The Parma Polyhedra Library
C Language Interface
User's Manual*
(version 0.12.1)

Roberto Bagnara[†]
Patricia M. Hill[‡]
Enea Zaffanella[§]
Abramo Bagnara[¶]

April 16, 2012

*This work has been partly supported by: University of Parma's FIL scientific research project (ex 60%) "Pure and Applied Mathematics"; MURST project "Automatic Program Certification by Abstract Interpretation"; MURST project "Abstract Interpretation, Type Systems and Control-Flow Analysis"; MURST project "Automatic Aggregate- and Number-Reasoning for Computing: from Decision Algorithms to Constraint Programming with Multisets, Sets, and Maps"; MURST project "Constraint Based Verification of Reactive Systems"; MURST project "Abstract Interpretation: Design and Applications"; EPSRC project "Numerical Domains for Software Analysis"; EPSRC project "Geometric Abstractions for Scalable Program Analyzers".

[†]bagnara@cs.unipr.it, Department of Mathematics, University of Parma, Italy, and BUGSENG srl.

[‡]patricia.hill@bugseng.com, BUGSENG srl.

[§]zaffanella@cs.unipr.it, Department of Mathematics, University of Parma, Italy, and BUGSENG srl.

[¶]abramo.bagnara@bugseng.com, BUGSENG srl.

This document describes the Parma Polyhedra Library (PPL).

For the most up-to-date information see the Parma Polyhedra Library site:

http://bugseng.com/products/ppl/

Contents

1 Main Page

All the declarations needed for using the PPL's C interface (preprocessor symbols, data types, variables and functions) are collected in the header file `ppl_c.h`. This file, which is designed to work with pre-ANSI and ANSI C compilers as well as C99 and C++ compilers, should be included, either directly or via some other header file, with the directive

```
#include <ppl_c.h>
```

If this directive does not work, then your compiler is unable to find the file `ppl_c.h`. So check that the library is installed (if it is not installed, you may want to `make install`, perhaps with root privileges) in the right place (if not you may want to reconfigure the library using the appropriate pathname for the `--prefix` option); and that your compiler knows where it is installed (if not you should add the path to the directory where `ppl_c.h` is located to the compiler's include file search path; this is usually done with the `-I` option).

The name space of the PPL's C interface is `PPL_*` for preprocessor symbols, enumeration values and variables; and `ppl_*` for data types and function names. The interface systematically uses *opaque data types* (generic pointers that completely hide the internal representations from the client code) and provides all required access functions. By using just the interface, the client code can exploit all the functionalities of the library yet avoid directly manipulating the library's data structures. The advantages are that (1) applications do not depend on the internals of the library (these may change from release to release), and (2) the interface invariants can be thoroughly checked (by the access functions).

Note

> All functions taking as input argument an opaque pointer datatype assume that such an argument is actually *referring to a valid PPL object*. For instance, a function with an argument having type `ppl_MIP_Problem_t` will expect a valid MIP_Problem object, previously initialized by calling, e.g., `ppl_new_MIP_Problem`. If that is not the case (e.g., if a null pointer is passed in), the behavior is undefined.

The PPL's C interface is initialized by means of the `ppl_initialize` function. This function must be called *before using any other interface of the library*. The application can release the resources allocated by the library by calling the `ppl_finalize` function. After this function is called *no other interface of the library may be used* until the interface is re-initialized using `ppl_initialize`.

Any application using the PPL should make sure that only the intended version(s) of the library are ever used. The version used can be checked at compile-time thanks to the macros PPL_VERSION_MAJOR, PPL_VERSION_MINOR, PPL_VERSION_REVISION and PPL_VERSION_BETA, which give, respectively major, minor, revision and beta numbers of the PPL version. This is an example of their use:

```
#if PPL_VERSION_MAJOR == 0 && PPL_VERSION_MINOR < 6
# error "PPL version 0.6 or following is required"
#endif
```

Compile-time checking, however, is not normally enough, particularly in an environment where there is dynamic linking. Run-time checking can be performed by means of the functions `ppl_version_major`, `ppl_version_minor`, `ppl_version_revision`, and `ppl_version_beta`. The PPL's C interface also provides functions `ppl_version`, returning character string containing the full version number, and `ppl_banner`, returning a string that, in addition, provides (pointers to) other useful information for the library user.

All programs using the PPL's C interface must link with the following libraries: `libppl_c` (PPL's C interface), `libppl` (PPL's core), `libgmpxx` (GMP's C++ interface), and `libgmp` (GMP's library core). On most Unix-like systems, this is done by adding `-lppl_c`, `-lppl`, `-lgmpxx`, and `-lgmp` to the compiler's or linker's command line. For example:

```
gcc myprogram.o -lppl_c -lppl -lgmpxx -lgmp
```

If this does not work, it means that your compiler/linker is not finding the libraries where it expects. Again, this could be because you forgot to install the library or you installed it in a non-standard location. In the latter case you will need to use the appropriate options (usually `-L`) and, if you use shared libraries, some sort of run-time path selection mechanisms. Consult your compiler's documentation for details. Notice that the PPL is built using `Libtool` and an application can exploit this fact to significantly simplify the linking phase. See Libtool's documentation for details. Those working under Linux can find a lot of useful information on how to use program libraries (including static, shared, and dynamically loaded libraries) in the `Program Library HOWTO`.

For examples on how to use the functions provided by the C interface, you are referred to the directory `demos/ppl_lpsol/` in the source distribution. It contains a *Mixed Integer (Linear) Programming* solver written in C. In order to use this solver you will need to install `GLPK` (the GNU Linear Programming Kit): this is used to read linear programs in MPS format.

2 GNU General Public License

Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. `http://fsf.org/`

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, 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 them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The “Corresponding Source” for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work’s System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users’ Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work’s users, your or third parties’ legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program’s source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

- a) The work must carry prominent notices stating that you modified it, and giving a relevant date.
- b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
- c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
- d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

- a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
- b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.
- c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

- d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

- e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

- a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
- b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
- c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
- d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
- e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
- f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered “further restrictions” within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An “entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party’s predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A “contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor’s “contributor version”.

A contributor’s “essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, “control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor’s essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a “patent license” is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To “grant” such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. “Knowingly relying” means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient’s use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is “discriminatory” if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others’ Freedom.

If 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 convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General 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 Program specifies that a certain numbered version of the GNU General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the

Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "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 PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM 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 PROGRAM (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 PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state 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 program's name and a brief idea of what it does.
Copyright (C)  year   name of author
```

```
This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program 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 General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see http://www.gnu.org/licenses/.
```

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

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
 program  Copyright (C)  year   name of author
This program comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type 'show c' for details.
```

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see http://www.gnu.org/licenses/.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read http://www.gnu.org/philosophy/why-not-lgpl.html.

3 GNU Free Documentation License

Version 1.2, November 2002

```
Copyright (C) 2000,2001,2002  Free Software Foundation, Inc.
51 Franklin St, Fifth Floor, Boston, MA  02110-1301,  USA.
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.
```

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- **A.** Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- **B.** List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the

Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

- **C.** State on the Title page the name of the publisher of the Modified Version, as the publisher.
- **D.** Preserve all the copyright notices of the Document.
- **E.** Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- **F.** Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- **G.** Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- **H.** Include an unaltered copy of this License.
- **I.** Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- **J.** Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- **K.** For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- **L.** Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- **M.** Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- **N.** Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- **O.** Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

4 Module Index

4.1 Modules

Here is a list of all modules:

5 Class Index

5.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

6 Module Documentation

6.1 C Language Interface

The Parma Polyhedra Library comes equipped with an interface for the C language.

6.2 Library Initialization and Finalization

Functions

- int ppl_initialize (void)

  Initializes the Parma Polyhedra Library. This function must be called before any other function.
- int ppl_finalize (void)

  Finalizes the Parma Polyhedra Library. This function must be called after any other function.
- int ppl_set_rounding_for_PPL (void)

  Sets the FPU rounding mode so that the PPL abstractions based on floating point numbers work correctly.
- int ppl_restore_pre_PPL_rounding (void)

  Sets the FPU rounding mode as it was before initialization of the PPL.
- int ppl_irrational_precision (unsigned *p)

  Writes to `p` the precision parameter used for irrational calculations.
- int ppl_set_irrational_precision (unsigned p)

  Sets the precision parameter used for irrational calculations.

6.2.1 Detailed Description

Functions for initialization/finalization of the library, as well as setting/resetting of floating-point rounding mode.

6.2.2 Function Documentation

6.2.2.1 int ppl_initialize (void)

Initializes the Parma Polyhedra Library. This function must be called before any other function.

Returns

`PPL_ERROR_INVALID_ARGUMENT` if the library was already initialized.

6.2.2.2 int ppl_finalize (void)

Finalizes the Parma Polyhedra Library. This function must be called after any other function.

Returns

`PPL_ERROR_INVALID_ARGUMENT` if the library was already finalized.

6.2.2.3 int ppl_set_rounding_for_PPL (void)

Sets the FPU rounding mode so that the PPL abstractions based on floating point numbers work correctly.

This is performed automatically at initialization-time. Calling this function is needed only if restore_pre_-PPL_rounding() has been previously called.

6.2.2.4 int ppl_restore_pre_PPL_rounding (void)

Sets the FPU rounding mode as it was before initialization of the PPL.

After calling this function it is absolutely necessary to call set_rounding_for_PPL() before using any PPL abstractions based on floating point numbers. This is performed automatically at finalization-time.

6.2.2.5 int ppl_set_irrational_precision (unsigned *p*)

Sets the precision parameter used for irrational calculations.

If p is less than or equal to INT_MAX, sets the precision parameter used for irrational calculations to p. Then, in the irrational calculations returning an unbounded rational, (e.g., when computing a square root), the lesser between numerator and denominator will be limited to 2**p.

6.3 Version Checking

Macros

- #define PPL_VERSION "0.12.1"

  A string containing the PPL version.
- #define PPL_VERSION_MAJOR 0

  The major number of the PPL version.
- #define PPL_VERSION_MINOR 12

  The minor number of the PPL version.
- #define PPL_VERSION_REVISION 1

  The revision number of the PPL version.
- #define PPL_VERSION_BETA 0

  The beta number of the PPL version. This is zero for official releases and nonzero for development snapshots.

Functions

- int ppl_version_major (void)

  Returns the major number of the PPL version.
- int ppl_version_minor (void)

  Returns the minor number of the PPL version.
- int ppl_version_revision (void)

  Returns the revision number of the PPL version.
- int ppl_version_beta (void)

  Returns the beta number of the PPL version.
- int ppl_version (const char **p)

  Writes to `*p` a pointer to a character string containing the PPL version.
- int ppl_banner (const char **p)

  Writes to `*p` a pointer to a character string containing the PPL banner.

6.3.1 Detailed Description

Symbolic constants and functions related to library version checking.

6.3.2 Macro Definition Documentation

6.3.2.1 #define PPL_VERSION "0.12.1"

A string containing the PPL version.

Let `M` and `m` denote the numbers associated to PPL_VERSION_MAJOR and PPL_VERSION_MINOR, respectively. The format of PPL_VERSION is `M "." m` if both PPL_VERSION_REVISION (`r`) and PPL_VERSION_BETA (`b`)are zero, `M "." m "pre" b` if PPL_VERSION_REVISION is zero and PPL_VERSION_BETA is not zero, `M "." m "." r` if PPL_VERSION_REVISION is not zero and PPL_VERSION_BETA is zero, `M "." m "." r "pre" b` if neither PPL_VERSION_REVISION nor PPL_VERSION_BETA are zero.

6.3.3 Function Documentation

6.3.3.1 int ppl_banner (const char ** p)

Writes to *p a pointer to a character string containing the PPL banner.

The banner provides information about the PPL version, the licensing, the lack of any warranty whatsoever, the C++ compiler used to build the library, where to report bugs and where to look for further information.

6.4 Error Handling

Enumerations

- enum ppl_enum_error_code {
PPL_ERROR_OUT_OF_MEMORY, PPL_ERROR_INVALID_ARGUMENT, PPL_ERROR_DOMAIN_ERROR, PPL_ERROR_LENGTH_ERROR,
PPL_ARITHMETIC_OVERFLOW, PPL_STDIO_ERROR, PPL_ERROR_INTERNAL_ERROR,
PPL_ERROR_UNKNOWN_STANDARD_EXCEPTION,
PPL_ERROR_UNEXPECTED_ERROR, PPL_TIMEOUT_EXCEPTION, PPL_ERROR_LOGIC_ERROR }

  Defines the error codes that any function may return.

Functions

- int ppl_set_error_handler (void(∗h)(enum ppl_enum_error_code code, const char ∗description))

  Installs the user-defined error handler pointed at by `h`.

6.4.1 Detailed Description

Symbolic constants and functions related to error reporting/handling.

6.4.2 Enumeration Type Documentation

6.4.2.1 enum ppl_enum_error_code

Defines the error codes that any function may return.

Enumerator:

PPL_ERROR_OUT_OF_MEMORY The virtual memory available to the process has been exhausted.

PPL_ERROR_INVALID_ARGUMENT A function has been invoked with an invalid argument.

PPL_ERROR_DOMAIN_ERROR A function has been invoked outside its domain of definition.

PPL_ERROR_LENGTH_ERROR The construction of an object that would exceed its maximum permitted size was attempted.

PPL_ARITHMETIC_OVERFLOW An arithmetic overflow occurred and the computation was consequently interrupted. This can *only* happen in library's incarnations using bounded integers as coefficients.

PPL_STDIO_ERROR An error occurred during a C input/output operation. A more precise indication of what went wrong is available via `errno`.

PPL_ERROR_INTERNAL_ERROR An internal error that was diagnosed by the PPL itself. This indicates a bug in the PPL.

PPL_ERROR_UNKNOWN_STANDARD_EXCEPTION A standard exception has been raised by the C++ run-time environment. This indicates a bug in the PPL.

PPL_ERROR_UNEXPECTED_ERROR A totally unknown, totally unexpected error happened. This indicates a bug in the PPL.

PPL_TIMEOUT_EXCEPTION An exception has been raised by the PPL as a timeout previously set by the user has expired.

PPL_ERROR_LOGIC_ERROR The client program attempted to use the PPL in a way that violates its internal logic. This happens, for instance, when the client attempts to use the timeout facilities on a system that does not support them.

6.4.3 Function Documentation

6.4.3.1 int ppl_set_error_handler (void(*)(enum ppl_enum_error_code code, const char *description) *h*)

Installs the user-defined error handler pointed at by `h`.

The error handler takes an error code and a textual description that gives further information about the actual error. The C string containing the textual description is read-only and its existence is not guaranteed after the handler has returned.

6.5 Timeout Handling

Functions

- int ppl_set_timeout (unsigned csecs)

  Sets the timeout for computations whose completion could require an exponential amount of time.
- int ppl_reset_timeout (void)

  Resets the timeout time so that the computation is not interrupted.
- int ppl_set_deterministic_timeout (unsigned long unscaled_weight, unsigned scale)

  Sets a threshold for computations whose completion could require an exponential amount of time.
- int ppl_reset_deterministic_timeout (void)

  Resets the deterministic timeout so that the computation is not interrupted.

6.5.1 Detailed Description

Functions for setting and resetting timeouts.

6.5.2 Function Documentation

6.5.2.1 int ppl_set_timeout (unsigned *csecs*)

Sets the timeout for computations whose completion could require an exponential amount of time.

Parameters

csecs	The number of centiseconds sometimes after which a timeout will occur; it must be strictly greater than zero.

Computations taking exponential time will be interrupted some time after `csecs` centiseconds have elapsed since the call to the timeout setting function. If the computation is interrupted that way, the interrupted function will return error code `PPL_TIMEOUT_EXCEPTION`. Otherwise, if the computation completes without being interrupted, then the timeout should be reset by calling `ppl_reset_timeout()`.

6.5.2.2 int ppl_set_deterministic_timeout (unsigned long *unscaled_weight*, unsigned *scale*)

Sets a threshold for computations whose completion could require an exponential amount of time.

Returns

`PPL_ERROR_INVALID_ARGUMENT` if `unscaled_weight` is zero or if the computed weight threshold exceeds the maximum allowed value.

Parameters

unscaled_-weight	The unscaled maximum computational weight; it has to be non-zero.
scale	The scaling factor to be applied to `unscaled_weight`.

If `unscaled_weight` has value u and `scale` has value s, then the (scaled) weight threshold is computed as $w = u \cdot 2^s$. Computations taking exponential time will be interrupted some time after reaching the complexity threshold w. If the computation is interrupted that way, the interrupted function will return error code PPL_TIMEOUT_EXCEPTION. Otherwise, if the computation completes without being interrupted, then the deterministic timeout should be reset by calling ppl_reset_deterministic_timeout().

Note

This "timeout" checking functionality is said to be *deterministic* because it is not based on actual elapsed time. Its behavior will only depend on (some of the) computations performed in the PPL library and it will be otherwise independent from the computation environment (CPU, operating system, compiler, etc.).

Warning

The weight mechanism is under beta testing. In particular, there is still no clear relation between the weight threshold and the actual computational complexity. As a consequence, client applications should be ready to reconsider the tuning of these weight thresholds when upgrading to newer version of the PPL.

6.6 Library Datatypes

Typedefs for the library datatypes and related symbolic constants.

Typedefs

- typedef size_t ppl_dimension_type

    An unsigned integral type for representing space dimensions.
- typedef const char ∗ ppl_io_variable_output_function_type (ppl_dimension_type var)

    The type of output functions used for printing variables.
- typedef struct
ppl_Coefficient_tag ∗ ppl_Coefficient_t

    Opaque pointer.
- typedef struct
ppl_Coefficient_tag const ∗ ppl_const_Coefficient_t

    Opaque pointer to const object.
- typedef struct
ppl_Linear_Expression_tag ∗ ppl_Linear_Expression_t

    Opaque pointer.
- typedef struct
ppl_Linear_Expression_tag
const ∗ ppl_const_Linear_Expression_t

    Opaque pointer to const object.
- typedef struct ppl_Constraint_tag ∗ ppl_Constraint_t

    Opaque pointer.
- typedef struct
ppl_Constraint_tag const ∗ ppl_const_Constraint_t

    Opaque pointer to const object.
- typedef struct
ppl_Constraint_System_tag ∗ ppl_Constraint_System_t

    Opaque pointer.
- typedef struct
ppl_Constraint_System_tag
const ∗ ppl_const_Constraint_System_t

    Opaque pointer to const object.
- typedef struct
ppl_Constraint_System_const_iterator_tag ∗ ppl_Constraint_System_const_iterator_t

    Opaque pointer.
- typedef struct
ppl_Constraint_System_const_iterator_tag
const ∗ ppl_const_Constraint_System_const_iterator_t

    Opaque pointer to const object.
- typedef struct ppl_Generator_tag ∗ ppl_Generator_t

    Opaque pointer.
- typedef struct
ppl_Generator_tag const ∗ ppl_const_Generator_t

    Opaque pointer to const object.
- typedef struct
ppl_Generator_System_tag ∗ ppl_Generator_System_t

Opaque pointer.

- typedef struct ppl_Generator_System_tag const ∗ ppl_const_Generator_System_t

  Opaque pointer to const object.
- typedef struct ppl_Generator_System_const_iterator_tag ∗ ppl_Generator_System_const_iterator_t

  Opaque pointer.
- typedef struct ppl_Generator_System_const_iterator_tag const ∗ ppl_const_Generator_System_const_iterator_t

  Opaque pointer to const object.
- typedef struct ppl_Congruence_tag ∗ ppl_Congruence_t

  Opaque pointer.
- typedef struct ppl_Congruence_tag const ∗ ppl_const_Congruence_t

  Opaque pointer to const object.
- typedef struct ppl_Congruence_System_tag ∗ ppl_Congruence_System_t

  Opaque pointer.
- typedef struct ppl_Congruence_System_tag const ∗ ppl_const_Congruence_System_t

  Opaque pointer to const object.
- typedef struct ppl_Congruence_System_const_iterator_tag ∗ ppl_Congruence_System_const_iterator_t

  Opaque pointer.
- typedef struct ppl_Congruence_System_const_iterator_tag const ∗ ppl_const_Congruence_System_const_iterator_t

  Opaque pointer to const object.
- typedef struct ppl_Grid_Generator_tag ∗ ppl_Grid_Generator_t

  Opaque pointer.
- typedef struct ppl_Grid_Generator_tag const ∗ ppl_const_Grid_Generator_t

  Opaque pointer to const object.
- typedef struct ppl_Grid_Generator_System_tag ∗ ppl_Grid_Generator_System_t

  Opaque pointer.
- typedef struct ppl_Grid_Generator_System_tag const ∗ ppl_const_Grid_Generator_System_t

  Opaque pointer to const object.
- typedef struct ppl_Grid_Generator_System_const_iterator_tag ∗ ppl_Grid_Generator_System_const_iterator_t

  Opaque pointer.
- typedef struct ppl_Grid_Generator_System_const_iterator_tag const ∗ ppl_const_Grid_Generator_System_const_iterator_t

Opaque pointer to const object.

