dimchange_t* dimchange*) [Function]

`ap_lincons0_array_t ap_lincons0_array_add_dimensions` (*ap_lincons0_array_t* cons, ap_dimchange_t* dimchange*) [Function]

`void ap_lincons0_array_permute_dimensions_with` (*ap_lincons0_array_t* cons, ap_dimperm_t* perm*) [Function]

`ap_lincons0_array_t ap_lincons0_array_permute_dimensions` (*ap_lincons0_array_t* cons, ap_dimperm_t* perm*) [Function]

Extension to arrays of the corresponding functions on constraints.

8.4 Generators of level 0 ('`ap_generator0.h`')

Datatypes and functions are almost isomorphic to datatypes and functions for linear constraints.

`ap_gentyp_t` [datatype]

```
typedef enum ap_gentyp_t {
  AP_GEN_LINE,
  AP_GEN_RAY,
  AP_GEN_VERTEX,
  AP_GEN_LINEMOD,
  AP_GEN_RAYMOD
} ap_gentyp_t;
```

Datatype for type of generators.

`ap_generator0_t` [datatype]

```
typedef struct ap_generator0_t {
  ap_linexpr0_t* linexpr0; /* underlying expression.  */
  ap_gentyp_t gentyp;      /* type of generator */
} ap_generator0_t;
```

Datatype for generators.

The constant of the expression is ignored, and the expression is assumed to be truly linear (without intervals).

`ap_generator0_array_t` [datatype]

```
typedef struct ap_generator0_array_t {
  ap_generator0_t* p;
  size_t size;
} ap_generator0_array_t;
```

Datatype for arrays of generators.

8.4.1 Allocating generators of level 0

`ap_generator0_t ap_generator0_make` (*ap_gentyp_t* **`gentyp`**, *ap_linexpr0_t** **`linexpr`**) [Function]

Create a generator of type *gentyp* with the expression *linexpr*.

The expression is not duplicated, just pointed to, so it becomes managed via the generator.

`ap_generator0_t ap_generator0_copy` (*gent ap_generator0_t** **`gen`**) [Function]

Duplication

`void ap_generator0_clear` (*ap_generator0_t** **`gen`**) [Function]

Clear the generator.

`void ap_generator0_fprint` (*FILE** **`stream`**, *gent ap_generator0_t** **`gen`**, *char*** **`name_of_dim`***);* [Function]

Print the linear generator on stream *stream*, using the array *name_of_dim* to convert dimensions to variable names. If *name_of_dim* is `NULL`, the dimensions are named `x0,x1,...`.

8.4.2 Arrays of generators of level 0

Arrays are accessed directly, for example by writing `array->p[i]` (of type `ap_generator0_t`), `array->p[i].gentyp` and `array->p[i].linexpr0`.

One can assign a generator to the index *index* by writing: `array->p[index] = ap_generator0_make(gentyp,expr)`.

`ap_generator0_array_t ap_generator0_array_make` (*size_t* **`size`**) [Function]

Allocate an array of size generators. The generators are initialized with NULL pointers for underlying expressions.

`void ap_generator0_array_clear` (*ap_generator0_array_t** **`array`**) [Function]

Clear the generators of the array, and then the array itself.

`void ap_generator0_array_fprint` (*FILE** **`stream`**, *gent ap_generator0_array_t** **`array`**, *char*** **`name_of_dim`**) [Function]

Print the array on the stream.

8.4.3 Change of dimensions and permutations of generators of level 0

`void ap_generator0_add_dimensions_with` (*ap_generator0_t** **`gen`**, *gent ap_dimchange_t** **`dimchange`**) [Function] `ap_generator0_t ap_generator0_add_dimensions` (*gent ap_generator0_t** **gen**, *gent ap_dimchange_t** **dimchange**) [Function]

These two functions add dimensions to the generator, following the semantics of dimchange (see the type definition of `ap_dimchange_t`).