- typedef struct
  ppl_MIP_Problem_tag ∗ ppl_MIP_Problem_t

  Opaque pointer.
- typedef struct
  ppl_MIP_Problem_tag const ∗ ppl_const_MIP_Problem_t

  Opaque pointer to const object.
- typedef struct
  ppl_PIP_Problem_tag ∗ ppl_PIP_Problem_t

  Opaque pointer.
- typedef struct
  ppl_PIP_Problem_tag const ∗ ppl_const_PIP_Problem_t

  Opaque pointer to const object.
- typedef struct
  ppl_PIP_Tree_Node_tag ∗ ppl_PIP_Tree_Node_t

  Opaque pointer.
- typedef struct
  ppl_PIP_Tree_Node_tag const ∗ ppl_const_PIP_Tree_Node_t

  Opaque pointer to const object.
- typedef struct
  ppl_PIP_Decision_Node_tag ∗ ppl_PIP_Decision_Node_t

  Opaque pointer.
- typedef struct
  ppl_PIP_Decision_Node_tag
  const ∗ ppl_const_PIP_Decision_Node_t

  Opaque pointer to const object.
- typedef struct
  ppl_PIP_Solution_Node_tag ∗ ppl_PIP_Solution_Node_t

  Opaque pointer.
- typedef struct
  ppl_PIP_Solution_Node_tag
  const ∗ ppl_const_PIP_Solution_Node_t

  Opaque pointer to const object.
- typedef struct
  ppl_Artificial_Parameter_tag ∗ ppl_Artificial_Parameter_t

  Opaque pointer.
- typedef struct
  ppl_Artificial_Parameter_tag
  const ∗ ppl_const_Artificial_Parameter_t

  Opaque pointer to const object.
- typedef struct
  ppl_Artificial_Parameter_Sequence_tag ∗ ppl_Artificial_Parameter_Sequence_t

  Opaque pointer.
- typedef struct
  ppl_Artificial_Parameter_Sequence_tag
  const ∗ ppl_const_Artificial_Parameter_Sequence_t

  Opaque pointer to const object.
- typedef struct
  ppl_Artificial_Parameter_Sequence_const_iterator_tag ∗ ppl_Artificial_Parameter_Sequence_const-_iterator_t

Opaque pointer.
- typedef struct
ppl_Artificial_Parameter_Sequence_const_iterator_tag
const ∗ ppl_const_Artificial_Parameter_Sequence_const_iterator_t

  Opaque pointer to const object.
- typedef struct ppl_Polyhedron_tag ∗ ppl_Polyhedron_t

  Opaque pointer.
- typedef struct
ppl_Polyhedron_tag const ∗ ppl_const_Polyhedron_t

  Opaque pointer to const object.
- typedef struct
ppl_Pointset_Powerset_C_Polyhedron_tag ∗ ppl_Pointset_Powerset_C_Polyhedron_t

  Opaque pointer.
- typedef struct
ppl_Pointset_Powerset_C_Polyhedron_tag
const ∗ ppl_const_Pointset_Powerset_C_Polyhedron_t

  Opaque pointer to const object.
- typedef struct
ppl_Pointset_Powerset_C_Polyhedron_iterator_tag ∗ ppl_Pointset_Powerset_C_Polyhedron_iterator-_t

  Opaque pointer.
- typedef struct
ppl_Pointset_Powerset_C_Polyhedron_iterator_tag
const ∗ ppl_const_Pointset_Powerset_C_Polyhedron_iterator_t

  Opaque pointer to const object.
- typedef struct
ppl_Pointset_Powerset_C_Polyhedron_const_iterator_tag ∗ ppl_Pointset_Powerset_C_Polyhedron-_const_iterator_t

  Opaque pointer.
- typedef struct
ppl_Pointset_Powerset_C_Polyhedron_const_iterator_tag
const ∗ ppl_const_Pointset_Powerset_C_Polyhedron_const_iterator_t

  Opaque pointer to const object.

Enumerations

- enum ppl_enum_Constraint_Type {
PPL_CONSTRAINT_TYPE_LESS_THAN, PPL_CONSTRAINT_TYPE_LESS_OR_EQUAL, P-PL_CONSTRAINT_TYPE_EQUAL, PPL_CONSTRAINT_TYPE_GREATER_OR_EQUAL, PPL_CONSTRAINT_TYPE_GREATER_THAN }

  Describes the relations represented by a constraint.
- enum ppl_enum_Generator_Type { PPL_GENERATOR_TYPE_LINE, PPL_GENERATOR_TY-PE_RAY, PPL_GENERATOR_TYPE_POINT, PPL_GENERATOR_TYPE_CLOSURE_POINT }

  Describes the different kinds of generators.
- enum ppl_enum_Grid_Generator_Type { PPL_GRID_GENERATOR_TYPE_LINE, PPL_GRID_-GENERATOR_TYPE_PARAMETER, PPL_GRID_GENERATOR_TYPE_POINT }

  Describes the different kinds of grid generators.

- enum ppl_enum_Bounded_Integer_Type_Width {
PPL_BITS_8, PPL_BITS_16, PPL_BITS_32, PPL_BITS_64,
PPL_BITS_128 }

  Widths of bounded integer types.
- enum ppl_enum_Bounded_Integer_Type_Representation { PPL_UNSIGNED, PPL_SIGNED_2_C-OMPLEMENT }

  Representation of bounded integer types.
- enum ppl_enum_Bounded_Integer_Type_Overflow { PPL_OVERFLOW_WRAPS, PPL_OVERF-LOW_UNDEFINED, PPL_OVERFLOW_IMPOSSIBLE }

  Overflow behavior of bounded integer types.

Functions

- int ppl_max_space_dimension (ppl_dimension_type ∗m)

  Writes to `m` *the maximum space dimension this library can handle.*
- int ppl_not_a_dimension (ppl_dimension_type ∗m)

  Writes to `m` *a value that does not designate a valid dimension.*
- int ppl_io_print_variable (ppl_dimension_type var)

  Pretty-prints `var` *to* `stdout`*.*
- int ppl_io_fprint_variable (FILE ∗stream, ppl_dimension_type var)

  Pretty-prints `var` *to the given output* `stream`*.*
- int ppl_io_asprint_variable (char ∗∗strp, ppl_dimension_type var)

  Pretty-prints `var` *to a malloc-allocated string, a pointer to which is returned via* `strp`*.*
- int ppl_io_set_variable_output_function (ppl_io_variable_output_function_type ∗p)

  Sets the output function to be used for printing variables to `p`*.*
- int ppl_io_get_variable_output_function (ppl_io_variable_output_function_type ∗∗pp)

  Writes a pointer to the current variable output function to `pp`*.*
- char ∗ ppl_io_wrap_string (const char ∗src, unsigned indent_depth, unsigned preferred_first_line_-length, unsigned preferred_line_length)

  Utility function for the wrapping of lines of text.

Variables

- unsigned int PPL_COMPLEXITY_CLASS_POLYNOMIAL

  Code of the worst-case polynomial complexity class.
- unsigned int PPL_COMPLEXITY_CLASS_SIMPLEX

  Code of the worst-case exponential but typically polynomial complexity class.
- unsigned int PPL_COMPLEXITY_CLASS_ANY

  Code of the universal complexity class.
- unsigned int PPL_POLY_CON_RELATION_IS_DISJOINT

  Individual bit saying that the polyhedron and the set of points satisfying the constraint are disjoint.
- unsigned int PPL_POLY_CON_RELATION_STRICTLY_INTERSECTS

  Individual bit saying that the polyhedron intersects the set of points satisfying the constraint, but it is not included in it.
- unsigned int PPL_POLY_CON_RELATION_IS_INCLUDED

  Individual bit saying that the polyhedron is included in the set of points satisfying the constraint.
- unsigned int PPL_POLY_CON_RELATION_SATURATES

Individual bit saying that the polyhedron is included in the set of points saturating the constraint.

- unsigned int PPL_POLY_GEN_RELATION_SUBSUMES

  Individual bit saying that adding the generator would not change the polyhedron.

6.6.1 Detailed Description

Typedefs for the library datatypes and related symbolic constants. The datatypes provided by the library should be manipulated by means of the corresponding opaque pointer types and the functions working on them.

Note

To simplify the detection of common programming mistakes, we provide both pointer-to-const and pointer-to-nonconst opaque pointers, with implicit conversions mapping each pointer-to-nonconst to the corresponding pointer-to-const when needed. The user of the C interface is therefore recommended to adopt the pointer-to-const type whenever read-only access is meant.

6.6.2 Typedef Documentation

6.6.2.1 typedef const char* ppl_io_variable_output_function_type(ppl_dimension_type var)

The type of output functions used for printing variables.

An output function for variables must write a textual representation for `var` to a character buffer, null-terminate it, and return a pointer to the beginning of the buffer. In case the operation fails, 0 should be returned and perhaps `errno` should be set in a meaningful way. The library does nothing with the buffer, besides printing its contents.

6.6.3 Enumeration Type Documentation

6.6.3.1 enum ppl_enum_Constraint_Type

Describes the relations represented by a constraint.

Enumerator:

PPL_CONSTRAINT_TYPE_LESS_THAN The constraint is of the form $e < 0$.

PPL_CONSTRAINT_TYPE_LESS_OR_EQUAL The constraint is of the form $e \leq 0$.

PPL_CONSTRAINT_TYPE_EQUAL The constraint is of the form $e = 0$.

PPL_CONSTRAINT_TYPE_GREATER_OR_EQUAL The constraint is of the form $e \geq 0$.

PPL_CONSTRAINT_TYPE_GREATER_THAN The constraint is of the form $e > 0$.

6.6.3.2 enum ppl_enum_Generator_Type

Describes the different kinds of generators.

Enumerator:

PPL_GENERATOR_TYPE_LINE The generator is a line.

PPL_GENERATOR_TYPE_RAY The generator is a ray.

PPL_GENERATOR_TYPE_POINT The generator is a point.

PPL_GENERATOR_TYPE_CLOSURE_POINT The generator is a closure point.

6.6.3.3 enum ppl_enum_Grid_Generator_Type

Describes the different kinds of grid generators.

Enumerator:

PPL_GRID_GENERATOR_TYPE_LINE The grid generator is a line.

PPL_GRID_GENERATOR_TYPE_PARAMETER The grid generator is a parameter.

PPL_GRID_GENERATOR_TYPE_POINT The grid generator is a point.

6.6.3.4 enum ppl_enum_Bounded_Integer_Type_Width

Widths of bounded integer types.

Enumerator:

PPL_BITS_8 8 bits.

PPL_BITS_16 16 bits.

PPL_BITS_32 32 bits.

PPL_BITS_64 64 bits.

PPL_BITS_128 128 bits.

6.6.3.5 enum ppl_enum_Bounded_Integer_Type_Representation

Representation of bounded integer types.

Enumerator:

PPL_UNSIGNED Unsigned binary.

PPL_SIGNED_2_COMPLEMENT Signed binary where negative values are represented by the two's complement of the absolute value.

6.6.3.6 enum ppl_enum_Bounded_Integer_Type_Overflow

Overflow behavior of bounded integer types.

Enumerator:

PPL_OVERFLOW_WRAPS On overflow, wrapping takes place. This means that, for a w-bit bounded integer, the computation happens modulo 2^w.

PPL_OVERFLOW_UNDEFINED On overflow, the result is undefined. This simply means that the result of the operation resulting in an overflow can take any value.

Note

Even though something more serious can happen in the system being analyzed ---due to, e.g., C's undefined behavior---, here we are only concerned with the results of arithmetic operations. It is the responsibility of the analyzer to ensure that other manifestations of undefined behavior are conservatively approximated.

PPL_OVERFLOW_IMPOSSIBLE Overflow is impossible. This is for the analysis of languages where overflow is trapped before it affects the state, for which, thus, any indication that an overflow may have affected the state is necessarily due to the imprecision of the analysis.

6.6.4 Function Documentation

6.6.4.1 char* ppl_io_wrap_string (const char * *src*, unsigned *indent_depth*, unsigned *preferred_first_line_length*, unsigned *preferred_line_length*)

Utility function for the wrapping of lines of text.

Parameters

src	The source string holding the text to wrap.
indent_depth	The indentation depth.
preferred_first-_line_length	The preferred length for the first line of text.
preferred_line-_length	The preferred length for all the lines but the first one.

Returns

The wrapped string in a malloc-allocated buffer.

7 Class Documentation

7.1 ppl_Artificial_Parameter_Sequence_const_iterator_tag Interface Reference

Types and functions for iterating on PIP artificial parameters.

`#include <ppl_c_header.h>`

Related Functions

(Note that these are not member functions.)

Constructors, Assignment and Destructor

- int ppl_new_Artificial_Parameter_Sequence_const_iterator (ppl_Artificial_Parameter_Sequence_const_iterator_t *papit)

  Builds a new 'const iterator' and writes a handle to it at address `papit`*.*
- int ppl_new_Artificial_Parameter_Sequence_const_iterator_from_Artificial_Parameter_Sequence_const_iterator (ppl_Artificial_Parameter_Sequence_const_iterator_t *papit, ppl_const_Artificial_Parameter_Sequence_const_iterator_t apit)

  Builds a const iterator that is a copy of `apit`*; writes a handle for the newly created const iterator at address* `papit`*.*
- int ppl_assign_Artificial_Parameter_Sequence_const_iterator_from_Artificial_Parameter_Sequence_const_iterator (ppl_Artificial_Parameter_Sequence_const_iterator_t dst, ppl_const_Artificial_Parameter_Sequence_const_iterator_t src)

  Assigns a copy of the const iterator `src` *to* `dst`*.*
- int ppl_delete_Artificial_Parameter_Sequence_const_iterator (ppl_const_Artificial_Parameter_Sequence_const_iterator_t apit)

  Invalidates the handle `apit`*: this makes sure the corresponding resources will eventually be released.*

Dereferencing, Incrementing and Equality Testing

- int ppl_Artificial_Parameter_Sequence_const_iterator_dereference (ppl_const_Artificial_Parameter_Sequence_const_iterator_t apit, ppl_const_Artificial_Parameter_t *pap)

  Dereference `apit` *writing a const handle to the resulting artificial parameter at address* `pap`*.*
- int ppl_Artificial_Parameter_Sequence_const_iterator_increment (ppl_Artificial_Parameter_Sequence_const_iterator_t apit)

  Increment `apit` *so that it "points" to the next artificial parameter.*
- int ppl_Artificial_Parameter_Sequence_const_iterator_equal_test (ppl_const_Artificial_Parameter_Sequence_const_iterator_t x, ppl_const_Artificial_Parameter_Sequence_const_iterator_t y)

  Returns a positive integer if the iterators corresponding to `x` *and* `y` *are equal; returns 0 if they are different.*

7.1.1 Detailed Description

Types and functions for iterating on PIP artificial parameters.

The documentation for this interface was generated from the following file:

- ppl_c_header.h

7.2 ppl_Artificial_Parameter_tag Interface Reference

Types and functions for PIP artificial parameters.

`#include <ppl_c_header.h>`

Related Functions

(Note that these are not member functions.)

- int ppl_Artificial_Parameter_get_Linear_Expression (ppl_const_Artificial_Parameter_t ap, ppl_Linear_Expression_t le)

    Copies into `le` *the linear expression in artificial parameter* `ap`.
- int ppl_Artificial_Parameter_coefficient (ppl_const_Artificial_Parameter_t ap, ppl_dimension_type var, ppl_Coefficient_t n)

    Copies into `n` *the coefficient of variable* `var` *in the artificial parameter* `ap`.
- int ppl_Artificial_Parameter_get_inhomogeneous_term (ppl_const_Artificial_Parameter_t ap, ppl_Coefficient_t n)

    Copies into `n` *the inhomogeneous term of the artificial parameter* `ap`.
- int ppl_Artificial_Parameter_denominator (ppl_const_Artificial_Parameter_t ap, ppl_Coefficient_t n)

    Copies into `n` *the denominator in artificial parameter* `ap`.

Input/Output Functions

- int ppl_io_print_Artificial_Parameter (ppl_const_Artificial_Parameter_t x)

    Prints `x` *to* `stdout`.
- int ppl_io_fprint_Artificial_Parameter (FILE ∗stream, ppl_const_Artificial_Parameter_t x)

    Prints `x` *to the given output* `stream`.
- int ppl_io_asprint_Artificial_Parameter (char ∗∗strp, ppl_const_Artificial_Parameter_t x)

    Prints `x` *to a malloc-allocated string, a pointer to which is returned via* `strp`.
- int ppl_Artificial_Parameter_ascii_dump (ppl_const_Artificial_Parameter_t x, FILE ∗stream)

    Dumps an ascii representation of `x` *on* `stream`.
- int ppl_Artificial_Parameter_ascii_load (ppl_Artificial_Parameter_t x, FILE ∗stream)

    Loads an ascii representation of `x` *from* `stream`.

7.2.1 Detailed Description

Types and functions for PIP artificial parameters.

The types and functions for PIP artificial parameters provide an interface towards *Artificial_Parameter*.

The documentation for this interface was generated from the following file:

- ppl_c_header.h

7.3 ppl_Coefficient_tag Interface Reference

Types and functions for coefficients.

`#include <ppl_c_header.h>`

Related Functions

(Note that these are not member functions.)

Constructors, Assignment and Destructor

- int ppl_new_Coefficient (ppl_Coefficient_t *pc)
  Creates a new coefficient with value 0 and writes a handle for the newly created coefficient at address `pc`*.*
- int ppl_new_Coefficient_from_mpz_t (ppl_Coefficient_t *pc, mpz_t z)
  Creates a new coefficient with the value given by the GMP integer `z` *and writes a handle for the newly created coefficient at address* `pc`*.*
- int ppl_new_Coefficient_from_Coefficient (ppl_Coefficient_t *pc, ppl_const_Coefficient_t c)
  Builds a coefficient that is a copy of `c`*; writes a handle for the newly created coefficient at address* `pc`*.*
- int ppl_assign_Coefficient_from_mpz_t (ppl_Coefficient_t dst, mpz_t z)
  Assign to `dst` *the value given by the GMP integer* `z`*.*
- int ppl_assign_Coefficient_from_Coefficient (ppl_Coefficient_t dst, ppl_const_Coefficient_t src)
  Assigns a copy of the coefficient `src` *to* `dst`*.*
- int ppl_delete_Coefficient (ppl_const_Coefficient_t c)
  Invalidates the handle `c`*: this makes sure the corresponding resources will eventually be released.*

Read-Only Accessor Functions

- int ppl_Coefficient_to_mpz_t (ppl_const_Coefficient_t c, mpz_t z)
  Sets the value of the GMP integer `z` *to the value of* `c`*.*
- int ppl_Coefficient_OK (ppl_const_Coefficient_t c)
  Returns a positive integer if `c` *is well formed, i.e., if it satisfies all its implementation invariants; returns 0 and perhaps makes some noise if* `c` *is broken. Useful for debugging purposes.*
- int ppl_Coefficient_is_bounded (void)
  Returns a positive integer if coefficients are bounded; returns 0 otherwise.
- int ppl_Coefficient_min (mpz_t min)
  Returns a positive integer if coefficients are bounded, in which case `min` *is set to their minimum value; returns 0 otherwise.*
- int ppl_Coefficient_max (mpz_t max)
  Returns a positive integer if coefficients are bounded, in which case `max` *is set to their maximum value; returns 0 otherwise.*

I/O Functions

- int ppl_io_print_Coefficient (ppl_const_Coefficient_t x)
  Prints `x` *to* `stdout`*.*
- int ppl_io_fprint_Coefficient (FILE *stream, ppl_const_Coefficient_t x)
  Prints `x` *to the given output* `stream`*.*
- int ppl_io_asprint_Coefficient (char **strp, ppl_const_Coefficient_t x)
  Prints `x` *to a malloc-allocated string, a pointer to which is returned via* `strp`*.*

7.3.1 Detailed Description

Types and functions for coefficients.

The types and functions for coefficients provide an interface towards *Coefficient*. Depending on configuration, the PPL coefficients may be implemented by the unbounded precision integers provided by GMP (default), or by bounded precision integers (with checks for overflows).

The documentation for this interface was generated from the following file:

- ppl_c_header.h

7.4 ppl_Congruence_System_const_iterator_tag Interface Reference

Types and functions for iterating on congruence systems.

`#include <ppl_c_header.h>`

Related Functions

(Note that these are not member functions.)

Constructors, Assignment and Destructor

- int ppl_new_Congruence_System_const_iterator (ppl_Congruence_System_const_iterator_t *pcit)
  Builds a new 'const iterator' and writes a handle to it at address `pcit`.
- int ppl_new_Congruence_System_const_iterator_from_Congruence_System_const_iterator (ppl_Congruence_System_const_iterator_t *pcit, ppl_const_Congruence_System_const_iterator_t cit)
  Builds a const iterator that is a copy of `cit`*; writes a handle for the newly created const iterator at address* `pcit`.
- int ppl_assign_Congruence_System_const_iterator_from_Congruence_System_const_iterator (ppl_Congruence_System_const_iterator_t dst, ppl_const_Congruence_System_const_iterator_t src)
  Assigns a copy of the const iterator `src` *to* `dst`.
- int ppl_delete_Congruence_System_const_iterator (ppl_const_Congruence_System_const_iterator_t cit)
  Invalidates the handle `cit`*: this makes sure the corresponding resources will eventually be released.*

Dereferencing, Incrementing and Equality Testing

- int ppl_Congruence_System_const_iterator_dereference (ppl_const_Congruence_System_const_iterator_t cit, ppl_const_Congruence_t *pc)
  Dereference `cit` *writing a const handle to the resulting congruence at address* `pc`.
- int ppl_Congruence_System_const_iterator_increment (ppl_Congruence_System_const_iterator_t cit)
  Increment `cit` *so that it "points" to the next congruence.*
- int ppl_Congruence_System_const_iterator_equal_test (ppl_const_Congruence_System_const_iterator_t x, ppl_const_Congruence_System_const_iterator_t y)
  Returns a positive integer if the iterators corresponding to `x` *and* `y` *are equal; returns 0 if they are different.*

7.4.1 Detailed Description

Types and functions for iterating on congruence systems.

The types and functions for congruence systems iterators provide read-only access to the elements of a congruence system by interfacing *Congruence_System::const_iterator*.

The documentation for this interface was generated from the following file:

- ppl_c_header.h

7.5 ppl_Congruence_System_tag Interface Reference

Types and functions for congruence systems.

`#include <ppl_c_header.h>`

Related Functions

(Note that these are not member functions.)

Constructors, Assignment and Destructor

- int ppl_new_Congruence_System (ppl_Congruence_System_t *pcs)
  Builds an empty system of congruences and writes a handle to it at address `pcs`.
- int ppl_new_Congruence_System_zero_dim_empty (ppl_Congruence_System_t *pcs)
  Builds a zero-dimensional, unsatisfiable congruence system and writes a handle to it at address `pcs`.
- int ppl_new_Congruence_System_from_Congruence (ppl_Congruence_System_t *pcs, ppl_const_Congruence_t c)
  Builds the singleton congruence system containing only a copy of congruence `c`*; writes a handle for the newly created system at address* `pcs`.
- int ppl_new_Congruence_System_from_Congruence_System (ppl_Congruence_System_t *pcs, ppl_const_Congruence_System_t cs)
  Builds a congruence system that is a copy of `cs`*; writes a handle for the newly created system at address* `pcs`.
- int ppl_assign_Congruence_System_from_Congruence_System (ppl_Congruence_System_t dst, ppl_const_Congruence_System_t src)
  Assigns a copy of the congruence system `src` *to* `dst`.
- int ppl_delete_Congruence_System (ppl_const_Congruence_System_t cs)
  Invalidates the handle `cs`*: this makes sure the corresponding resources will eventually be released.*

Functions that Do Not Modify the Congruence System

- int ppl_Congruence_System_space_dimension (ppl_const_Congruence_System_t cs, ppl_dimension_type *m)
  Writes to `m` *the dimension of the vector space enclosing* `cs`.
- int ppl_Congruence_System_empty (ppl_const_Congruence_System_t cs)
  Returns a positive integer if `cs` *contains no (non-trivial) congruence; returns 0 otherwise.*
- int ppl_Congruence_System_begin (ppl_const_Congruence_System_t cs, ppl_Congruence_System_const_iterator_t cit)
  Assigns to `cit` *a const iterator "pointing" to the beginning of the congruence system* `cs`.
- int ppl_Congruence_System_end (ppl_const_Congruence_System_t cs, ppl_Congruence_System_const_iterator_t cit)
  Assigns to `cit` *a const iterator "pointing" past the end of the congruence system* `cs`.
- int ppl_Congruence_System_OK (ppl_const_Congruence_System_t cs)
  Returns a positive integer if `cs` *is well formed, i.e., if it satisfies all its implementation invariants; returns 0 and perhaps makes some noise if* `cs` *is broken. Useful for debugging purposes.*

Functions that May Modify the Congruence System

- int ppl_Congruence_System_clear (ppl_Congruence_System_t cs)
  Removes all the congruences from the congruence system `cs` *and sets its space dimension to 0.*
- int ppl_Congruence_System_insert_Congruence (ppl_Congruence_System_t cs, ppl_const_Congruence_t c)
  Inserts a copy of the congruence `c` *into* `cs`*; the space dimension is increased, if necessary.*

Input/Output Functions

- int ppl_io_print_Congruence_System (ppl_const_Congruence_System_t x)

Prints x *to* stdout.

- int ppl_io_fprint_Congruence_System (FILE *stream, ppl_const_Congruence_System_t x)

    Prints x *to the given output* stream.
- int ppl_io_asprint_Congruence_System (char **strp, ppl_const_Congruence_System_t x)

    Prints x *to a malloc-allocated string, a pointer to which is returned via* strp.
- int ppl_Congruence_System_ascii_dump (ppl_const_Congruence_System_t x, FILE *stream)

    Dumps an ascii representation of x *on* stream.
- int ppl_Congruence_System_ascii_load (ppl_Congruence_System_t x, FILE *stream)

    Loads an ascii representation of x *from* stream.

7.5.1 Detailed Description

Types and functions for congruence systems.

The types and functions for congruence systems provide an interface towards *Congruence_System*.

The documentation for this interface was generated from the following file:

- ppl_c_header.h

7.6 ppl_Congruence_tag Interface Reference

Types and functions for congruences.

```
#include <ppl_c_header.h>
```

Related Functions

(Note that these are not member functions.)

Constructors, Assignment and Destructor

- int ppl_new_Congruence (ppl_Congruence_t *pc, ppl_const_Linear_Expression_t le, ppl_const_Coefficient_t m)

    Creates the new congruence $le = 0 \pmod{m}$ *and writes a handle for it at address* pc. *The space dimension of the new congruence is equal to the space dimension of* le.
- int ppl_new_Congruence_zero_dim_false (ppl_Congruence_t *pc)

    Creates the unsatisfiable (zero-dimension space) congruence $0 = 1 \pmod{0}$ *and writes a handle for it at address* pc.
- int ppl_new_Congruence_zero_dim_integrality (ppl_Congruence_t *pc)

    Creates the true (zero-dimension space) congruence $0 = 1 \pmod{1}$*, also known as integrality congruence. A handle for the newly created congruence is written at address* pc.
- int ppl_new_Congruence_from_Congruence (ppl_Congruence_t *pc, ppl_const_Congruence_t c)

    Builds a congruence that is a copy of c*; writes a handle for the newly created congruence at address* pc.
- int ppl_assign_Congruence_from_Congruence (ppl_Congruence_t dst, ppl_const_Congruence_t src)

    Assigns a copy of the congruence src *to* dst.
- int ppl_delete_Congruence (ppl_const_Congruence_t c)

    Invalidates the handle c*: this makes sure the corresponding resources will eventually be released.*

Functions that Do Not Modify the Congruence

- int ppl_Congruence_space_dimension (ppl_const_Congruence_t c, ppl_dimension_type *m)
  Writes to `m` *the space dimension of* `c`.
- int ppl_Congruence_coefficient (ppl_const_Congruence_t c, ppl_dimension_type var, ppl_Coefficient_t n)
  Copies into `n` *the coefficient of variable* `var` *in congruence* `c`.
- int ppl_Congruence_inhomogeneous_term (ppl_const_Congruence_t c, ppl_Coefficient_t n)
  Copies into `n` *the inhomogeneous term of congruence* `c`.
- int ppl_Congruence_modulus (ppl_const_Congruence_t c, ppl_Coefficient_t m)
  Copies into `m` *the modulus of congruence* `c`.
- int ppl_Congruence_OK (ppl_const_Congruence_t c)
  Returns a positive integer if `c` *is well formed, i.e., if it satisfies all its implementation invariants; returns 0 and perhaps makes some noise if* `c` *is broken. Useful for debugging purposes.*

Input/Output Functions

- int ppl_io_print_Congruence (ppl_const_Congruence_t x)
  Prints `x` *to* `stdout`.
- int ppl_io_fprint_Congruence (FILE *stream, ppl_const_Congruence_t x)
  Prints `x` *to the given output* `stream`.
- int ppl_io_asprint_Congruence (char **strp, ppl_const_Congruence_t x)
  Prints `x` *to a malloc-allocated string, a pointer to which is returned via* `strp`.
- int ppl_Congruence_ascii_dump (ppl_const_Congruence_t x, FILE *stream)
  Dumps an ascii representation of `x` *on* `stream`.
- int ppl_Congruence_ascii_load (ppl_Congruence_t x, FILE *stream)
  Loads an ascii representation of `x` *from* `stream`.

7.6.1 Detailed Description

Types and functions for congruences.

The types and functions for congruences provide an interface towards *Congruence*.

The documentation for this interface was generated from the following file:

- ppl_c_header.h

7.7 ppl_Constraint_System_const_iterator_tag Interface Reference

Types and functions for iterating on constraint systems.

`#include <ppl_c_header.h>`

Related Functions

(Note that these are not member functions.)

Constructors, Assignment and Destructor

- int ppl_new_Constraint_System_const_iterator (ppl_Constraint_System_const_iterator_t *pcit)
  Builds a new 'const iterator' and writes a handle to it at address `pcit`.
- int ppl_new_Constraint_System_const_iterator_from_Constraint_System_const_iterator (ppl_Constraint_System_const_iterator_t *pcit, ppl_const_Constraint_System_const_iterator_t cit)

Builds a const iterator that is a copy of `cit`*; writes a handle for the newly created const iterator at address* `pcit`*.*

- int ppl_assign_Constraint_System_const_iterator_from_Constraint_System_const_iterator (ppl_Constraint_System_const_iterator_t dst, ppl_const_Constraint_System_const_iterator_t src)

  Assigns a copy of the const iterator `src` *to* `dst`*.*
- int ppl_delete_Constraint_System_const_iterator (ppl_const_Constraint_System_const_iterator_t cit)

  Invalidates the handle `cit`*: this makes sure the corresponding resources will eventually be released.*

Dereferencing, Incrementing and Equality Testing

- int ppl_Constraint_System_const_iterator_dereference (ppl_const_Constraint_System_const_iterator_t cit, ppl_const_Constraint_t *pc)

  Dereference `cit` *writing a const handle to the resulting constraint at address* `pc`*.*
- int ppl_Constraint_System_const_iterator_increment (ppl_Constraint_System_const_iterator_t cit)

  Increment `cit` *so that it "points" to the next constraint.*
- int ppl_Constraint_System_const_iterator_equal_test (ppl_const_Constraint_System_const_iterator_t x, ppl_const_Constraint_System_const_iterator_t y)

  Returns a positive integer if the iterators corresponding to `x` *and* `y` *are equal; returns 0 if they are different.*

7.7.1 Detailed Description

Types and functions for iterating on constraint systems.

The types and functions for constraint systems iterators provide read-only access to the elements of a constraint system by interfacing *Constraint_System::const_iterator*.

The documentation for this interface was generated from the following file:

- ppl_c_header.h

7.8 ppl_Constraint_System_tag Interface Reference

Types and functions for constraint systems.

`#include <ppl_c_header.h>`

Related Functions

(Note that these are not member functions.)

Constructors, Assignment and Destructor

- int ppl_new_Constraint_System (ppl_Constraint_System_t *pcs)

  Builds an empty system of constraints and writes a handle to it at address `pcs`*.*
- int ppl_new_Constraint_System_zero_dim_empty (ppl_Constraint_System_t *pcs)

  Builds a zero-dimensional, unsatisfiable constraint system and writes a handle to it at address `pcs`*.*
- int ppl_new_Constraint_System_from_Constraint (ppl_Constraint_System_t *pcs, ppl_const_Constraint_t c)

  Builds the singleton constraint system containing only a copy of constraint `c`*; writes a handle for the newly created system at address* `pcs`*.*

- int ppl_new_Constraint_System_from_Constraint_System (ppl_Constraint_System_t *pcs, ppl_const_Constraint_System_t cs)

  Builds a constraint system that is a copy of `cs`*; writes a handle for the newly created system at address* `pcs`*.*
- int ppl_assign_Constraint_System_from_Constraint_System (ppl_Constraint_System_t dst, ppl_const_Constraint_System_t src)

  Assigns a copy of the constraint system `src` *to* `dst`*.*
- int ppl_delete_Constraint_System (ppl_const_Constraint_System_t cs)

  Invalidates the handle `cs`*: this makes sure the corresponding resources will eventually be released.*

Functions that Do Not Modify the Constraint System

- int ppl_Constraint_System_space_dimension (ppl_const_Constraint_System_t cs, ppl_dimension_type *m)

  Writes to `m` *the dimension of the vector space enclosing* `cs`*.*
- int ppl_Constraint_System_empty (ppl_const_Constraint_System_t cs)

  Returns a positive integer if `cs` *contains no (non-trivial) constraint; returns 0 otherwise.*
- int ppl_Constraint_System_has_strict_inequalities (ppl_const_Constraint_System_t cs)

  Returns a positive integer if `cs` *contains any (non-trivial) strict inequality; returns 0 otherwise.*
- int ppl_Constraint_System_begin (ppl_const_Constraint_System_t cs, ppl_Constraint_System_const_iterator_t cit)

  Assigns to `cit` *a const iterator "pointing" to the beginning of the constraint system* `cs`*.*
- int ppl_Constraint_System_end (ppl_const_Constraint_System_t cs, ppl_Constraint_System_const_iterator_t cit)

  Assigns to `cit` *a const iterator "pointing" past the end of the constraint system* `cs`*.*
- int ppl_Constraint_System_OK (ppl_const_Constraint_System_t cs)

  Returns a positive integer if `cs` *is well formed, i.e., if it satisfies all its implementation invariants; returns 0 and perhaps makes some noise if* `cs` *is broken. Useful for debugging purposes.*

Functions that May Modify the Constraint System

- int ppl_Constraint_System_clear (ppl_Constraint_System_t cs)

  Removes all the constraints from the constraint system `cs` *and sets its space dimension to 0.*
- int ppl_Constraint_System_insert_Constraint (ppl_Constraint_System_t cs, ppl_const_Constraint_t c)

  Inserts a copy of the constraint `c` *into* `cs`*; the space dimension is increased, if necessary.*

Input/Output Functions

- int ppl_io_print_Constraint_System (ppl_const_Constraint_System_t x)

  Prints `x` *to* `stdout`*.*
- int ppl_io_fprint_Constraint_System (FILE *stream, ppl_const_Constraint_System_t x)

  Prints `x` *to the given output* `stream`*.*
- int ppl_io_asprint_Constraint_System (char **strp, ppl_const_Constraint_System_t x)

  Prints `x` *to a malloc-allocated string, a pointer to which is returned via* `strp`*.*
- int ppl_Constraint_System_ascii_dump (ppl_const_Constraint_System_t x, FILE *stream)

  Dumps an ascii representation of `x` *on* `stream`*.*
- int ppl_Constraint_System_ascii_load (ppl_Constraint_System_t x, FILE *stream)

  Loads an ascii representation of `x` *from* `stream`*.*

7.8.1 Detailed Description

Types and functions for constraint systems.

The types and functions for constraint systems provide an interface towards *Constraint_System*.

The documentation for this interface was generated from the following file:

- ppl_c_header.h

7.9 ppl_Constraint_tag Interface Reference

Types and functions for constraints.

`#include <ppl_c_header.h>`

Related Functions

(Note that these are not member functions.)

Constructors, Assignment and Destructor

- int ppl_new_Constraint (ppl_Constraint_t *pc, ppl_const_Linear_Expression_t le, enum ppl_enum_Constraint_Type rel)
  Creates the new constraint '`le rel` 0' and writes a handle for it at address `pc`. The space dimension of the new constraint is equal to the space dimension of `le`.
- int ppl_new_Constraint_zero_dim_false (ppl_Constraint_t *pc)
  Creates the unsatisfiable (zero-dimension space) constraint $0 = 1$ and writes a handle for it at address `pc`.
- int ppl_new_Constraint_zero_dim_positivity (ppl_Constraint_t *pc)
  Creates the true (zero-dimension space) constraint $0 \leq 1$, also known as positivity constraint. A handle for the newly created constraint is written at address `pc`.
- int ppl_new_Constraint_from_Constraint (ppl_Constraint_t *pc, ppl_const_Constraint_t c)
  Builds a constraint that is a copy of `c`; writes a handle for the newly created constraint at address `pc`.
- int ppl_assign_Constraint_from_Constraint (ppl_Constraint_t dst, ppl_const_Constraint_t src)
  Assigns a copy of the constraint `src` to `dst`.
- int ppl_delete_Constraint (ppl_const_Constraint_t c)
  Invalidates the handle `c`: this makes sure the corresponding resources will eventually be released.

Functions that Do Not Modify the Constraint

- int ppl_Constraint_space_dimension (ppl_const_Constraint_t c, ppl_dimension_type *m)
  Writes to `m` the space dimension of `c`.
- int ppl_Constraint_type (ppl_const_Constraint_t c)
  Returns the type of constraint `c`.
- int ppl_Constraint_coefficient (ppl_const_Constraint_t c, ppl_dimension_type var, ppl_Coefficient_t n)
  Copies into `n` the coefficient of variable `var` in constraint `c`.
- int ppl_Constraint_inhomogeneous_term (ppl_const_Constraint_t c, ppl_Coefficient_t n)
  Copies into `n` the inhomogeneous term of constraint `c`.
- int ppl_Constraint_OK (ppl_const_Constraint_t c)
  Returns a positive integer if `c` is well formed, i.e., if it satisfies all its implementation invariants; returns 0 and perhaps makes some noise if `c` is broken. Useful for debugging purposes.

Input/Output Functions

- int ppl_io_print_Constraint (ppl_const_Constraint_t x)
  Prints `x` *to* `stdout`*.*
- int ppl_io_fprint_Constraint (FILE ∗stream, ppl_const_Constraint_t x)
  Prints `x` *to the given output* `stream`*.*
- int ppl_io_asprint_Constraint (char ∗∗strp, ppl_const_Constraint_t x)
  Prints `x` *to a malloc-allocated string, a pointer to which is returned via* `strp`*.*
- int ppl_Constraint_ascii_dump (ppl_const_Constraint_t x, FILE ∗stream)
  Dumps an ascii representation of `x` *on* `stream`*.*
- int ppl_Constraint_ascii_load (ppl_Constraint_t x, FILE ∗stream)
  Loads an ascii representation of `x` *from* `stream`*.*

7.9.1 Detailed Description

Types and functions for constraints.