`void ap_generator0_permute_dimensions_with` (*ap_generator0_t** **gen**, *gent ap_dimperm_t** **perm**) [Function]

`ap_generator0_t ap_generator0_permute_dimensions` (*gent ap_generator0_t** **gen**, *gent ap_dimperm_t** **perm**) [Function]

These two functions apply the given permutation to the dimensions of *gen*.

`void ap_generator0_array_add_dimensions_with` (*ap_generator0_array_t** **gen**, *gent ap_dimchange_t** **dimchange**) [Function]

`ap_generator0_array_t ap_generator0_array_add_dimensions` (*gent ap_generator0_array_t** **gen**, *gent ap_dimchange_t** **dimchange**) [Function]

`void ap_generator0_array_permute_dimensions_with` (*ap_generator0_array_t** **gen**, *gent ap_dimperm_t** **perm**) [Function]

`ap_generator0_array_t ap_generator0_array_permute_dimensions` (*gent ap_generator0_array_t** **gen**, *gent ap_dimperm_t** **perm**) [Function]

Extension to arrays of the corresponding functions on generators.

8.5 Tree expressions of level 0 (‘`ap_texpr0.h`’)

8.6 Tree constraints of level 0 (‘`ap_tcons0.h`’)

8.7 Abstract values and operations of level 0 (‘`ap_abstract0.h`’)

`ap_abstract0_t` [datatype]

Datatype for abstract values at level 0.

Most operations are offered in 2 versions: *functional* or *destructive*. In such a case, the Boolean argument *destructive* controls the behaviour of the functionn:

- In the *destructive semantics*, after the call the first abstract value in the arguments of the function is destroyed and should not be referenced any more. Although the returned value might actually be equal to the (destroyed) argument, the user just manipulates the returned value and never refers directly to the (destroyed) argument.
- In the *functional semantics*, the first abstract value in the arguments is neither (semantically) modified nor deallocated.

8.7.1 Allocating abstract values of level 0

`ap_abstract0_t* ap_abstract0_copy` (*ap_manager_t** **man**, *ap_abstract0_t** **a**) [Function]

Return a copy of *a*, on which destructive update does not affect *a*.

`void ap_abstract0_free` (*ap_manager_t** **man**, *ap_abstract0_t** **a**) [Function]

Free all the memory used by *a*.

`size_t ap_abstract0_size` (*ap_manager_t** **man**, *ap_abstract0_t** **a**) [Function]

Return the abstract size of *a*.

8.7.2 Control of internal representation of level 0

`void ap_abstract0_minimize` (*ap_manager_t* man, ap_abstract0_t* a*) [Function]

Minimize the size of the representation of *a*. This may result in a later recomputation of internal information.

`void ap_abstract0_canonicalize` (*ap_manager_t* man, ap_abstract0_t* a*) [Function]

Put *a* in canonical form. (not yet clear definition)

`int ap_abstract0_hash` (*ap_manager_t* man, ap_abstract0_t* a*) [Function]

Return an hash value for *a*. Two abstract values in canonical from (according to `ap_abstract0_canonicalize`) and considered as equal by the function `ap_abstract0_is_eq` should be given the same hash value (this implies more or less a canonical form).

`void ap_abstract0_approximate` (*ap_manager_t* man, ap_abstract0_t* a, int algorithm*) [Function]

Perform some transformation on *a*, guided by the field algorithm.

The transformation may lose information. The argument *algorithm* overrides the field algorithm of the structure of type `ap_funopt_t` associated to `ap_abstract0_approximate`.

8.7.3 Printing abstract values of level 0

`void ap_abstract0_fprint` (*FILE* stream, ap_manager_t* man, ap_abstract0_t* a, char** name_of_dim*) [Function]

Print *a* in a pretty way, using array *name_of_dim* to name dimensions.. If *name_of_dim* is `NULL`, use the default names `x0, x1, ...`.

`void ap_abstract0_fprintdiff` (*FILE* stream, ap_manager_t* man, ap_abstract0_t* a1, ap_abstract0_t* a2, char** name_of_dim*) [Function]

Print the difference between *a1* (old value) and *a2* (new value), using array *name_of_dim* to name dimensions. The meaning of difference is library dependent.