The types and functions for constraints provide an interface towards *Constraint*.

The documentation for this interface was generated from the following file:

- ppl_c_header.h

7.10 ppl_Generator_System_const_iterator_tag Interface Reference

Types and functions for iterating on generator systems.

`#include <ppl_c_header.h>`

Related Functions

(Note that these are not member functions.)

Constructors, Assignment and Destructor

- int ppl_new_Generator_System_const_iterator (ppl_Generator_System_const_iterator_t ∗pgit)
  Builds a new 'const iterator' and writes a handle to it at address `pgit`*.*
- int ppl_new_Generator_System_const_iterator_from_Generator_System_const_iterator (ppl_Generator-_System_const_iterator_t ∗pgit, ppl_const_Generator_System_const_iterator_t git)
  Builds a const iterator that is a copy of `git`*; writes a handle for the newly created const iterator at address* `pgit`*.*
- int ppl_assign_Generator_System_const_iterator_from_Generator_System_const_iterator (ppl_-Generator_System_const_iterator_t dst, ppl_const_Generator_System_const_iterator_t src)
  Assigns a copy of the const iterator `src` *to* `dst`*.*
- int ppl_delete_Generator_System_const_iterator (ppl_const_Generator_System_const_iterator_t git)
  Invalidates the handle `git` *: this makes sure the corresponding resources will eventually be released.*

Dereferencing, Incrementing and Equality Testing

- int ppl_Generator_System_const_iterator_dereference (ppl_const_Generator_System_const_iterator-_t git, ppl_const_Generator_t ∗pg)

Dereference `git` writing a const handle to the resulting generator at address `pg`.

- int ppl_Generator_System_const_iterator_increment (ppl_Generator_System_const_iterator_t git)

  Increment `git` so that it "points" to the next generator.

- int ppl_Generator_System_const_iterator_equal_test (ppl_const_Generator_System_const_iterator_t x, ppl_const_Generator_System_const_iterator_t y)

  Returns a positive integer if the iterators corresponding to `x` and `y` are equal; returns 0 if they are different.

7.10.1 Detailed Description

Types and functions for iterating on generator systems.

The types and functions for generator systems iterators provide read-only access to the elements of a generator system by interfacing *Generator_System::const_iterator*.

The documentation for this interface was generated from the following file:

- ppl_c_header.h

7.11 ppl_Generator_System_tag Interface Reference

Types and functions for generator systems.

```
#include <ppl_c_header.h>
```

Related Functions

(Note that these are not member functions.)

Constructors, Assignment and Destructor

- int ppl_new_Generator_System (ppl_Generator_System_t ∗pgs)

  Builds an empty system of generators and writes a handle to it at address `pgs`.

- int ppl_new_Generator_System_from_Generator (ppl_Generator_System_t ∗pgs, ppl_const_Generator_t g)

  Builds the singleton generator system containing only a copy of generator `g`; writes a handle for the newly created system at address `pgs`.

- int ppl_new_Generator_System_from_Generator_System (ppl_Generator_System_t ∗pgs, ppl_const_Generator_System_t gs)

  Builds a generator system that is a copy of `gs`; writes a handle for the newly created system at address `pgs`.

- int ppl_assign_Generator_System_from_Generator_System (ppl_Generator_System_t dst, ppl_const_Generator_System_t src)

  Assigns a copy of the generator system `src` to `dst`.

- int ppl_delete_Generator_System (ppl_const_Generator_System_t gs)

  Invalidates the handle `gs`: this makes sure the corresponding resources will eventually be released.

Functions that Do Not Modify the Generator System

- int ppl_Generator_System_space_dimension (ppl_const_Generator_System_t gs, ppl_dimension_type ∗m)

  Writes to `m` the dimension of the vector space enclosing `gs`.

- int ppl_Generator_System_empty (ppl_const_Generator_System_t gs)

    Returns a positive integer if `gs` *contains no generators; returns 0 otherwise.*
- int ppl_Generator_System_begin (ppl_const_Generator_System_t gs, ppl_Generator_System_-const_iterator_t git)

    Assigns to `git` *a const iterator "pointing" to the beginning of the generator system* `gs`*.*
- int ppl_Generator_System_end (ppl_const_Generator_System_t gs, ppl_Generator_System_const-_iterator_t git)

    Assigns to `git` *a const iterator "pointing" past the end of the generator system* `gs`*.*
- int ppl_Generator_System_OK (ppl_const_Generator_System_t gs)

    Returns a positive integer if `gs` *is well formed, i.e., if it satisfies all its implementation invariants; returns 0 and perhaps makes some noise if* `gs` *is broken. Useful for debugging purposes.*

Functions that May Modify the Generator System

- int ppl_Generator_System_clear (ppl_Generator_System_t gs)

    Removes all the generators from the generator system `gs` *and sets its space dimension to 0.*
- int ppl_Generator_System_insert_Generator (ppl_Generator_System_t gs, ppl_const_Generator-_t g)

    Inserts a copy of the generator `g` *into* `gs`*; the space dimension is increased, if necessary.*

Input/Output Functions

- int ppl_io_print_Generator_System (ppl_const_Generator_System_t x)

    Prints `x` *to* `stdout`*.*
- int ppl_io_fprint_Generator_System (FILE *stream, ppl_const_Generator_System_t x)

    Prints `x` *to the given output* `stream`*.*
- int ppl_io_asprint_Generator_System (char **strp, ppl_const_Generator_System_t x)

    Prints `x` *to a malloc-allocated string, a pointer to which is returned via* `strp`*.*
- int ppl_Generator_System_ascii_dump (ppl_const_Generator_System_t x, FILE *stream)

    Dumps an ascii representation of `x` *on* `stream`*.*
- int ppl_Generator_System_ascii_load (ppl_Generator_System_t x, FILE *stream)

    Loads an ascii representation of `x` *from* `stream`*.*

7.11.1 Detailed Description

Types and functions for generator systems.

The types and functions for generator systems provide an interface towards *Generator_System*.

The documentation for this interface was generated from the following file:

- ppl_c_header.h

7.12 ppl_Generator_tag Interface Reference

Types and functions for generators.

```
#include <ppl_c_header.h>
```

Related Functions

(Note that these are not member functions.)

Constructors, Assignment and Destructor

- int ppl_new_Generator (ppl_Generator_t *pg, ppl_const_Linear_Expression_t le, enum ppl_enum_Generator_Type t, ppl_const_Coefficient_t d)

  Creates a new generator of direction `le` *and type* `t`*. If the generator to be created is a point or a closure point, the divisor* `d` *is applied to* `le`*. For other types of generators* `d` *is simply disregarded. A handle for the new generator is written at address* `pg`*. The space dimension of the new generator is equal to the space dimension of* `le`*.*
- int ppl_new_Generator_zero_dim_point (ppl_Generator_t *pg)

  Creates the point that is the origin of the zero-dimensional space $\mathbb{R}^0$*. Writes a handle for the new generator at address* `pg`*.*
- int ppl_new_Generator_zero_dim_closure_point (ppl_Generator_t *pg)

  Creates, as a closure point, the point that is the origin of the zero-dimensional space $\mathbb{R}^0$*. Writes a handle for the new generator at address* `pg`*.*
- int ppl_new_Generator_from_Generator (ppl_Generator_t *pg, ppl_const_Generator_t g)

  Builds a generator that is a copy of `g`*; writes a handle for the newly created generator at address* `pg`*.*
- int ppl_assign_Generator_from_Generator (ppl_Generator_t dst, ppl_const_Generator_t src)

  Assigns a copy of the generator `src` *to* `dst`*.*
- int ppl_delete_Generator (ppl_const_Generator_t g)

  Invalidates the handle `g`*: this makes sure the corresponding resources will eventually be released.*

Functions that Do Not Modify the Generator

- int ppl_Generator_space_dimension (ppl_const_Generator_t g, ppl_dimension_type *m)

  Writes to `m` *the space dimension of* `g`*.*
- int ppl_Generator_type (ppl_const_Generator_t g)

  Returns the type of generator `g`*.*
- int ppl_Generator_coefficient (ppl_const_Generator_t g, ppl_dimension_type var, ppl_Coefficient_t n)

  Copies into `n` *the coefficient of variable* `var` *in generator* `g`*.*
- int ppl_Generator_divisor (ppl_const_Generator_t g, ppl_Coefficient_t n)

  If `g` *is a point or a closure point assigns its divisor to* `n`*.*
- int ppl_Generator_OK (ppl_const_Generator_t g)

  Returns a positive integer if `g` *is well formed, i.e., if it satisfies all its implementation invariants; returns 0 and perhaps makes some noise if* `g` *is broken. Useful for debugging purposes.*

Input/Output Functions

- int ppl_io_print_Generator (ppl_const_Generator_t x)

  Prints `x` *to* `stdout`*.*
- int ppl_io_fprint_Generator (FILE *stream, ppl_const_Generator_t x)

  Prints `x` *to the given output* `stream`*.*
- int ppl_io_asprint_Generator (char **strp, ppl_const_Generator_t x)

  Prints `x` *to a malloc-allocated string, a pointer to which is returned via* `strp`*.*
- int ppl_Generator_ascii_dump (ppl_const_Generator_t x, FILE *stream)

  Dumps an ascii representation of `x` *on* `stream`*.*
- int ppl_Generator_ascii_load (ppl_Generator_t x, FILE *stream)

  Loads an ascii representation of `x` *from* `stream`*.*

7.12.1 Detailed Description

Types and functions for generators.

The types and functions for generators provide an interface towards *Generator*.

The documentation for this interface was generated from the following file:

- ppl_c_header.h

7.13 ppl_Grid_Generator_System_const_iterator_tag Interface Reference

Types and functions for iterating on grid generator systems.

`#include <ppl_c_header.h>`

Related Functions

(Note that these are not member functions.)

Constructors, Assignment and Destructor

- int ppl_new_Grid_Generator_System_const_iterator (ppl_Grid_Generator_System_const_iterator-_t ∗pgit)

    Builds a new 'const iterator' and writes a handle to it at address `pgit`*.*
- int ppl_new_Grid_Generator_System_const_iterator_from_Grid_Generator_System_const_iterator (ppl_Grid_Generator_System_const_iterator_t ∗pgit, ppl_const_Grid_Generator_System_const-_iterator_t git)

    Builds a const iterator that is a copy of `git`*; writes a handle for the newly created const iterator at address* `pgit`*.*
- int ppl_assign_Grid_Generator_System_const_iterator_from_Grid_Generator_System_const_iterator (ppl_Grid_Generator_System_const_iterator_t dst, ppl_const_Grid_Generator_System_const_iterator-_t src)

    Assigns a copy of the const iterator `src` *to* `dst`*.*
- int ppl_delete_Grid_Generator_System_const_iterator (ppl_const_Grid_Generator_System_const-_iterator_t git)

    Invalidates the handle `git`*: this makes sure the corresponding resources will eventually be released.*

Dereferencing, Incrementing and Equality Testing

- int ppl_Grid_Generator_System_const_iterator_dereference (ppl_const_Grid_Generator_System-_const_iterator_t git, ppl_const_Grid_Generator_t ∗pg)

    Dereference `git` *writing a const handle to the resulting grid generator at address* `pg`*.*
- int ppl_Grid_Generator_System_const_iterator_increment (ppl_Grid_Generator_System_const_-iterator_t git)

    Increment `git` *so that it "points" to the next grid generator.*
- int ppl_Grid_Generator_System_const_iterator_equal_test (ppl_const_Grid_Generator_System-_const_iterator_t x, ppl_const_Grid_Generator_System_const_iterator_t y)

    Returns a positive integer if the iterators corresponding to `x` *and* `y` *are equal; returns 0 if they are different.*

7.13.1 Detailed Description

Types and functions for iterating on grid generator systems.

The types and functions for grid generator systems iterators provide read-only access to the elements of a grid generator system by interfacing *Grid_Generator_System::const_iterator*.

The documentation for this interface was generated from the following file:

- ppl_c_header.h

7.14 ppl_Grid_Generator_System_tag Interface Reference

Types and functions for grid generator systems.

`#include <ppl_c_header.h>`

Related Functions

(Note that these are not member functions.)

Constructors, Assignment and Destructor

- int ppl_new_Grid_Generator_System (ppl_Grid_Generator_System_t *pgs)
  Builds an empty system of grid generators and writes a handle to it at address `pgs`.
- int ppl_new_Grid_Generator_System_from_Grid_Generator (ppl_Grid_Generator_System_t *pgs, ppl_const_Grid_Generator_t g)
  Builds the singleton grid generator system containing only a copy of generator `g`*; writes a handle for the newly created system at address* `pgs`.
- int ppl_new_Grid_Generator_System_from_Grid_Generator_System (ppl_Grid_Generator_System_t *pgs, ppl_const_Grid_Generator_System_t gs)
  Builds a grid generator system that is a copy of `gs`*; writes a handle for the newly created system at address* `pgs`.
- int ppl_assign_Grid_Generator_System_from_Grid_Generator_System (ppl_Grid_Generator_System_t dst, ppl_const_Grid_Generator_System_t src)
  Assigns a copy of the grid generator system `src` *to* `dst`.
- int ppl_delete_Grid_Generator_System (ppl_const_Grid_Generator_System_t gs)
  Invalidates the handle `gs`*: this makes sure the corresponding resources will eventually be released.*

Functions that Do Not Modify the Grid Generator System

- int ppl_Grid_Generator_System_space_dimension (ppl_const_Grid_Generator_System_t gs, ppl_dimension_type *m)
  Writes to `m` *the dimension of the vector space enclosing* `gs`.
- int ppl_Grid_Generator_System_empty (ppl_const_Grid_Generator_System_t gs)
  Returns a positive integer if `gs` *contains no generator; returns 0 otherwise.*
- int ppl_Grid_Generator_System_begin (ppl_const_Grid_Generator_System_t gs, ppl_Grid_Generator_System_const_iterator_t git)
  Assigns to `git` *a const iterator "pointing" to the beginning of the grid generator system* `gs`.
- int ppl_Grid_Generator_System_end (ppl_const_Grid_Generator_System_t gs, ppl_Grid_Generator_System_const_iterator_t git)
  Assigns to `git` *a const iterator "pointing" past the end of the grid generator system* `gs`.
- int ppl_Grid_Generator_System_OK (ppl_const_Grid_Generator_System_t gs)

Returns a positive integer if `gs` *is well formed, i.e., if it satisfies all its implementation invariants; returns 0 and perhaps makes some noise if* `gs` *is broken. Useful for debugging purposes.*

Functions that May Modify the Grid Generator System

- int ppl_Grid_Generator_System_clear (ppl_Grid_Generator_System_t gs)
  Removes all the generators from the grid generator system `gs` *and sets its space dimension to 0.*
- int ppl_Grid_Generator_System_insert_Grid_Generator (ppl_Grid_Generator_System_t gs, ppl_const_Grid_Generator_t g)
  Inserts a copy of the grid generator `g` *into* `gs`*; the space dimension is increased, if necessary.*

Input/Output Functions

- int ppl_io_print_Grid_Generator_System (ppl_const_Grid_Generator_System_t x)
  Prints `x` *to* `stdout`*.*
- int ppl_io_fprint_Grid_Generator_System (FILE ∗stream, ppl_const_Grid_Generator_System_t x)
  Prints `x` *to the given output* `stream`*.*
- int ppl_io_asprint_Grid_Generator_System (char ∗∗strp, ppl_const_Grid_Generator_System_t x)
  Prints `x` *to a malloc-allocated string, a pointer to which is returned via* `strp`*.*
- int ppl_Grid_Generator_System_ascii_dump (ppl_const_Grid_Generator_System_t x, FILE ∗stream)
  Dumps an ascii representation of `x` *on* `stream`*.*
- int ppl_Grid_Generator_System_ascii_load (ppl_Grid_Generator_System_t x, FILE ∗stream)
  Loads an ascii representation of `x` *from* `stream`*.*

7.14.1 Detailed Description

Types and functions for grid generator systems.

The types and functions for grid generator systems provide an interface towards *Grid_Generator_System*.

The documentation for this interface was generated from the following file:

- ppl_c_header.h

7.15 ppl_Grid_Generator_tag Interface Reference

Types and functions for grid generators.

`#include <ppl_c_header.h>`

Related Functions

(Note that these are not member functions.)

Constructors, Assignment and Destructor

- int ppl_new_Grid_Generator (ppl_Grid_Generator_t ∗pg, ppl_const_Linear_Expression_t le, enum ppl_enum_Grid_Generator_Type t, ppl_const_Coefficient_t d)
  Creates a new grid generator of direction `le` *and type* `t`*. If the grid generator to be created is a point or a parameter, the divisor* `d` *is applied to* `le`*. If it is a line,* `d` *is simply disregarded. A handle for the new grid generator is written at address* `pg`*. The space dimension of the new grid generator is equal to the space dimension of* `le`*.*

- int ppl_new_Grid_Generator_zero_dim_point (ppl_Grid_Generator_t *pg)

  Creates the point that is the origin of the zero-dimensional space $\mathbb{R}^0$*. Writes a handle for the new grid generator at address* `pg`*.*
- int ppl_new_Grid_Generator_from_Grid_Generator (ppl_Grid_Generator_t *pg, ppl_const_Grid_Generator_t g)

  Builds a grid generator that is a copy of `g`*; writes a handle for the newly created grid generator at address* `pg`*.*
- int ppl_assign_Grid_Generator_from_Grid_Generator (ppl_Grid_Generator_t dst, ppl_const_Grid_Generator_t src)

  Assigns a copy of the grid generator `src` *to* `dst`*.*
- int ppl_delete_Grid_Generator (ppl_const_Grid_Generator_t g)

  Invalidates the handle `g`*: this makes sure the corresponding resources will eventually be released.*

Functions that Do Not Modify the Grid Generator

- int ppl_Grid_Generator_space_dimension (ppl_const_Grid_Generator_t g, ppl_dimension_type *m)

  Writes to `m` *the space dimension of* `g`*.*
- int ppl_Grid_Generator_type (ppl_const_Grid_Generator_t g)

  Returns the type of grid generator `g`*.*
- int ppl_Grid_Generator_coefficient (ppl_const_Grid_Generator_t g, ppl_dimension_type var, ppl_Coefficient_t n)

  Copies into `n` *the coefficient of variable* `var` *in grid generator* `g`*.*
- int ppl_Grid_Generator_divisor (ppl_const_Grid_Generator_t g, ppl_Coefficient_t n)

  If `g` *is a point or a parameter assigns its divisor to* `n`*.*
- int ppl_Grid_Generator_OK (ppl_const_Grid_Generator_t g)

  Returns a positive integer if `g` *is well formed, i.e., if it satisfies all its implementation invariants; returns 0 and perhaps makes some noise if* `g` *is broken. Useful for debugging purposes.*

Input/Output Functions

- int ppl_io_print_Grid_Generator (ppl_const_Grid_Generator_t x)

  Prints `x` *to* `stdout`*.*
- int ppl_io_fprint_Grid_Generator (FILE *stream, ppl_const_Grid_Generator_t x)

  Prints `x` *to the given output* `stream`*.*
- int ppl_io_asprint_Grid_Generator (char **strp, ppl_const_Grid_Generator_t x)

  Prints `x` *to a malloc-allocated string, a pointer to which is returned via* `strp`*.*
- int ppl_Grid_Generator_ascii_dump (ppl_const_Grid_Generator_t x, FILE *stream)

  Dumps an ascii representation of `x` *on* `stream`*.*
- int ppl_Grid_Generator_ascii_load (ppl_Grid_Generator_t x, FILE *stream)

  Loads an ascii representation of `x` *from* `stream`*.*

7.15.1 Detailed Description

Types and functions for grid generators.

The types and functions for grid generators provide an interface towards *Grid_Generator*.