`void ap_abstract0_fdump` (*FILE* stream, ap_manager_t* man, ap_abstract0_t* a*) [Function]

Dump the internal representation of *a* for debugging purposes.

8.7.4 Serialization of abstract values of level 0

`ap_membuf_t ap_abstract0_serialize_raw` (*ap_manager_t* man, ap_abstract0_t* a*) [Function]

Allocate a memory buffer (with `malloc`), output *a* in raw binary format to it and return a pointer on the memory buffer and the number of bytes written. It is the user responsability to free the memory afterwards (with free).

`ap_abstract0_t* ap_abstract0_deserialize_raw` (*ap_manager_t* man, void* ptr, size_t* size*) [Function]

Return the abstract value read in raw binary format from the buffer pointed by *ptr* and store in size the number of bytes read.

8.7.5 Constructors for abstract values of level 0

`ap_abstract0_t* ap_abstract0_bottom` (*ap_manager_t* man, size_t intdim, size_t realdim*) [Function] `ap_abstract0_t* ap_abstract0_top` (*ap_manager_t** **man**, *size_t* **intdim**, *size_t* **realdim**) [Function]

Create resp. a bottom (empty) value and a top (universe) value with *intdim* integer dimensions and *realdim* real dimensions.

`ap_abstract0_t* ap_abstract0_of_box` (*ap_manager_t** **man**, *size_t* **intdim**, *size_t* **realdim**, *ap_interval_t*** **array**) [Function]

Abstract an hypercube defined by the array of intervals *array* of size *intdim+realdim*.

8.7.6 Accessors for abstract values of level 0

`ap_dimension_t ap_abstract0_dimension` (*ap_manager_t** **man**, *ap_abstract0_t** **a**) [Function]

Return the dimensionality of *a*.

8.7.7 Tests on abstract values of level 0

In abstract tests,

- true means that the predicate is certainly true;
- false means false *or* don't know (an exception has occurred, or the exact computation was considered too expensive to be performed, according to the options).

`bool ap_abstract0_is_bottom` (*ap_manager_t** **man**, *ap_abstract0_t** **a**) [Function]
`bool ap_abstract0_is_top` (*ap_manager_t** **man**, *ap_abstract0_t** **a**) [Function]

Emtpiness and universality tests.

`bool ap_abstract0_is_leq` (*ap_manager_t** **man**, *ap_abstract0_t** **a1**, *ap_abstract0_t** **a2**) [Function]
`bool ap_abstract0_is_eq` (*ap_manager_t** **man**, *ap_abstract0_t** **a1**, *ap_abstract0_t** **a2**) [Function]

Inclusion and equality tests.

`bool ap_abstract0_sat_interval` (*ap_manager_t** **man**, *ap_abstract0_t** **a**, *ap_dim_t* **dim**, *ap_interval_t** **interval**) [Function]

Is the dimension *dim* included in the interval *interval* in the abstract value *a* ?

`bool ap_abstract0_sat_lincons` (*ap_manager_t** **man**, *ap_abstract0_t** **a**, *ap_lincons0_t** **cons**) [Function]
`bool ap_abstract0_sat_tcons` (*ap_manager_t** **man**, *ap_abstract0_t** **a**, *ap_tcons0_t** **cons**) [Function]

Does the abstract value *a* satisfy the constraint *cons* ?

`bool ap_abstract0_is_dimension_unconstrained` (*ap_manager_t** **man**, *ap_abstract0_t** **a**, *ap_dim_t* **dim**) [Function]

Is the dimension *dim* unconstrained in the abstract value *a* ? If it is the case, we have `forget(man,a,dim) == a`.

8.7.8 Extraction of properties of abstract values of level 0

`ap_interval_t* ap_abstract0_bound_dimension` (*ap_manager_t** **man**, *ap_abstract0_t** **a**, *ap_dim_t* **dim**) [Function]

Return the interval taken by the dimension *dim* over the abstract valuea `ap_interval_t* ap_abstract0_bound_linexpr` (*ap_manager_t** `man`, *ap_abstract0_t** `a`, *ap_linexpr0_t** `expr`) [Function]

`ap_interval_t* ap_abstract0_bound_texpr` (*ap_manager_t** `man`, *ap_abstract0_t** `a`, *ap_texpr0_t** `expr`) [Function]

Return the interval taken by a linear expression *expr* over the abstract value *a*.