The documentation for this interface was generated from the following file:

- ppl_c_header.h

7.16 ppl_Linear_Expression_tag Interface Reference

Types and functions for linear expressions.

`#include <ppl_c_header.h>`

Related Functions

(Note that these are not member functions.)

Constructors, Assignment and Destructor

- int ppl_new_Linear_Expression (ppl_Linear_Expression_t *ple)
  Creates a new linear expression corresponding to the constant 0 in a zero-dimensional space; writes a handle for the new linear expression at address `ple`.
- int ppl_new_Linear_Expression_with_dimension (ppl_Linear_Expression_t *ple, ppl_dimension_type d)
  Creates a new linear expression corresponding to the constant 0 in a `d`*-dimensional space; writes a handle for the new linear expression at address* `ple`.
- int ppl_new_Linear_Expression_from_Linear_Expression (ppl_Linear_Expression_t *ple, ppl_const_Linear_Expression_t le)
  Builds a linear expression that is a copy of `le`*; writes a handle for the newly created linear expression at address* `ple`.
- int ppl_new_Linear_Expression_from_Constraint (ppl_Linear_Expression_t *ple, ppl_const_Constraint_t c)
  Builds a linear expression corresponding to constraint `c`*; writes a handle for the newly created linear expression at address* `ple`.
- int ppl_new_Linear_Expression_from_Generator (ppl_Linear_Expression_t *ple, ppl_const_Generator_t g)
  Builds a linear expression corresponding to generator `g`*; writes a handle for the newly created linear expression at address* `ple`.
- int ppl_new_Linear_Expression_from_Congruence (ppl_Linear_Expression_t *ple, ppl_const_Congruence_t c)
  Builds a linear expression corresponding to congruence `c`*; writes a handle for the newly created linear expression at address* `ple`.
- int ppl_new_Linear_Expression_from_Grid_Generator (ppl_Linear_Expression_t *ple, ppl_const_Grid_Generator_t g)
  Builds a linear expression corresponding to grid generator `g`*; writes a handle for the newly created linear expression at address* `ple`.
- int ppl_assign_Linear_Expression_from_Linear_Expression (ppl_Linear_Expression_t dst, ppl_const_Linear_Expression_t src)
  Assigns a copy of the linear expression `src` *to* `dst`.
- int ppl_delete_Linear_Expression (ppl_const_Linear_Expression_t le)
  Invalidates the handle `le`*: this makes sure the corresponding resources will eventually be released.*

Functions that Do Not Modify the Linear Expression

- int ppl_Linear_Expression_space_dimension (ppl_const_Linear_Expression_t le, ppl_dimension_type *m)
  Writes to `m` *the space dimension of* `le`.
- int ppl_Linear_Expression_coefficient (ppl_const_Linear_Expression_t le, ppl_dimension_type var, ppl_Coefficient_t n)
  Copies into `n` *the coefficient of variable* `var` *in the linear expression* `le`.

- int ppl_Linear_Expression_inhomogeneous_term (ppl_const_Linear_Expression_t le, ppl_Coefficient-_t n)

  Copies into `n` *the inhomogeneous term of linear expression* `le`*.*
- int ppl_Linear_Expression_OK (ppl_const_Linear_Expression_t le)

  Returns a positive integer if `le` *is well formed, i.e., if it satisfies all its implementation invariants; returns 0 and perhaps makes some noise if* `le` *is broken. Useful for debugging purposes.*
- int ppl_Linear_Expression_is_zero (ppl_const_Linear_Expression_t le)

  Returns `true` *if and only if* `*this` *is* 0.
- int ppl_Linear_Expression_all_homogeneous_terms_are_zero (ppl_const_Linear_Expression_t le)

  Returns `true` *if and only if all the homogeneous terms of* `*this` *are* 0.

Functions that May Modify the Linear Expression

- int ppl_Linear_Expression_add_to_coefficient (ppl_Linear_Expression_t le, ppl_dimension_type var, ppl_const_Coefficient_t n)

  Adds `n` *to the coefficient of variable* `var` *in the linear expression* `le`*. The space dimension is set to be the maximum between* `var` *+ 1 and the old space dimension.*
- int ppl_Linear_Expression_add_to_inhomogeneous (ppl_Linear_Expression_t le, ppl_const_Coefficient-_t n)

  Adds `n` *to the inhomogeneous term of the linear expression* `le`*.*
- int ppl_add_Linear_Expression_to_Linear_Expression (ppl_Linear_Expression_t dst, ppl_const-_Linear_Expression_t src)

  Adds the linear expression `src` *to* `dst`*.*
- int ppl_subtract_Linear_Expression_from_Linear_Expression (ppl_Linear_Expression_t dst, ppl-_const_Linear_Expression_t src)

  Subtracts the linear expression `src` *from* `dst`*.*
- int ppl_multiply_Linear_Expression_by_Coefficient (ppl_Linear_Expression_t le, ppl_const_Coefficient-_t n)

  Multiply the linear expression `dst` *by* `n`*.*

Input/Output Functions

- int ppl_io_print_Linear_Expression (ppl_const_Linear_Expression_t x)

  Prints `x` *to* `stdout`*.*
- int ppl_io_fprint_Linear_Expression (FILE *stream, ppl_const_Linear_Expression_t x)

  Prints `x` *to the given output* `stream`*.*
- int ppl_io_asprint_Linear_Expression (char **strp, ppl_const_Linear_Expression_t x)

  Prints `x` *to a malloc-allocated string, a pointer to which is returned via* `strp`*.*
- int ppl_Linear_Expression_ascii_dump (ppl_const_Linear_Expression_t x, FILE *stream)

  Dumps an ascii representation of `x` *on* `stream`*.*
- int ppl_Linear_Expression_ascii_load (ppl_Linear_Expression_t x, FILE *stream)

  Loads an ascii representation of `x` *from* `stream`*.*

7.16.1 Detailed Description

Types and functions for linear expressions.

The types and functions for linear expression provide an interface towards *Linear_Expression*.

The documentation for this interface was generated from the following file:

- ppl_c_header.h

7.17 ppl_MIP_Problem_tag Interface Reference

Types and functions for MIP problems.

`#include <ppl_c_header.h>`

Related Functions

(Note that these are not member functions.)

Symbolic Constants

- int PPL_OPTIMIZATION_MODE_MAXIMIZATION
  Code of the "maximization" optimization mode.
- int PPL_OPTIMIZATION_MODE_MINIMIZATION
  Code of the "minimization" optimization mode.
- int PPL_MIP_PROBLEM_STATUS_UNFEASIBLE
  Code of the "unfeasible MIP problem" status.
- int PPL_MIP_PROBLEM_STATUS_UNBOUNDED
  Code of the "unbounded MIP problem" status.
- int PPL_MIP_PROBLEM_STATUS_OPTIMIZED
  Code of the "optimized MIP problem" status.
- int PPL_MIP_PROBLEM_CONTROL_PARAMETER_NAME_PRICING
  Code for the MIP problem's "pricing" control parameter name.
- int PPL_MIP_PROBLEM_CONTROL_PARAMETER_PRICING_TEXTBOOK
  Code of MIP problem's "textbook" pricing method.
- int PPL_MIP_PROBLEM_CONTROL_PARAMETER_PRICING_STEEPEST_EDGE_EXAC-T
  Code of MIP problem's "exact steepest-edge" pricing method.
- int PPL_MIP_PROBLEM_CONTROL_PARAMETER_PRICING_STEEPEST_EDGE_FLOAT
  Code of MIP problem's "float steepest-edge" pricing method.

Constructors, Assignment and Destructor

- int ppl_new_MIP_Problem_from_space_dimension (ppl_MIP_Problem_t ∗pmip, ppl_dimension-_type d)
  Builds a trivial MIP problem of dimension `d` and writes a handle to it at address `pmip`.
- int ppl_new_MIP_Problem (ppl_MIP_Problem_t ∗pmip, ppl_dimension_type d, ppl_const_Constraint-_System_t cs, ppl_const_Linear_Expression_t le, int m)
  Builds a MIP problem of space dimension `d` having feasible region `cs`, objective function `le` and optimization mode `m`; writes a handle to it at address `pmip`.
- int ppl_new_MIP_Problem_from_MIP_Problem (ppl_MIP_Problem_t ∗pmip, ppl_const_MIP_-Problem_t mip)
  Builds a MIP problem that is a copy of `mip`; writes a handle for the newly created system at address `pmip`.
- int ppl_assign_MIP_Problem_from_MIP_Problem (ppl_MIP_Problem_t dst, ppl_const_MIP_-Problem_t src)
  Assigns a copy of the MIP problem `src` to `dst`.
- int ppl_delete_MIP_Problem (ppl_const_MIP_Problem_t mip)
  Invalidates the handle `mip`: this makes sure the corresponding resources will eventually be released.

Functions that Do Not Modify the MIP_Problem

- int ppl_MIP_Problem_space_dimension (ppl_const_MIP_Problem_t mip, ppl_dimension_type *m)

  Writes to `m` *the dimension of the vector space enclosing* `mip`*.*
- int ppl_MIP_Problem_number_of_integer_space_dimensions (ppl_const_MIP_Problem_t mip, ppl-_dimension_type *m)

  Writes to `m` *the number of integer space dimensions of* `mip`*.*
- int ppl_MIP_Problem_integer_space_dimensions (ppl_const_MIP_Problem_t mip, ppl_dimension-_type ds[ ])

  Writes in the first positions of the array `ds` *all the integer space dimensions of problem* `mip`*. If the array is not big enough to hold all of the integer space dimensions, the behavior is undefined.*
- int ppl_MIP_Problem_number_of_constraints (ppl_const_MIP_Problem_t mip, ppl_dimension-_type *m)

  Writes to `m` *the number of constraints defining the feasible region of* `mip`*.*
- int ppl_MIP_Problem_constraint_at_index (ppl_const_MIP_Problem_t mip, ppl_dimension_type i, ppl_const_Constraint_t *pc)

  Writes at address `pc` *a const handle to the* `i-th` *constraint defining the feasible region of the MIP problem* `mip`*.*
- int ppl_MIP_Problem_objective_function (ppl_const_MIP_Problem_t mip, ppl_const_Linear_-Expression_t *ple)

  Writes a const handle to the linear expression defining the objective function of the MIP problem `mip` *at address* `ple`*.*
- int ppl_MIP_Problem_optimization_mode (ppl_const_MIP_Problem_t mip)

  Returns the optimization mode of the MIP problem `mip`*.*
- int ppl_MIP_Problem_OK (ppl_const_MIP_Problem_t mip)

  Returns a positive integer if `mip` *is well formed, i.e., if it satisfies all its implementation invariants; returns 0 and perhaps makes some noise if* `mip` *is broken. Useful for debugging purposes.*

Functions that May Modify the MIP_Problem

- int ppl_MIP_Problem_clear (ppl_MIP_Problem_t mip)

  Resets the MIP problem to be a trivial problem of space dimension 0.
- int ppl_MIP_Problem_add_space_dimensions_and_embed (ppl_MIP_Problem_t mip, ppl_dimension-_type d)

  Adds `d` *new dimensions to the space enclosing the MIP problem* `mip` *and to* `mip` *itself.*
- int ppl_MIP_Problem_add_to_integer_space_dimensions (ppl_MIP_Problem_t mip, ppl_dimension-_type ds[ ], size_t n)

  Sets the space dimensions that are specified in first `n` *positions of the array* `ds` *to be integer dimensions of problem* `mip`*. The presence of duplicates in* `ds` *is a waste but an innocuous one.*
- int ppl_MIP_Problem_add_constraint (ppl_MIP_Problem_t mip, ppl_const_Constraint_t c)

  Modifies the feasible region of the MIP problem `mip` *by adding a copy of the constraint* `c`*.*
- int ppl_MIP_Problem_add_constraints (ppl_MIP_Problem_t mip, ppl_const_Constraint_System-_t cs)

  Modifies the feasible region of the MIP problem `mip` *by adding a copy of the constraints in* `cs`*.*
- int ppl_MIP_Problem_set_objective_function (ppl_MIP_Problem_t mip, ppl_const_Linear_Expression-_t le)

  Sets the objective function of the MIP problem `mip` *to a copy of* `le`*.*
- int ppl_MIP_Problem_set_optimization_mode (ppl_MIP_Problem_t mip, int mode)

  Sets the optimization mode of the MIP problem `mip` *to* `mode`*.*

Computing the Solution of the MIP_Problem

- int ppl_MIP_Problem_is_satisfiable (ppl_const_MIP_Problem_t mip)

  Returns a positive integer if `mip` *is satisfiable; returns 0 otherwise.*

- int ppl_MIP_Problem_solve (ppl_const_MIP_Problem_t mip)
  Solves the MIP problem `mip`*, returning an exit status.*
- int ppl_MIP_Problem_evaluate_objective_function (ppl_const_MIP_Problem_t mip, ppl_const_Generator_t g, ppl_Coefficient_t num, ppl_Coefficient_t den)
  Evaluates the objective function of `mip` *on point* `g`*.*
- int ppl_MIP_Problem_feasible_point (ppl_const_MIP_Problem_t mip, ppl_const_Generator_t *pg)
  Writes a const handle to a feasible point for the MIP problem `mip` *at address* `pg`*.*
- int ppl_MIP_Problem_optimizing_point (ppl_const_MIP_Problem_t mip, ppl_const_Generator_t *pg)
  Writes a const handle to an optimizing point for the MIP problem `mip` *at address* `pg`*.*
- int ppl_MIP_Problem_optimal_value (ppl_const_MIP_Problem_t mip, ppl_Coefficient_t num, ppl_Coefficient_t den)
  Returns the optimal value for `mip`*.*

Querying/Setting Control Parameters

- int ppl_MIP_Problem_get_control_parameter (ppl_const_MIP_Problem_t mip, int name)
  Returns the value of control parameter `name` *in problem* `mip`*.*
- int ppl_MIP_Problem_set_control_parameter (ppl_MIP_Problem_t mip, int value)
  Sets control parameter `value` *in problem* `mip`*.*
- int ppl_MIP_Problem_total_memory_in_bytes (ppl_const_MIP_Problem_t mip, size_t *sz)
  Writes into `*sz` *the size in bytes of the memory occupied by* `mip`*.*
- int ppl_MIP_Problem_external_memory_in_bytes (ppl_const_MIP_Problem_t mip, size_t *sz)
  Writes into `*sz` *the size in bytes of the memory managed by* `mip`*.*

Input/Output Functions

- int ppl_io_print_MIP_Problem (ppl_const_MIP_Problem_t x)
  Prints `x` *to* `stdout`*.*
- int ppl_io_fprint_MIP_Problem (FILE *stream, ppl_const_MIP_Problem_t x)
  Prints `x` *to the given output* `stream`*.*
- int ppl_io_asprint_MIP_Problem (char **strp, ppl_const_MIP_Problem_t x)
  Prints `x` *to a malloc-allocated string, a pointer to which is returned via* `strp`*.*
- int ppl_MIP_Problem_ascii_dump (ppl_const_MIP_Problem_t x, FILE *stream)
  Dumps an ascii representation of `x` *on* `stream`*.*
- int ppl_MIP_Problem_ascii_load (ppl_MIP_Problem_t x, FILE *stream)
  Loads an ascii representation of `x` *from* `stream`*.*

7.17.1 Detailed Description

Types and functions for MIP problems.

The types and functions for MIP problems provide an interface towards *MIP_Problem*.

7.17.2 Friends And Related Function Documentation

7.17.2.1 int ppl_MIP_Problem_solve (ppl_const_MIP_Problem_t *mip*) [related]

Solves the MIP problem `mip`, returning an exit status.

Returns

`PPL_MIP_PROBLEM_STATUS_UNFEASIBLE` if the MIP problem is not satisfiable; `PPL_MIP_PROBLEM_STATUS_UNBOUNDED` if the MIP problem is satisfiable but there is no finite bound to the value of the objective function; `PPL_MIP_PROBLEM_STATUS_OPTIMIZED` if the MIP problem admits an optimal solution.

7.17.2.2 int ppl_MIP_Problem_evaluate_objective_function (ppl_const_MIP_Problem_t *mip*, ppl_const_Generator_t *g*, ppl_Coefficient_t *num*, ppl_Coefficient_t *den*) [related]

Evaluates the objective function of mip on point g.

Parameters

mip	The MIP problem defining the objective function;
g	The generator on which the objective function will be evaluated;
num	Will be assigned the numerator of the objective function value;
den	Will be assigned the denominator of the objective function value;

7.17.2.3 int ppl_MIP_Problem_optimal_value (ppl_const_MIP_Problem_t *mip*, ppl_Coefficient_t *num*, ppl_Coefficient_t *den*) [related]

Returns the optimal value for mip.

Parameters

mip	The MIP problem;
num	Will be assigned the numerator of the optimal value;
den	Will be assigned the denominator of the optimal value.

The documentation for this interface was generated from the following file:

- ppl_c_header.h

7.18 ppl_PIP_Decision_Node_tag Interface Reference

Types and functions for PIP decision nodes.

`#include <ppl_c_header.h>`

Related Functions

(Note that these are not member functions.)

- int ppl_PIP_Decision_Node_get_child_node (ppl_const_PIP_Decision_Node_t pip_dec, int b, ppl_const_PIP_Tree_Node_t *pip_tree)

  Writes to pip_tree a const pointer to either the true branch (if b is not zero) or the false branch (if b is zero) of pip_dec.

Input/Output Functions

- int ppl_io_print_PIP_Decision_Node (ppl_const_PIP_Decision_Node_t x)

  Prints x to stdout.
- int ppl_io_fprint_PIP_Decision_Node (FILE *stream, ppl_const_PIP_Decision_Node_t x)

  Prints x to the given output stream.
- int ppl_io_asprint_PIP_Decision_Node (char **strp, ppl_const_PIP_Decision_Node_t x)

  Prints x to a malloc-allocated string, a pointer to which is returned via strp.
- int ppl_PIP_Decision_Node_ascii_dump (ppl_const_PIP_Decision_Node_t x, FILE *stream)

  Dumps an ascii representation of x on stream.
- int ppl_PIP_Decision_Node_ascii_load (ppl_PIP_Decision_Node_t x, FILE *stream)

  Loads an ascii representation of x from stream.

7.18.1 Detailed Description

Types and functions for PIP decision nodes.

The types and functions for decision nodes provide an interface towards *PIP_Decision_Node*.

The documentation for this interface was generated from the following file:

- ppl_c_header.h

7.19 ppl_PIP_Problem_tag Interface Reference

Types and functions for PIP problems.

`#include <ppl_c_header.h>`

Related Functions

(Note that these are not member functions.)

Symbolic Constants

- int PPL_PIP_PROBLEM_STATUS_UNFEASIBLE
  Code of the "unfeasible PIP problem" status.
- int PPL_PIP_PROBLEM_STATUS_OPTIMIZED
  Code of the "optimized PIP problem" status.
- int PPL_PIP_PROBLEM_CONTROL_PARAMETER_NAME_CUTTING_STRATEGY
  Code for the PIP problem's "cutting strategy" control parameter name.
- int PPL_PIP_PROBLEM_CONTROL_PARAMETER_NAME_PIVOT_ROW_STRATEGY
  Code for the PIP problem's "pivot row strategy" control parameter name.
- int PPL_PIP_PROBLEM_CONTROL_PARAMETER_CUTTING_STRATEGY_FIRST
  Code of PIP problem's "first" cutting strategy.
- int PPL_PIP_PROBLEM_CONTROL_PARAMETER_CUTTING_STRATEGY_DEEPEST
  Code of PIP problem's "deepest" cutting strategy.
- int PPL_PIP_PROBLEM_CONTROL_PARAMETER_CUTTING_STRATEGY_ALL
  Code of PIP problem's "all" cutting strategy.
- int PPL_PIP_PROBLEM_CONTROL_PARAMETER_PIVOT_ROW_STRATEGY_FIRST
  Code of PIP problem's "first" pivot row strategy.
- int PPL_PIP_PROBLEM_CONTROL_PARAMETER_PIVOT_ROW_STRATEGY_MAX_CO-LUMN
  Code of PIP problem's "max column" pivot row strategy.

Constructors, Assignment and Destructor

- int ppl_new_PIP_Problem_from_space_dimension (ppl_PIP_Problem_t ∗ppip, ppl_dimension_-type d)
  Builds a trivial PIP problem of dimension `d` and writes a handle to it at address `ppip`.
- int ppl_new_PIP_Problem_from_PIP_Problem (ppl_PIP_Problem_t ∗ppip, ppl_const_PIP_Problem-_t pip)
  Builds a PIP problem that is a copy of `pip`; writes a handle for the newly created problem at address `ppip`.
- int ppl_assign_PIP_Problem_from_PIP_Problem (ppl_PIP_Problem_t dst, ppl_const_PIP_Problem-_t src)

Assigns a copy of the PIP problem `src` to `dst`.

- int ppl_new_PIP_Problem_from_constraints (ppl_PIP_Problem_t *ppip, ppl_dimension_type d, ppl_Constraint_System_const_iterator_t first, ppl_Constraint_System_const_iterator_t last, size_t n, ppl_dimension_type ds[ ])

  Builds a PIP problem having space dimension `d` from the sequence of constraints in the range [first, last); the `n` dimensions whose indices occur in `ds` are interpreted as parameters.
- int ppl_delete_PIP_Problem (ppl_const_PIP_Problem_t pip)

  Invalidates the handle `pip`: this makes sure the corresponding resources will eventually be released.

Functions that Do Not Modify the PIP_Problem

- int ppl_PIP_Problem_space_dimension (ppl_const_PIP_Problem_t pip, ppl_dimension_type *m)

  Writes to `m` the dimension of the vector space enclosing `pip`.
- int ppl_PIP_Problem_number_of_parameter_space_dimensions (ppl_const_PIP_Problem_t pip, ppl_dimension_type *m)

  Writes to `m` the number of parameter space dimensions of `pip`.
- int ppl_PIP_Problem_parameter_space_dimensions (ppl_const_PIP_Problem_t pip, ppl_dimension_type ds[ ])

  Writes in the first positions of the array `ds` all the parameter space dimensions of problem `pip`. If the array is not big enough to hold all of the parameter space dimensions, the behavior is undefined.
- int ppl_PIP_Problem_get_big_parameter_dimension (ppl_const_PIP_Problem_t pip, ppl_dimension_type *pd)

  Writes into `*pd` the big parameter dimension of PIP problem `pip`.
- int ppl_PIP_Problem_number_of_constraints (ppl_const_PIP_Problem_t pip, ppl_dimension_type *m)

  Writes to `m` the number of constraints defining the feasible region of `pip`.
- int ppl_PIP_Problem_constraint_at_index (ppl_const_PIP_Problem_t pip, ppl_dimension_type i, ppl_const_Constraint_t *pc)

  Writes at address `pc` a const handle to the `i-th` constraint defining the feasible region of the PIP problem `pip`.
- int ppl_PIP_Problem_total_memory_in_bytes (ppl_const_PIP_Problem_t pip, size_t *sz)

  Writes into `*sz` the size in bytes of the memory occupied by `pip`.
- int ppl_PIP_Problem_external_memory_in_bytes (ppl_const_PIP_Problem_t pip, size_t *sz)

  Writes into `*sz` the size in bytes of the memory managed by `pip`.
- int ppl_PIP_Problem_OK (ppl_const_PIP_Problem_t pip)

  Returns a positive integer if `pip` is well formed, i.e., if it satisfies all its implementation invariants; returns 0 and perhaps makes some noise if `pip` is broken. Useful for debugging purposes.

Functions that May Modify the PIP_Problem

- int ppl_PIP_Problem_clear (ppl_PIP_Problem_t pip)

  Resets the PIP problem to be a trivial problem of space dimension 0.
- int ppl_PIP_Problem_add_space_dimensions_and_embed (ppl_PIP_Problem_t pip, ppl_dimension_type pip_vars, ppl_dimension_type pip_params)

  Adds `pip_vars + pip_params` new space dimensions and embeds the PIP problem `pip` in the new vector space.
- int ppl_PIP_Problem_add_to_parameter_space_dimensions (ppl_PIP_Problem_t pip, ppl_dimension_type ds[ ], size_t n)

  Sets the space dimensions that are specified in first `n` positions of the array `ds` to be parameter dimensions of problem `pip`. The presence of duplicates in `ds` is a waste but an innocuous one.
- int ppl_PIP_Problem_set_big_parameter_dimension (ppl_PIP_Problem_t pip, ppl_dimension_type d)

Sets the big parameter dimension of PIP problem `pip` *to* `d`.

- int ppl_PIP_Problem_add_constraint (ppl_PIP_Problem_t pip, ppl_const_Constraint_t c)

  Modifies the feasible region of the PIP problem `pip` *by adding a copy of the constraint* `c`.
- int ppl_PIP_Problem_add_constraints (ppl_PIP_Problem_t pip, ppl_const_Constraint_System_t cs)

  Modifies the feasible region of the PIP problem `pip` *by adding a copy of the constraints in* `cs`.

Computing and Printing the Solution of the PIP_Problem

- int ppl_PIP_Problem_is_satisfiable (ppl_const_PIP_Problem_t pip)

  Returns a positive integer if `pip` *is satisfiable and an optimal solution can be found; returns 0 otherwise.*
- int ppl_PIP_Problem_solve (ppl_const_PIP_Problem_t pip)

  Solves the PIP problem `pip`*, returning an exit status.*
- int ppl_PIP_Problem_solution (ppl_const_PIP_Problem_t pip, ppl_const_PIP_Tree_Node_t *pip-_tree)

  Writes to `pip_tree` *a solution for* `pip`*, if it exists.*
- int ppl_PIP_Problem_optimizing_solution (ppl_const_PIP_Problem_t pip, ppl_const_PIP_Tree-_Node_t *pip_tree)

  Writes to `pip_tree` *an optimizing solution for* `pip`*, if it exists.*

Querying/Setting Control Parameters

- int ppl_PIP_Problem_get_control_parameter (ppl_const_PIP_Problem_t pip, int name)

  Returns the value of control parameter `name` *in problem* `pip`.
- int ppl_PIP_Problem_set_control_parameter (ppl_PIP_Problem_t pip, int value)

  Sets control parameter `value` *in problem* `pip`.

Input/Output Functions

- int ppl_io_print_PIP_Problem (ppl_const_PIP_Problem_t x)

  Prints `x` *to* `stdout`.
- int ppl_io_fprint_PIP_Problem (FILE *stream, ppl_const_PIP_Problem_t x)

  Prints `x` *to the given output* `stream`.
- int ppl_io_asprint_PIP_Problem (char **strp, ppl_const_PIP_Problem_t x)

  Prints `x` *to a malloc-allocated string, a pointer to which is returned via* `strp`.
- int ppl_PIP_Problem_ascii_dump (ppl_const_PIP_Problem_t x, FILE *stream)

  Dumps an ascii representation of `x` *on* `stream`.
- int ppl_PIP_Problem_ascii_load (ppl_PIP_Problem_t x, FILE *stream)

  Loads an ascii representation of `x` *from* `stream`.

7.19.1 Detailed Description

Types and functions for PIP problems.

The types and functions for PIP problems provide an interface towards *PIP_Problem*.

7.19.2 Friends And Related Function Documentation

7.19.2.1 int ppl_PIP_Problem_space_dimension (ppl_const_PIP_Problem_t *pip*, ppl_dimension_type * *m*) [related]

Writes to `m` the dimension of the vector space enclosing `pip`.

The vector space dimensions includes both the problem variables and the problem parameters, but they do not include the artificial parameters.

7.19.2.2 int ppl_PIP_Problem_add_space_dimensions_and_embed (ppl_PIP_Problem_t *pip*, ppl_dimension_type *pip_vars*, ppl_dimension_type *pip_params*) [related]

Adds `pip_vars + pip_params` new space dimensions and embeds the PIP problem `pip` in the new vector space.

Parameters

pip	The PIP problem to be embedded in the new vector space.
pip_vars	The number of space dimensions to add that are interpreted as PIP problem variables (i.e., non parameters). These are added *before* adding the `pip_params` parameters.
pip_params	The number of space dimensions to add that are interpreted as PIP problem parameters. These are added *after* having added the `pip_vars` problem variables.

The new space dimensions will be those having the highest indexes in the new PIP problem; they are initially unconstrained.

7.19.2.3 int ppl_PIP_Problem_solve (ppl_const_PIP_Problem_t *pip*) [related]

Solves the PIP problem `pip`, returning an exit status.

Returns

`PPL_PIP_PROBLEM_STATUS_UNFEASIBLE` if the PIP problem is not satisfiable; `PPL_PIP_PROBLEM_STATUS_OPTIMIZED` if the PIP problem admits an optimal solution.

The documentation for this interface was generated from the following file:

- ppl_c_header.h

7.20 ppl_PIP_Solution_Node_tag Interface Reference

Types and functions for PIP solution nodes.

```
#include <ppl_c_header.h>
```

Related Functions

(Note that these are not member functions.)

- int ppl_PIP_Solution_Node_get_parametric_values (ppl_const_PIP_Solution_Node_t pip_sol, ppl_dimension_type var, ppl_const_Linear_Expression_t *le)

  Writes to `le` *a const pointer to the parametric expression of the values of variable* `var` *in solution node* `pip_sol`*.*

Input/Output Functions

- int ppl_io_print_PIP_Solution_Node (ppl_const_PIP_Solution_Node_t x)

  Prints `x` *to* `stdout`*.*
- int ppl_io_fprint_PIP_Solution_Node (FILE *stream, ppl_const_PIP_Solution_Node_t x)

  Prints `x` *to the given output* `stream`*.*
- int ppl_io_asprint_PIP_Solution_Node (char **strp, ppl_const_PIP_Solution_Node_t x)

Prints `x` *to a malloc-allocated string, a pointer to which is returned via* `strp`*.*

- int ppl_PIP_Solution_Node_ascii_dump (ppl_const_PIP_Solution_Node_t x, FILE *stream)

  Dumps an ascii representation of `x` *on* `stream`*.*
- int ppl_PIP_Solution_Node_ascii_load (ppl_PIP_Solution_Node_t x, FILE *stream)

  Loads an ascii representation of `x` *from* `stream`*.*

7.20.1 Detailed Description

Types and functions for PIP solution nodes.

The types and functions for solution nodes provide an interface towards *PIP_Solution_Node*.

7.20.2 Friends And Related Function Documentation

7.20.2.1 int ppl_PIP_Solution_Node_get_parametric_values (ppl_const_PIP_Solution_Node_t *pip_sol*, ppl_dimension_type *var*, ppl_const_Linear_Expression_t * *le*) `[related]`

Writes to `le` a const pointer to the parametric expression of the values of variable `var` in solution node `pip_sol`.

The linear expression assigned to `le` will only refer to (problem or artificial) parameters.

Parameters

pip_sol	The solution tree node.
var	The variable which is queried about.
le	The returned expression for variable `var`.

Returns

PPL_ERROR_INVALID_ARGUMENT Returned if `var` is dimension-incompatible with `*this` or if `var` is a problem parameter.

The documentation for this interface was generated from the following file:

- ppl_c_header.h

7.21 ppl_PIP_Tree_Node_tag Interface Reference

Types and functions for generic PIP tree nodes.

`#include <ppl_c_header.h>`

Related Functions

(Note that these are not member functions.)

- int ppl_PIP_Tree_Node_as_solution (ppl_const_PIP_Tree_Node_t spip_tree, ppl_const_PIP_Solution_Node_t *dpip_tree)

  Writes to `dpip_tree` *the solution node if* `spip_tree` *is a solution node, and 0 otherwise.*
- int ppl_PIP_Tree_Node_as_decision (ppl_const_PIP_Tree_Node_t spip_tree, ppl_const_PIP_Decision_Node_t *dpip_tree)

Writes to `dpip_tree` the decision node if `spip_tree` is a decision node, and 0 otherwise.

- int ppl_PIP_Tree_Node_get_constraints (ppl_const_PIP_Tree_Node_t pip_tree, ppl_const_Constraint-_System_t ∗pcs)

  Writes to `pcs` the local system of parameter constraints at the pip tree node `pip_tree`.
- int ppl_PIP_Tree_Node_OK (ppl_const_PIP_Tree_Node_t pip)

  Returns a positive integer if `pip_tree` is well formed, i.e., if it satisfies all its implementation invariants; returns 0 and perhaps makes some noise if `pip_tree` is broken. Useful for debugging purposes.
- int ppl_PIP_Tree_Node_number_of_artificials (ppl_const_PIP_Tree_Node_t pip_tree, ppl_dimension-_type ∗m)

  Writes to `m` the number of elements in the artificial parameter sequence in the pip tree node `pip_tree`.
- int ppl_PIP_Tree_Node_begin (ppl_const_PIP_Tree_Node_t pip_tree, ppl_Artificial_Parameter_Sequence-_const_iterator_t pit)

  Assigns to `pit` a const iterator "pointing" to the beginning of the artificial parameter sequence in the pip tree node `pip_tree`.
- int ppl_PIP_Tree_Node_end (ppl_const_PIP_Tree_Node_t pip_tree, ppl_Artificial_Parameter_Sequence-_const_iterator_t pit)

  Assigns to `pit` a const iterator "pointing" to the end of the artificial parameter sequence in the pip tree node `pip_tree`.

Input/Output Functions

- int ppl_io_print_PIP_Tree_Node (ppl_const_PIP_Tree_Node_t x)

  Prints `x` to `stdout`.
- int ppl_io_fprint_PIP_Tree_Node (FILE ∗stream, ppl_const_PIP_Tree_Node_t x)

  Prints `x` to the given output `stream`.
- int ppl_io_asprint_PIP_Tree_Node (char ∗∗strp, ppl_const_PIP_Tree_Node_t x)

  Prints `x` to a malloc-allocated string, a pointer to which is returned via `strp`.
- int ppl_PIP_Tree_Node_ascii_dump (ppl_const_PIP_Tree_Node_t x, FILE ∗stream)

  Dumps an ascii representation of `x` on `stream`.
- int ppl_PIP_Tree_Node_ascii_load (ppl_PIP_Tree_Node_t x, FILE ∗stream)

  Loads an ascii representation of `x` from `stream`.

7.21.1 Detailed Description

Types and functions for generic PIP tree nodes.

The types and functions for tree nodes provide an interface towards *PIP_Tree_Node*.

The documentation for this interface was generated from the following file:

- ppl_c_header.h

7.22 ppl_Pointset_Powerset_C_Polyhedron_const_iterator_tag Interface Reference

Types and functions for iterating on the disjuncts of a const ppl_Pointset_Powerset_C_Polyhedron_tag.

Related Functions

(Note that these are not member functions.)

Construction, Initialization and Destruction

- int ppl_new_Pointset_Powerset_C_Polyhedron_const_iterator (ppl_Pointset_Powerset_C_Polyhedron_const_iterator_t *pit)

  Builds a new 'const iterator' and writes a handle to it at address `pit`.
- int ppl_new_Pointset_Powerset_C_Polyhedron_const_iterator_from_const_iterator (ppl_Pointset_Powerset_C_Polyhedron_const_iterator_t *pit, ppl_const_Pointset_Powerset_C_Polyhedron_const_iterator_t y)

  Builds a copy of `y` *and writes a handle to it at address* `pit`.
- int ppl_Pointset_Powerset_C_Polyhedron_const_iterator_begin (ppl_const_Pointset_Powerset_C_Polyhedron_t ps, ppl_Pointset_Powerset_C_Polyhedron_const_iterator_t psit)

  Assigns to `psit` *a const iterator "pointing" to the beginning of the sequence of disjuncts of* `ps`.
- int ppl_Pointset_Powerset_C_Polyhedron_const_iterator_end (ppl_const_Pointset_Powerset_C_Polyhedron_t ps, ppl_Pointset_Powerset_C_Polyhedron_const_iterator_t psit)

  Assigns to `psit` *a const iterator "pointing" past the end of the sequence of disjuncts of* `ps`.
- int ppl_delete_Pointset_Powerset_C_Polyhedron_const_iterator (ppl_const_Pointset_Powerset_C_Polyhedron_const_iterator_t it)

  Invalidates the handle `it` *: this makes sure the corresponding resources will eventually be released.*

Dereferencing, Increment, Decrement and Equality Testing

- int ppl_Pointset_Powerset_C_Polyhedron_const_iterator_dereference (ppl_const_Pointset_Powerset_C_Polyhedron_const_iterator_t it, ppl_const_Polyhedron_t *d)

  Dereferences `it` *writing a const handle to the resulting disjunct at address* `d`*.*
- int ppl_Pointset_Powerset_C_Polyhedron_const_iterator_increment (ppl_Pointset_Powerset_C_Polyhedron_const_iterator_t it)

  Increments `it` *so that it "points" to the next disjunct.*
- int ppl_Pointset_Powerset_C_Polyhedron_const_iterator_decrement (ppl_Pointset_Powerset_C_Polyhedron_const_iterator_t it)

  Decrements `it` *so that it "points" to the previous disjunct.*
- int ppl_Pointset_Powerset_C_Polyhedron_const_iterator_equal_test (ppl_const_Pointset_Powerset_C_Polyhedron_const_iterator_t x, ppl_const_Pointset_Powerset_C_Polyhedron_const_iterator_t y)

  Returns a positive integer if the iterators corresponding to `x` *and* `y` *are equal; returns 0 if they are different.*

7.22.1 Detailed Description

Types and functions for iterating on the disjuncts of a const ppl_Pointset_Powerset_C_Polyhedron_tag.

7.22.2 Friends And Related Function Documentation

7.22.2.1 int ppl_Pointset_Powerset_C_Polyhedron_const_iterator_dereference (ppl_const_Pointset_Powerset_C_Polyhedron_const_iterator_t *it*, ppl_const_Polyhedron_t * *d*) [related]

Dereferences `it` writing a const handle to the resulting disjunct at address `d`.

Warning

On exit, the disjunct `d` is still owned by the powerset object: any function call on the owning powerset object may invalidate it. Moreover, `d` should **not** be deleted directly: its resources will be released when deleting the owning powerset.

The documentation for this interface was generated from the following file:

- C_interface.dox

7.23 ppl_Pointset_Powerset_C_Polyhedron_iterator_tag Interface Reference

Types and functions for iterating on the disjuncts of a ppl_Pointset_Powerset_C_Polyhedron_tag.

Related Functions

(Note that these are not member functions.)

Construction, Initialization and Destruction

- int ppl_new_Pointset_Powerset_C_Polyhedron_iterator (ppl_Pointset_Powerset_C_Polyhedron-_iterator_t *pit)
  Builds a new 'iterator' and writes a handle to it at address `pit`.
- int ppl_new_Pointset_Powerset_C_Polyhedron_iterator_from_iterator (ppl_Pointset_Powerset_-C_Polyhedron_iterator_t *pit, ppl_const_Pointset_Powerset_C_Polyhedron_iterator_t y)
  Builds a copy of `y` *and writes a handle to it at address* `pit`.
- int ppl_Pointset_Powerset_C_Polyhedron_iterator_begin (ppl_Pointset_Powerset_C_Polyhedron-_t ps, ppl_Pointset_Powerset_C_Polyhedron_iterator_t psit)
  Assigns to `psit` *an iterator "pointing" to the beginning of the sequence of disjuncts of* `ps`.
- int ppl_Pointset_Powerset_C_Polyhedron_iterator_end (ppl_Pointset_Powerset_C_Polyhedron_t ps, ppl_Pointset_Powerset_C_Polyhedron_iterator_t psit)
  Assigns to `psit` *an iterator "pointing" past the end of the sequence of disjuncts of* `ps`.
- int ppl_delete_Pointset_Powerset_C_Polyhedron_iterator (ppl_const_Pointset_Powerset_C_Polyhedron-_iterator_t it)
  Invalidates the handle `it`*: this makes sure the corresponding resources will eventually be released.*

Dereferencing, Increment, Decrement and Equality Testing

- int ppl_Pointset_Powerset_C_Polyhedron_iterator_dereference (ppl_const_Pointset_Powerset_C-_Polyhedron_iterator_t it, ppl_const_Polyhedron_t *d)
  Dereferences `it` *writing a const handle to the resulting disjunct at address* `d`.
- int ppl_Pointset_Powerset_C_Polyhedron_iterator_increment (ppl_Pointset_Powerset_C_Polyhedron-_iterator_t it)
  Increments `it` *so that it "points" to the next disjunct.*
- int ppl_Pointset_Powerset_C_Polyhedron_iterator_decrement (ppl_Pointset_Powerset_C_Polyhedron-_iterator_t it)
  Decrements `it` *so that it "points" to the previous disjunct.*
- int ppl_Pointset_Powerset_C_Polyhedron_iterator_equal_test (ppl_const_Pointset_Powerset_C-_Polyhedron_iterator_t x, ppl_const_Pointset_Powerset_C_Polyhedron_iterator_t y)
  Returns a positive integer if the iterators corresponding to `x` *and* `y` *are equal; returns 0 if they are different.*

7.23.1 Detailed Description

Types and functions for iterating on the disjuncts of a ppl_Pointset_Powerset_C_Polyhedron_tag.