This function allows to solve a Linear Programming (LP) problem, but depending on the underlying domain the solution may be not optimal.

`ap_interval_t** ap_abstract0_to_box` (*ap_manager_t** `man`, *ap_abstract0_t** `a`) [Function]

Convert *a* to an interval/hypercube. The size of the resulting array is ap_abstract0_dimension(man,a).

`ap_lincons0_array_t ap_abstract0_to_lincons_array` (*ap_manager_t** `man`, *ap_abstract0_t** `a`) [Function]

`ap_tcons0_array_t ap_abstract0_to_tcons_array` (*ap_manager_t** `man`, *ap_abstract0_t** `a`) [Function]

Convert *a* to a conjunction of constraints.

The constraints are normally guaranteed to be scalar (without intervals)

`ap_generator0_array_t ap_abstract0_to_generator_array` (*ap_manager_t** `man`, *ap_abstract0_t** `a`) [Function]

Convert *a* to an array of generators.

8.7.9 Meet and Join of abstract values of level 0

`ap_abstract0_t* ap_abstract0_meet` (*ap_manager_t** `man`, *bool* `destructive`, *ap_abstract0_t** `a1`, *ap_abstract0_t** `a2`) [Function]

`ap_abstract0_t* ap_abstract0_join` (*ap_manager_t** `man`, *bool* `destructive`, *ap_abstract0_t** `a1`, *ap_abstract0_t** `a2`) [Function]

Meet and Join of 2 abstract values

`ap_abstract0_t* ap_abstract0_meet_array` (*ap_manager_t** `man`, *ap_abstract0_t*** `array`, *size_t* `size`) [Function]

`ap_abstract0_t* ap_abstract0_join_array` (*ap_manager_t** `man`, *ap_abstract0_t*** `array`, *size_t* `size`) [Function]

Meet and Join of the array *array* of abstract values of size *size*.

Raise an `AP_EXC_INVALID_ARGUMENT` exception if `size==0` (no way to define the dimensionality of the result in such a case).

`ap_abstract0_t* ap_abstract0_meet_lincons_array` (*ap_manager_t** `man`, *bool* `destructive`, *ap_abstract0_t** `a`, *ap_lincons0_array_t** `array`) [Function]

`ap_abstract0_t* ap_abstract0_meet_tcons_array` (*ap_manager_t** `man`, *bool* `destructive`, *ap_abstract0_t** `a`, *ap_tcons0_array_t** `array`) [Function]

Meet of the abstract value *a* with the set of constraints *array*.

array should have exactly the same dimensionality as *a*.

`ap_abstract0_t* ap_abstract0_add_ray_array` (*ap_manager_t** `man`, *bool* `destructive`, *ap_abstract0_t** `a`, *ap_generator0_array_t** `array`) [Function]

Generalized time elapse operator.

array is supposed to contain only rays or lines, no vertices.

array should have exactly the same dimensionality as *a*.

8.7.10 Assignements and Substitutions of abstract values of level 0

`ap_abstract0_t* ap_abstract0_assign_linexpr_array` [Function]
(*ap_manager_t** **`man`**, *bool* **`destructive`**, *ap_abstract0_t** **`org`**, *ap_dim_t** **`tdim`**, *ap_linexpr0_t*** **`texpr`**, *size_t* **`size`**, *ap_abstract0_t** **`dest`**)

`ap_abstract0_t* ap_abstract0_substitute_linexpr_array` [Function]
(*ap_manager_t** **`man`**, *bool* **`destructive`**, *ap_abstract0_t** **`org`**, *ap_dim_t** **`tdim`**, *ap_linexpr0_t*** **`texpr`**, *size_t* **`size`**, *ap_abstract0_t** **`dest`**)

`ap_abstract0_t* ap_abstract0_assign_texpr_array` (*ap_manager_t** **`man`**, *bool* **`destructive`**, *ap_abstract0_t** **`org`**, *ap_dim_t** **`tdim`**, *ap_texpr0_t*** **`texpr`**, *size_t* **`size`**, *ap_abstract0_t** **`dest`**) [Function]

`ap_abstract0_t* ap_abstract0_substitute_texpr_array` [Function]
(*ap_manager_t** **`man`**, *bool* **`destructive`**, *ap_abstract0_t** **`org`**, *ap_dim_t** **`tdim`**, *ap_texpr0_t*** **`texpr`**, *size_t* **`size`**, *ap_abstract0_t** **`dest`**)

Parallel Assignement and Substitution of several dimensions by expressions in abstract value *org*.

dest is an optional argument. If not NULL, semantically speaking, the result of the transformation is intersected with *dest*. This is useful for precise backward transformations in lattices like intervals or octagons.

8.7.11 Existential quantification of abstract values of level 0

`ap_abstract0_t* ap_abstract0_forget_array` (*ap_manager_t** **`man`**, *bool* **`destructive`**, *ap_abstract0_t** **`a`**, *ap_dim_t** **`tdim`**, *size_t* **`size`**, *bool* **`project`**) [Function]

Forget (`project=false`) or Project (`project=true`) the array of dimensions *tdim* of size *size* in the abstract value *a*.

8.7.12 Change and permutation of dimensions of abstract values of level 0

`ap_abstract0_t* ap_abstract0_add_dimensions` (*ap_manager_t** **`man`**, *bool* **`destructive`**, *ap_abstract0_t** **`a`**, *ap_dimchange_t** **`dimchange`**, *bool* **`project`**) [Function]

`ap_abstract0_t* ap_abstract0_remove_dimensions` (*ap_manager_t** **`man`**, *bool* **`destructive`**, *ap_abstract0_t** **`a`**, *ap_dimchange_t** **`dimchange`**) [Function]

Addition and Removal of dimensions in a according to *dimchange*. In the case of addition, new dimensions are either unconstrained (`project==false`) or initialized to 0 ((`project==true`).

`ap_abstract0_t* ap_abstract0_apply_dimchange2` (*ap_manager_t** **`man`**, *bool* **`destructive`**, *ap_abstract0_t** **`a`**, *ap_dimchange2_t** **`dimchange2`**, *bool* **`project`**) [Function]

Apply the transformation specified by *dimchange2*. New dimensions are either unconstrained (`project==false`) or initialized to 0 ((`project==true`).

`ap_abstract0_t* ap_abstract0_permute_dimensions` (*ap_manager_t** **`man`**, *bool* **`destructive`**, *ap_abstract0_t** **`a`**, *ap_dimperm_t** **`perm`**) [Function]

Permute the dimensions of *a* according to the permutation *perm*.

The size of the permutation is supposed to be large enough w.r.t. *a*.

8.7.13 Expansion and Folding of dimensions of abstract values of level 0

Formally, expanding `z` into `z` and `w` in abstract value (predicate) `P` is defined by $expand(P(x, y, z), z, w) = P(x, y, z)andP(x, y, w)$.

Conversely, folding `z` and `w` into `z` in abstract value (predicate) `Q` is defined by $fold(Q(x, y, z, w), z, w) = (existsw : Q(x, y, z, w))or(existsz : Q(x, y, z, w)[z < -w])$.

`ap_abstract0_t* ap_abstract0_expand` (*ap_manager_t** **man**, *bool* **destructive**, *ap_abstract0_t** **a**, *ap_dim_t* **dim**, *size_t* **n**) [Function]

Expand 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 *dim* 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.

`ap_abstract0_t* ap_abstract0_fold` (*ap_manager_t** **man**, *bool* **destructive**, *ap_abstract0_t** **a**, *ap_dim_t** **tdim**, *size_t* **size**) [Function]

Fold the dimensions in the array *tdim* of size *size*>=1 and put the result in the first dimension in the array *assumed to be sorted*. The other dimensions of the array are then removed.