7.23.2 Friends And Related Function Documentation

7.23.2.1 int ppl_Pointset_Powerset_C_Polyhedron_iterator_dereference (ppl_const_-Pointset_Powerset_C_Polyhedron_iterator_t *it*, ppl_const_Polyhedron_t * *d*) `[related]`

Dereferences `it` writing a const handle to the resulting disjunct at address `d`.

Note

Even though `it` is an non-const iterator, dereferencing it results in a handle to a **const** disjunct. This is because mutable iterators are meant to allow for the modification of the sequence of disjuncts (e.g., by dropping elements), while preventing direct modifications of the disjuncts they point to.

Warning

On exit, the disjunct `d` is still owned by the powerset object: any function call on the owning powerset object may invalidate it. Moreover, `d` should **not** be deleted directly: its resources will be released when deleting the owning powerset.

The documentation for this interface was generated from the following file:

- C_interface.dox

7.24 ppl_Pointset_Powerset_C_Polyhedron_tag Interface Reference

Types and functions for the Pointset_Powerset of C_Polyhedron objects.

Related Functions

(Note that these are not member functions.)

Ad Hoc Functions for Pointset_Powerset domains

- int ppl_Pointset_Powerset_C_Polyhedron_omega_reduce (ppl_const_Pointset_Powerset_C_Polyhedron_t ps)
  Drops from the sequence of disjuncts in `ps` *all the non-maximal elements so that* `ps` *is non-redundant.*
- int ppl_Pointset_Powerset_C_Polyhedron_size (ppl_const_Pointset_Powerset_C_Polyhedron_t ps, size_t *sz)
  Writes to `sz` *the number of disjuncts in* `ps`*.*
- int ppl_Pointset_Powerset_C_Polyhedron_geometrically_covers_Pointset_Powerset_C_Polyhedron (ppl_const_Pointset_Powerset_C_Polyhedron_t x, ppl_const_Pointset_Powerset_C_Polyhedron_t y)
  Returns a positive integer if powerset `x` *geometrically covers powerset* `y`*; returns 0 otherwise.*
- int ppl_Pointset_Powerset_C_Polyhedron_geometrically_equals_Pointset_Powerset_C_Polyhedron (ppl_const_Pointset_Powerset_C_Polyhedron_t x, ppl_const_Pointset_Powerset_C_Polyhedron_t y)
  Returns a positive integer if powerset `x` *is geometrically equal to powerset* `y`*; returns 0 otherwise.*
- int ppl_Pointset_Powerset_C_Polyhedron_add_disjunct (ppl_Pointset_Powerset_C_Polyhedron_t ps, ppl_const_Polyhedron_t d)
  Adds to `ps` *a copy of disjunct* `d`*.*
- int ppl_Pointset_Powerset_C_Polyhedron_drop_disjunct (ppl_Pointset_Powerset_C_Polyhedron_t ps, ppl_const_Pointset_Powerset_C_Polyhedron_iterator_t cit, ppl_Pointset_Powerset_C_Polyhedron_iterator_t it)
  Drops from `ps` *the disjunct pointed to by* `cit`*, assigning to* `it` *an iterator to the disjunct following* `cit`*.*
- int ppl_Pointset_Powerset_C_Polyhedron_drop_disjuncts (ppl_Pointset_Powerset_C_Polyhedron_t ps, ppl_const_Pointset_Powerset_C_Polyhedron_iterator_t first, ppl_const_Pointset_Powerset_C_Polyhedron_iterator_t last)
  Drops from `ps` *all the disjuncts from* `first` *to* `last` *(excluded).*
- int ppl_Pointset_Powerset_C_Polyhedron_pairwise_reduce (ppl_Pointset_Powerset_C_Polyhedron_t ps)
  Modifies `ps` *by (recursively) merging together the pairs of disjuncts whose upper-bound is the same as their set-theoretical union.*

7.24.1 Detailed Description

Types and functions for the Pointset_Powerset of C_Polyhedron objects.

The powerset domains can be instantiated by taking as a base domain any fixed semantic geometric description (C and NNC polyhedra, BD and octagonal shapes, boxes and grids). An element of the powerset domain represents a disjunctive collection of base objects (its disjuncts), all having the same space dimension.

Besides the functions that are available in all semantic geometric descriptions (whose documentation is not repeated here), the powerset domain also provides several ad hoc functions. In particular, the iterator types allow for the examination and manipulation of the collection of disjuncts.

7.24.2 Friends And Related Function Documentation

7.24.2.1 int ppl_Pointset_Powerset_C_Polyhedron_size (ppl_const_Pointset_Powerset_C_Polyhedron_t *ps*, size_t ∗ *sz*) [related]

Writes to sz the number of disjuncts in ps.

Note

If present, Omega-redundant elements will be counted too.

The documentation for this interface was generated from the following file:

- C_interface.dox

7.25 ppl_Polyhedron_tag Interface Reference

Types and functions for the domains of C and NNC convex polyhedra.

Related Functions

(Note that these are not member functions.)

Constructors and Assignment for C_Polyhedron

- int ppl_new_C_Polyhedron_from_space_dimension (ppl_Polyhedron_t ∗pph, ppl_dimension_type d, int empty)

  Builds a C polyhedron of dimension d *and writes an handle to it at address* pph*. If* empty *is different from zero, the newly created polyhedron will be empty; otherwise, it will be a universe polyhedron.*
- int ppl_new_C_Polyhedron_from_C_Polyhedron (ppl_Polyhedron_t ∗pph, ppl_const_Polyhedron_t ph)

  Builds a C polyhedron that is a copy of ph*; writes a handle for the newly created polyhedron at address* pph*.*
- int ppl_new_C_Polyhedron_from_C_Polyhedron_with_complexity (ppl_Polyhedron_t ∗pph, ppl_const_Polyhedron_t ph, int complexity)

  Builds a C polyhedron that is a copy of ph*; writes a handle for the newly created polyhedron at address* pph*.*
- int ppl_new_C_Polyhedron_from_Constraint_System (ppl_Polyhedron_t ∗pph, ppl_const_Constraint_System_t cs)

Builds a new C polyhedron from the system of constraints `cs` *and writes a handle for the newly created polyhedron at address* `pph`.

- int ppl_new_C_Polyhedron_recycle_Constraint_System (ppl_Polyhedron_t *pph, ppl_Constraint_System_t cs)

  Builds a new C polyhedron recycling the system of constraints `cs` *and writes a handle for the newly created polyhedron at address* `pph`.
- int ppl_new_C_Polyhedron_from_Congruence_System (ppl_Polyhedron_t *pph, ppl_const_Congruence_System_t cs)

  Builds a new C polyhedron from the system of congruences `cs` *and writes a handle for the newly created polyhedron at address* `pph`.
- int ppl_new_C_Polyhedron_recycle_Congruence_System (ppl_Polyhedron_t *pph, ppl_Congruence_System_t cs)

  Builds a new C polyhedron recycling the system of congruences `cs` *and writes a handle for the newly created polyhedron at address* `pph`.
- int ppl_assign_C_Polyhedron_from_C_Polyhedron (ppl_Polyhedron_t dst, ppl_const_Polyhedron_t src)

  Assigns a copy of the C polyhedron `src` *to the C polyhedron* `dst`.

Constructors and Assignment for NNC_Polyhedron

- int ppl_new_NNC_Polyhedron_from_space_dimension (ppl_Polyhedron_t *pph, ppl_dimension_type d, int empty)

  Builds an NNC polyhedron of dimension `d` *and writes an handle to it at address* `pph`. *If* `empty` *is different from zero, the newly created polyhedron will be empty; otherwise, it will be a universe polyhedron.*
- int ppl_new_NNC_Polyhedron_from_NNC_Polyhedron (ppl_Polyhedron_t *pph, ppl_const_Polyhedron_t ph)

  Builds an NNC polyhedron that is a copy of `ph`*; writes a handle for the newly created polyhedron at address* `pph`.
- int ppl_new_NNC_Polyhedron_from_NNC_Polyhedron_with_complexity (ppl_Polyhedron_t *pph, ppl_const_Polyhedron_t ph, int complexity)

  Builds an NNC polyhedron that is a copy of `ph`*; writes a handle for the newly created polyhedron at address* `pph`.
- int ppl_new_NNC_Polyhedron_from_Constraint_System (ppl_Polyhedron_t *pph, ppl_const_Constraint_System_t cs)

  Builds a new NNC polyhedron from the system of constraints `cs` *and writes a handle for the newly created polyhedron at address* `pph`.
- int ppl_new_NNC_Polyhedron_recycle_Constraint_System (ppl_Polyhedron_t *pph, ppl_Constraint_System_t cs)

  Builds a new NNC polyhedron recycling the system of constraints `cs` *and writes a handle for the newly created polyhedron at address* `pph`.
- int ppl_new_NNC_Polyhedron_from_Congruence_System (ppl_Polyhedron_t *pph, ppl_const_Congruence_System_t cs)

  Builds a new NNC polyhedron from the system of congruences `cs` *and writes a handle for the newly created polyhedron at address* `pph`.
- int ppl_new_NNC_Polyhedron_recycle_Congruence_System (ppl_Polyhedron_t *pph, ppl_Congruence_System_t cs)

  Builds a new NNC polyhedron recycling the system of congruences `cs` *and writes a handle for the newly created polyhedron at address* `pph`.
- int ppl_assign_NNC_Polyhedron_from_NNC_Polyhedron (ppl_Polyhedron_t dst, ppl_const_Polyhedron_t src)

  Assigns a copy of the NNC polyhedron `src` *to the NNC polyhedron* `dst`.

Constructors Behaving as Conversion Operators

*Besides the conversions listed here below, the library also provides conversion operators that build a semantic geometric description starting from **any** other semantic geometric description (e.g., ppl_new_Grid_from_C_Polyhedron, ppl_new_C_Polyhedron_from_BD_Shape_mpq_class, etc.). Clearly, the conversion operators are only available if both the source and the target semantic geometric descriptions have been enabled when configuring the library. The conversions also taking as argument a complexity class sometimes provide non-trivial precision/efficiency trade-offs.*

- int ppl_new_C_Polyhedron_from_NNC_Polyhedron (ppl_Polyhedron_t ∗pph, ppl_const_Polyhedron_t ph)
  Builds a C polyhedron that is a copy of the topological closure of the NNC polyhedron `ph`*; writes a handle for the newly created polyhedron at address* `pph`*.*
- int ppl_new_C_Polyhedron_from_NNC_Polyhedron_with_complexity (ppl_Polyhedron_t ∗pph, ppl_const_Polyhedron_t ph, int complexity)
  Builds a C polyhedron that approximates NNC_Polyhedron `ph`*, using an algorithm whose complexity does not exceed* `complexity`*; writes a handle for the newly created polyhedron at address* `pph`*.*
- int ppl_new_NNC_Polyhedron_from_C_Polyhedron (ppl_Polyhedron_t ∗pph, ppl_const_Polyhedron_t ph)
  Builds an NNC polyhedron that is a copy of the C polyhedron `ph`*; writes a handle for the newly created polyhedron at address* `pph`*.*
- int ppl_new_NNC_Polyhedron_from_C_Polyhedron_with_complexity (ppl_Polyhedron_t ∗pph, ppl_const_Polyhedron_t ph, int complexity)
  Builds an NNC polyhedron that approximates C_Polyhedron `ph`*, using an algorithm whose complexity does not exceed* `complexity`*; writes a handle for the newly created polyhedron at address* `pph`*.*

Destructor for (C or NNC) Polyhedra

- int ppl_delete_Polyhedron (ppl_const_Polyhedron_t ph)
  Invalidates the handle `ph`*: this makes sure the corresponding resources will eventually be released.*

Functions that Do Not Modify the Polyhedron

- int ppl_Polyhedron_space_dimension (ppl_const_Polyhedron_t ph, ppl_dimension_type ∗m)
  Writes to `m` *the dimension of the vector space enclosing* `ph`*.*
- int ppl_Polyhedron_affine_dimension (ppl_const_Polyhedron_t ph, ppl_dimension_type ∗m)
  Writes to `m` *the affine dimension of* `ph` *(not to be confused with the dimension of its enclosing vector space) or 0, if* `ph` *is empty.*
- int ppl_Polyhedron_relation_with_Constraint (ppl_const_Polyhedron_t ph, ppl_const_Constraint_t c)
  Checks the relation between the polyhedron `ph` *and the constraint* `c`*.*
- int ppl_Polyhedron_relation_with_Generator (ppl_const_Polyhedron_t ph, ppl_const_Generator_t g)
  Checks the relation between the polyhedron `ph` *and the generator* `g`*.*
- int ppl_Polyhedron_get_constraints (ppl_const_Polyhedron_t ph, ppl_const_Constraint_System_t ∗pcs)
  Writes a const handle to the constraint system defining the polyhedron `ph` *at address* `pcs`*.*
- int ppl_Polyhedron_get_congruences (ppl_const_Polyhedron_t ph, ppl_const_Congruence_System_t ∗pcs)
  Writes at address `pcs` *a const handle to a system of congruences approximating the polyhedron* `ph`*.*
- int ppl_Polyhedron_get_minimized_constraints (ppl_const_Polyhedron_t ph, ppl_const_Constraint_System_t ∗pcs)
  Writes a const handle to the minimized constraint system defining the polyhedron `ph` *at address* `pcs`*.*
- int ppl_Polyhedron_get_minimized_congruences (ppl_const_Polyhedron_t ph, ppl_const_Congruence_System_t ∗pcs)

Writes at address `pcs` *a const handle to a system of minimized congruences approximating the polyhedron* `ph`*.*

- int ppl_Polyhedron_is_empty (ppl_const_Polyhedron_t ph)

  Returns a positive integer if `ph` *is empty; returns 0 if* `ph` *is not empty.*
- int ppl_Polyhedron_is_universe (ppl_const_Polyhedron_t ph)

  Returns a positive integer if `ph` *is a universe polyhedron; returns 0 if it is not.*
- int ppl_Polyhedron_is_bounded (ppl_const_Polyhedron_t ph)

  Returns a positive integer if `ph` *is bounded; returns 0 if* `ph` *is unbounded.*
- int ppl_Polyhedron_contains_integer_point (ppl_const_Polyhedron_t ph)

  Returns a positive integer if `ph` *contains at least one integer point; returns 0 otherwise.*
- int ppl_Polyhedron_is_topologically_closed (ppl_const_Polyhedron_t ph)

  Returns a positive integer if `ph` *is topologically closed; returns 0 if* `ph` *is not topologically closed.*
- int ppl_Polyhedron_is_discrete (ppl_const_Polyhedron_t ph)

  Returns a positive integer if `ph` *is a discrete set; returns 0 if* `ph` *is not a discrete set.*
- int ppl_Polyhedron_constrains (ppl_Polyhedron_t ph, ppl_dimension_type var)

  Returns a positive integer if `ph` *constrains* `var`*; returns 0 if* `ph` *does not constrain* `var`*.*
- int ppl_Polyhedron_bounds_from_above (ppl_const_Polyhedron_t ph, ppl_const_Linear_Expression_t le)

  Returns a positive integer if `le` *is bounded from above in* `ph`*; returns 0 otherwise.*
- int ppl_Polyhedron_bounds_from_below (ppl_const_Polyhedron_t ph, ppl_const_Linear_Expression_t le)

  Returns a positive integer if `le` *is bounded from below in* `ph`*; returns 0 otherwise.*
- int ppl_Polyhedron_maximize_with_point (ppl_const_Polyhedron_t ph, ppl_const_Linear_Expression_t le, ppl_Coefficient_t sup_n, ppl_Coefficient_t sup_d, int *pmaximum, ppl_Generator_t point)

  Returns a positive integer if `ph` *is not empty and* `le` *is bounded from above in* `ph`*, in which case the supremum value and a point where* `le` *reaches it are computed.*
- int ppl_Polyhedron_maximize (ppl_const_Polyhedron_t ph, ppl_const_Linear_Expression_t le, ppl_Coefficient_t sup_n, ppl_Coefficient_t sup_d, int *pmaximum)

  The same as ppl_Polyhedron_maximize_with_point, but without the output argument for the location where the supremum value is reached.
- int ppl_Polyhedron_minimize_with_point (ppl_const_Polyhedron_t ph, ppl_const_Linear_Expression_t le, ppl_Coefficient_t inf_n, ppl_Coefficient_t inf_d, int *pminimum, ppl_Generator_t point)

  Returns a positive integer if `ph` *is not empty and* `le` *is bounded from below in* `ph`*, in which case the infimum value and a point where* `le` *reaches it are computed.*
- int ppl_Polyhedron_minimize_with_point (ppl_const_Polyhedron_t ph, ppl_const_Linear_Expression_t le, ppl_Coefficient_t inf_n, ppl_Coefficient_t inf_d, int *pminimum)

  The same as ppl_Polyhedron_minimize_with_point, but without the output argument for the location where the infimum value is reached.
- int ppl_Polyhedron_contains_Polyhedron (ppl_const_Polyhedron_t x, ppl_const_Polyhedron_t y)

  Returns a positive integer if `x` *contains or is equal to* `y`*; returns 0 if it does not.*
- int ppl_Polyhedron_strictly_contains_Polyhedron (ppl_const_Polyhedron_t x, ppl_const_Polyhedron_t y)

  Returns a positive integer if `x` *strictly contains* `y`*; returns 0 if it does not.*
- int ppl_Polyhedron_is_disjoint_from_Polyhedron (ppl_const_Polyhedron_t x, ppl_const_Polyhedron_t y)

  Returns a positive integer if `x` *and* `y` *are disjoint; returns 0 if they are not.*
- int ppl_Polyhedron_equals_Polyhedron (ppl_const_Polyhedron_t x, ppl_const_Polyhedron_t y)

  Returns a positive integer if `x` *and* `y` *are the same polyhedron; returns 0 if they are different.*
- int ppl_Polyhedron_OK (ppl_const_Polyhedron_t ph)

  Returns a positive integer if `ph` *is well formed, i.e., if it satisfies all its implementation invariants; returns 0 and perhaps makes some noise if* `ph` *is broken. Useful for debugging purposes.*
- int ppl_Polyhedron_external_memory_in_bytes (ppl_const_Polyhedron_t ph, size_t *sz)

Writes to `sz` *a lower bound to the size in bytes of the memory managed by* `ph`*.*

- int ppl_Polyhedron_total_memory_in_bytes (ppl_const_Polyhedron_t ph, size_t *sz)

  Writes to `sz` *a lower bound to the size in bytes of the memory managed by* `ph`*.*

Space Dimension Preserving Functions that May Modify the Polyhedron

- int ppl_Polyhedron_add_constraint (ppl_Polyhedron_t ph, ppl_const_Constraint_t c)

  Adds a copy of the constraint `c` *to the system of constraints of* `ph`*.*
- int ppl_Polyhedron_add_congruence (ppl_Polyhedron_t ph, ppl_const_Congruence_t c)