8.7.14 Widening of abstract values of level 0

`ap_abstract0_t* ap_abstract0_widening` (*ap_manager_t** **man**, *ap_abstract0_t** **a1**, *ap_abstract0_t** **a2**) [Function]

Widening of *a1* with *a2*. *a1* is supposed to be included in *a2*.

8.7.15 Topological closure of abstract values of level 0

`ap_abstract0_t* ap_abstract0_closure` (*ap_manager_t** **man**, *bool* **destructive**, *ap_abstract0_t** **a**) [Function]

Relax strict constraints into non strict constraints.

8.7.16 Additional functions on abstract values of level 0

These functions do not have corresponding functions into underlying libraries.

`ap_manager_t* ap_abstract0_manager` (*ap_abstract0_t** **a**) [Function]

Return a reference to the manager contained in *a*.

The reference should not be freed.

`ap_abstract0_t* ap_abstract0_of_lincons_array` (*ap_manager_t** **man**, *size_t* **intdim**, *size_t* **realdim**, *ap_lincons0_array_t** **array**) [Function]

`ap_abstract0_t* ap_abstract0_of_tcons_array` (*ap_manager_t** **man**, *size_t* **intdim**, *size_t* **realdim**, *ap_tcons0_array_t** **array**) [Function]

Abstract a conjunction of constraints. The constraints in the array should have exactly the dimensions (*intdim*,*realdim*).

`ap_abstract0_t* ap_abstract0_assign_linexpr` (*ap_manager_t** **man**, *bool* **destructive**, *ap_abstract0_t** **org**, *ap_dim_t* **dim**, *ap_linexpr0_t** **expr**, *ap_abstract0_t** **dest**) [Function] `ap_abstract0_t* ap_abstract0_substitute_linexpr` (*ap_manager_t* `man`, *bool* `destructive`, *ap_abstract0_t** `org`, *ap_dim_t* `dim`, *ap_linexpr0_t** `expr`, *ap_abstract0_t** `dest`) [Function]

`ap_abstract0_t* ap_abstract0_assign_texpr` (*ap_manager_t** `man`, *bool* `destructive`, *ap_abstract0_t** `org`, *ap_dim_t* `dim`, *ap_texpr0_t** `expr`, *ap_abstract0_t** `dest`) [Function]

`ap_abstract0_t* ap_abstract0_substitute_texpr` (*ap_manager_t** `man`, *bool* `destructive`, *ap_abstract0_t** `org`, *ap_dim_t* `dim`, *ap_texpr0_t** `expr`, *ap_abstract0_t** `dest`) [Function]

Assignement and Substitution of the dimension *dim* by the expression *expr* in abstract value *org*.

dest is an optional argument. If not NULL, semantically speaking, the result of the transformation is intersected with *dest*. This is useful for precise backward transformations in lattices like intervals or octagons.

`ap_abstract0_t* ap_abstract0_widening_threshold` (*ap_manager_t** `man`, *ap_abstract0_t** `a1`, *ap_abstract0_t** `a2`, *ap_lincons0_array_t** `array`) [Function]

Widening with threshold.

Intersect the result of the standard widening with all the constraints in *array* that are satisfied by both *a1* and *a2*.

9 Functions for implementors

The signatures and documentation of these functions are provided by the files 'ap_generic.h', 'ap_linearize.h' and 'ap_reducedproduct.h'.

These functions are dedicated to implementors of underlying libraries. They offer generic default implementations for some of the operations required by the APRON API, when there is no more specific and efficient implementation for the domain being implemented.

To use one of these, the function allocating manager, which is specific to the domain, should put the corresponding pointer in the virtual table to such a generic implementation.

They manipulated "unboxed" abstract values, which are native to the underlying library: they are not yet boxed with the manager in the type ap_abstract0_t.

10 Examples

Look at the examples in the '`example`' subdirectoy of the full distribution, or click on the links below:

Appendix A Appendices

A.1 C Datatype Index

A

P

T

A.2 C Variable Index

A.3 C Function Index

A

B

C

P