  Adds a copy of the congruence `c` *to polyhedron of* `ph`*.*
- int ppl_Polyhedron_add_constraints (ppl_Polyhedron_t ph, ppl_const_Constraint_System_t cs)

  Adds a copy of the system of constraints `cs` *to the system of constraints of* `ph`*.*
- int ppl_Polyhedron_add_congruences (ppl_Polyhedron_t ph, ppl_const_Congruence_System_t cs)

  Adds a copy of the system of congruences `cs` *to the polyhedron* `ph`*.*
- int ppl_Polyhedron_add_recycled_constraints (ppl_Polyhedron_t ph, ppl_Constraint_System_t cs)

  Adds the system of constraints `cs` *to the system of constraints of* `ph`*.*
- int ppl_Polyhedron_add_recycled_congruences (ppl_Polyhedron_t ph, ppl_Congruence_System_t cs)

  Adds the system of congruences `cs` *to the polyhedron* `ph`*.*
- int ppl_Polyhedron_refine_with_constraint (ppl_Polyhedron_t ph, ppl_const_Constraint_t c)

  Refines `ph` *using constraint* `c`*.*
- int ppl_Polyhedron_refine_with_congruence (ppl_Polyhedron_t ph, ppl_const_Congruence_t c)

  Refines `ph` *using congruence* `c`*.*
- int ppl_Polyhedron_refine_with_constraints (ppl_Polyhedron_t ph, ppl_const_Constraint_System_t cs)

  Refines `ph` *using the constraints in* `cs`*.*
- int ppl_Polyhedron_refine_with_congruences (ppl_Polyhedron_t ph, ppl_const_Congruence_System_t cs)

  Refines `ph` *using the congruences in* `cs`*.*
- int ppl_Polyhedron_intersection_assign (ppl_Polyhedron_t x, ppl_const_Polyhedron_t y)

  Intersects `x` *with polyhedron* `y` *and assigns the result to* `x`*.*
- int ppl_Polyhedron_upper_bound_assign (ppl_Polyhedron_t x, ppl_const_Polyhedron_t y)

  Assigns to `x` *an upper bound of* `x` *and* `y`*.*
- int ppl_Polyhedron_difference_assign (ppl_Polyhedron_t x, ppl_const_Polyhedron_t y)

  Same as ppl_Polyhedron_poly_difference_assign(x, y).
- int ppl_Polyhedron_simplify_using_context_assign (ppl_Polyhedron_t x, ppl_const_Polyhedron_t y)

  Assigns to `x` *the meet-preserving simplification of* `x` *with respect to context* `y`*. Returns a positive integer if* `x` *and* `y` *have a nonempty intersection; returns* `0` *if they are disjoint.*
- int ppl_Polyhedron_time_elapse_assign (ppl_Polyhedron_t x, ppl_const_Polyhedron_t y)

  Assigns to `x` *the time-elapse between the polyhedra* `x` *and* `y`*.*
- int ppl_Polyhedron_topological_closure_assign (ppl_Polyhedron_t ph)

  Assigns to `ph` *its topological closure.*
- int ppl_Polyhedron_unconstrain_space_dimension (ppl_Polyhedron_t ph, ppl_dimension_type var)

  Modifies `ph` *by unconstraining the space dimension* `var`*.*
- int ppl_Polyhedron_unconstrain_space_dimensions (ppl_Polyhedron_t ph, ppl_dimension_type ds[ ], size_t n)

  Modifies `ph` *by unconstraining the space dimensions that are specified in the first* `n` *positions of the array* `ds`*. The presence of duplicates in* `ds` *is a waste but an innocuous one.*

- int ppl_Polyhedron_affine_image (ppl_Polyhedron_t ph, ppl_dimension_type var, ppl_const_Linear_Expression_t le, ppl_const_Coefficient_t d)

  Transforms the polyhedron `ph`, assigning an affine expression to the specified variable.
- int ppl_Polyhedron_affine_preimage (ppl_Polyhedron_t ph, ppl_dimension_type var, ppl_const_Linear_Expression_t le, ppl_const_Coefficient_t d)

  Transforms the polyhedron `ph`, substituting an affine expression to the specified variable.
- int ppl_Polyhedron_bounded_affine_image (ppl_Polyhedron_t ph, ppl_dimension_type var, ppl_const_Linear_Expression_t lb, ppl_const_Linear_Expression_t ub, ppl_const_Coefficient_t d)

  Assigns to `ph` the image of `ph` with respect to the generalized affine transfer relation $\frac{\mathrm{lb}}{\mathrm{d}} \leq \mathrm{var}' \leq \frac{\mathrm{ub}}{\mathrm{d}}$.
- int ppl_Polyhedron_bounded_affine_preimage (ppl_Polyhedron_t ph, ppl_dimension_type var, ppl_const_Linear_Expression_t lb, ppl_const_Linear_Expression_t ub, ppl_const_Coefficient_t d)

  Assigns to `ph` the preimage of `ph` with respect to the generalized affine transfer relation $\frac{\mathrm{lb}}{\mathrm{d}} \leq \mathrm{var}' \leq \frac{\mathrm{ub}}{\mathrm{d}}$.
- int ppl_Polyhedron_generalized_affine_image (ppl_Polyhedron_t ph, ppl_dimension_type var, enum ppl_enum_Constraint_Type relsym, ppl_const_Linear_Expression_t le, ppl_const_Coefficient_t d)

  Assigns to `ph` the image of `ph` with respect to the generalized affine transfer relation $\mathrm{var}' \bowtie \frac{\mathrm{le}}{\mathrm{d}}$, *where* $\bowtie$ *is the relation symbol encoded by `relsym`.*
- int ppl_Polyhedron_generalized_affine_preimage (ppl_Polyhedron_t ph, ppl_dimension_type var, enum ppl_enum_Constraint_Type relsym, ppl_const_Linear_Expression_t le, ppl_const_Coefficient_t d)

  Assigns to `ph` the preimage of `ph` with respect to the generalized affine transfer relation $\mathrm{var}' \bowtie \frac{\mathrm{le}}{\mathrm{d}}$, *where* $\bowtie$ *is the relation symbol encoded by `relsym`.*
- int ppl_Polyhedron_generalized_affine_image_lhs_rhs (ppl_Polyhedron_t ph, ppl_const_Linear_Expression_t lhs, enum ppl_enum_Constraint_Type relsym, ppl_const_Linear_Expression_t rhs)

  Assigns to `ph` the image of `ph` with respect to the generalized affine transfer relation $\mathrm{lhs}' \bowtie \mathrm{rhs}$, *where* $\bowtie$ *is the relation symbol encoded by `relsym`.*
- int ppl_Polyhedron_generalized_affine_preimage_lhs_rhs (ppl_Polyhedron_t ph, ppl_const_Linear_Expression_t lhs, enum ppl_enum_Constraint_Type relsym, ppl_const_Linear_Expression_t rhs)

  Assigns to `ph` the preimage of `ph` with respect to the generalized affine transfer relation $\mathrm{lhs}' \bowtie \mathrm{rhs}$, *where* $\bowtie$ *is the relation symbol encoded by `relsym`.*

Functions that May Modify the Dimension of the Vector Space

- int ppl_Polyhedron_concatenate_assign (ppl_Polyhedron_t x, ppl_const_Polyhedron_t y)

  Seeing a polyhedron as a set of tuples (its points), assigns to `x` all the tuples that can be obtained by concatenating, in the order given, a tuple of `x` with a tuple of `y`.
- int ppl_Polyhedron_add_space_dimensions_and_embed (ppl_Polyhedron_t ph, ppl_dimension_type d)

  Adds `d` new dimensions to the space enclosing the polyhedron `ph` and to `ph` itself.
- int ppl_Polyhedron_add_space_dimensions_and_project (ppl_Polyhedron_t ph, ppl_dimension_type d)

  Adds `d` new dimensions to the space enclosing the polyhedron `ph`.
- int ppl_Polyhedron_remove_space_dimensions (ppl_Polyhedron_t ph, ppl_dimension_type ds[ ], size_t n)

  Removes from the vector space enclosing `ph` the space dimensions that are specified in first `n` positions of the array `ds`. The presence of duplicates in `ds` is a waste but an innocuous one.
- int ppl_Polyhedron_remove_higher_space_dimensions (ppl_Polyhedron_t ph, ppl_dimension_type d)

  Removes the higher dimensions from the vector space enclosing `ph` so that, upon successful return, the new space dimension is `d`.

- int ppl_Polyhedron_map_space_dimensions (ppl_Polyhedron_t ph, ppl_dimension_type maps[ ], size_t n)

  Remaps the dimensions of the vector space according to a partial function. This function is specified by means of the `maps` *array, which has* `n` *entries.*
- int ppl_Polyhedron_expand_space_dimension (ppl_Polyhedron_t ph, ppl_dimension_type d, ppl-_dimension_type m)

  Expands the `d`*-th dimension of the vector space enclosing* `ph` *to* `m` *new space dimensions.*
- int ppl_Polyhedron_fold_space_dimensions (ppl_Polyhedron_t ph, ppl_dimension_type ds[ ], size-_t n, ppl_dimension_type d)

  Modifies `ph` *by folding the space dimensions contained in the first* `n` *positions of the array* `ds` *into dimension* `d`*. The presence of duplicates in* `ds` *is a waste but an innocuous one.*

Input/Output Functions

- int ppl_io_print_Polyhedron (ppl_const_Polyhedron_t x)

  Prints `x` *to* `stdout`*.*
- int ppl_io_fprint_Polyhedron (FILE ∗stream, ppl_const_Polyhedron_t x)

  Prints `x` *to the given output* `stream`*.*
- int ppl_io_asprint_Polyhedron (char ∗∗strp, ppl_const_Polyhedron_t x)

  Prints `x` *to a malloc-allocated string, a pointer to which is returned via* `strp`*.*
- int ppl_Polyhedron_ascii_dump (ppl_const_Polyhedron_t x, FILE ∗stream)

  Dumps an ascii representation of `x` *on* `stream`*.*
- int ppl_Polyhedron_ascii_load (ppl_Polyhedron_t x, FILE ∗stream)

  Loads an ascii representation of `x` *from* `stream`*.*

Ad Hoc Functions for (C or NNC) Polyhedra

The functions listed here below, being specific of the polyhedron domains, do not have a correspondence in other semantic geometric descriptions.

- int ppl_new_C_Polyhedron_from_Generator_System (ppl_Polyhedron_t ∗pph, ppl_const_Generator-_System_t gs)

  Builds a new C polyhedron from the system of generators `gs` *and writes a handle for the newly created polyhedron at address* `pph`*.*
- int ppl_new_C_Polyhedron_recycle_Generator_System (ppl_Polyhedron_t ∗pph, ppl_Generator-_System_t gs)

  Builds a new C polyhedron recycling the system of generators `gs` *and writes a handle for the newly created polyhedron at address* `pph`*.*
- int ppl_new_NNC_Polyhedron_from_Generator_System (ppl_Polyhedron_t ∗pph, ppl_const_-Generator_System_t gs)

  Builds a new NNC polyhedron from the system of generators `gs` *and writes a handle for the newly created polyhedron at address* `pph`*.*
- int ppl_new_NNC_Polyhedron_recycle_Generator_System (ppl_Polyhedron_t ∗pph, ppl_Generator-_System_t gs)

  Builds a new NNC polyhedron recycling the system of generators `gs` *and writes a handle for the newly created polyhedron at address* `pph`*.*
- int ppl_Polyhedron_get_generators (ppl_const_Polyhedron_t ph, ppl_const_Generator_System_t ∗pgs)

  Writes a const handle to the generator system defining the polyhedron `ph` *at address* `pgs`*.*
- int ppl_Polyhedron_get_minimized_generators (ppl_const_Polyhedron_t ph, ppl_const_Generator-_System_t ∗pgs)

  Writes a const handle to the minimized generator system defining the polyhedron `ph` *at address* `pgs`*.*
- int ppl_Polyhedron_add_generator (ppl_Polyhedron_t ph, ppl_const_Generator_t g)

  Adds a copy of the generator `g` *to the system of generators of* `ph`*.*

- int ppl_Polyhedron_add_generators (ppl_Polyhedron_t ph, ppl_const_Generator_System_t gs)
  Adds a copy of the system of generators gs *to the system of generators of* ph*.*
- int ppl_Polyhedron_add_recycled_generators (ppl_Polyhedron_t ph, ppl_Generator_System_t gs)
  Adds the system of generators gs *to the system of generators of* ph*.*
- int ppl_Polyhedron_poly_hull_assign (ppl_Polyhedron_t x, ppl_const_Polyhedron_t y)
  Assigns to x *the poly-hull of* x *and* y*.*
- int ppl_Polyhedron_poly_difference_assign (ppl_Polyhedron_t x, ppl_const_Polyhedron_t y)
  Assigns to x *the poly-difference of* x *and* y*.*
- int wrap_assign (ppl_Polyhedron_t ph, ppl_dimension_type ds[ ], size_t n, ppl_enum_Bounded_Integer_Type_Width w, ppl_enum_Bounded_Integer_Type_Representation r, ppl_enum_Bounded_Integer_Type_Overflow o, const ppl_const_Constraint_System_t *pcs, unsigned complexity_threshold, int wrap_individually)
  Assigns to ph *the polyhedron obtained from* ph *by "wrapping" the vector space defined by the first* n *space dimensions in* ds[]*.*
- int ppl_Polyhedron_BHRZ03_widening_assign_with_tokens (ppl_Polyhedron_t x, ppl_const_Polyhedron_t y, unsigned *tp)
  If the polyhedron y *is contained in (or equal to) the polyhedron* x*, assigns to* x *the BHRZ03-widening of* x *and* y*. If* tp *is not the null pointer, the widening with tokens delay technique is applied with* *tp *available tokens.*
- int ppl_Polyhedron_H79_widening_assign_with_tokens (ppl_Polyhedron_t x, ppl_const_Polyhedron_t y, unsigned *tp)
  If the polyhedron y *is contained in (or equal to) the polyhedron* x*, assigns to* x *the H79-widening of* x *and* y*. If* tp *is not the null pointer, the widening with tokens delay technique is applied with* *tp *available tokens.*
- int ppl_Polyhedron_BHRZ03_widening_assign (ppl_Polyhedron_t x, ppl_const_Polyhedron_t y)
  If the polyhedron y *is contained in (or equal to) the polyhedron* x*, assigns to* x *the BHRZ03-widening of* x *and* y*.*
- int ppl_Polyhedron_H79_widening_assign (ppl_Polyhedron_t x, ppl_const_Polyhedron_t y)
  If the polyhedron y *is contained in (or equal to) the polyhedron* x*, assigns to* x *the H79-widening of* x *and* y*.*
- int ppl_Polyhedron_limited_BHRZ03_extrapolation_assign_with_tokens (ppl_Polyhedron_t x, ppl_const_Polyhedron_t y, ppl_const_Constraint_System_t cs, unsigned *tp)
  If the polyhedron y *is contained in (or equal to) the polyhedron* x*, assigns to* x *the BHRZ03-widening of* x *and* y *intersected with the constraints in* cs *that are satisfied by all the points of* x*. If* tp *is not the null pointer, the widening with tokens delay technique is applied with* *tp *available tokens.*
- int ppl_Polyhedron_limited_H79_extrapolation_assign_with_tokens (ppl_Polyhedron_t x, ppl_const_Polyhedron_t y, ppl_const_Constraint_System_t cs, unsigned *tp)
  If the polyhedron y *is contained in (or equal to) the polyhedron* x*, assigns to* x *the H79-widening of* x *and* y *intersected with the constraints in* cs *that are satisfied by all the points of* x*. If* tp *is not the null pointer, the widening with tokens delay technique is applied with* *tp *available tokens.*
- int ppl_Polyhedron_limited_BHRZ03_extrapolation_assign (ppl_Polyhedron_t x, ppl_const_Polyhedron_t y, ppl_const_Constraint_System_t cs)
  If the polyhedron y *is contained in (or equal to) the polyhedron* x*, assigns to* x *the BHRZ03-widening of* x *and* y *intersected with the constraints in* cs *that are satisfied by all the points of* x*.*
- int ppl_Polyhedron_limited_H79_extrapolation_assign (ppl_Polyhedron_t x, ppl_const_Polyhedron_t y, ppl_const_Constraint_System_t cs)
  If the polyhedron y *is contained in (or equal to) the polyhedron* x*, assigns to* x *the H79-widening of* x *and* y *intersected with the constraints in* cs *that are satisfied by all the points of* x*.*
- int ppl_Polyhedron_bounded_BHRZ03_extrapolation_assign_with_tokens (ppl_Polyhedron_t x, ppl_const_Polyhedron_t y, ppl_const_Constraint_System_t cs, unsigned *tp)
  If the polyhedron y *is contained in (or equal to) the polyhedron* x*, assigns to* x *the BHRZ03-widening of* x *and* y *intersected with the constraints in* cs *that are satisfied by all the points of* x*, further intersected with all the constraints of the form* $\pm v \leq r$ *and* $\pm v < r$*, with* $r \in \mathbb{Q}$*, that are satisfied by all the points of* x*. If* tp *is not the null pointer, the widening with tokens delay technique is applied with* *tp *available tokens.*

- int ppl_Polyhedron_bounded_H79_extrapolation_assign_with_tokens (ppl_Polyhedron_t x, ppl_const_Polyhedron_t y, ppl_const_Constraint_System_t cs, unsigned *tp)

  If the polyhedron `y` *is contained in (or equal to) the polyhedron* `x`*, assigns to* `x` *the H79-widening of* `x` *and* `y` *intersected with the constraints in* `cs` *that are satisfied by all the points of* `x`*, further intersected with all the constraints of the form* $\pm v \leq r$ *and* $\pm v < r$*, with* $r \in \mathbb{Q}$*, that are satisfied by all the points of* `x`*. If* `tp` *is not the null pointer, the widening with tokens delay technique is applied with* `*tp` *available tokens.*
- int ppl_Polyhedron_bounded_BHRZ03_extrapolation_assign (ppl_Polyhedron_t x, ppl_const_Polyhedron_t y, ppl_const_Constraint_System_t cs)

  If the polyhedron `y` *is contained in (or equal to) the polyhedron* `x`*, assigns to* `x` *the BHRZ03-widening of* `x` *and* `y` *intersected with the constraints in* `cs` *that are satisfied by all the points of* `x`*, further intersected with all the constraints of the form* $\pm v \leq r$ *and* $\pm v < r$*, with* $r \in \mathbb{Q}$*, that are satisfied by all the points of* `x`*.*
- int ppl_Polyhedron_bounded_H79_extrapolation_assign (ppl_Polyhedron_t x, ppl_const_Polyhedron_t y, ppl_const_Constraint_System_t cs)

  If the polyhedron `y` *is contained in (or equal to) the polyhedron* `x`*, assigns to* `x` *the H79-widening of* `x` *and* `y` *intersected with the constraints in* `cs` *that are satisfied by all the points of* `x`*, further intersected with all the constraints of the form* $\pm v \leq r$ *and* $\pm v < r$*, with* $r \in \mathbb{Q}$*, that are satisfied by all the points of* `x`*.*

7.25.1 Detailed Description

Types and functions for the domains of C and NNC convex polyhedra.

The types and functions for convex polyhedra provide a single interface for accessing both topologically closed (C) and not necessarily closed (NNC) convex polyhedra. The distinction between C and NNC polyhedra need only be explicitly stated when *creating* or *assigning* a polyhedron object, by means of one of the functions `ppl_new_*` and `ppl_assign_*`.

Having a single datatype does not mean that C and NNC polyhedra can be freely interchanged: as specified in the main manual, most library functions require their arguments to be topologically and/or space-dimension compatible.

7.25.2 Friends And Related Function Documentation

7.25.2.1 int ppl_new_C_Polyhedron_from_C_Polyhedron_with_complexity (ppl_Polyhedron_t * *pph*, ppl_const_Polyhedron_t *ph*, int *complexity*) [related]

Builds a C polyhedron that is a copy of `ph`; writes a handle for the newly created polyhedron at address `pph`.

Note

The complexity argument is ignored.

7.25.2.2 int ppl_new_C_Polyhedron_from_Constraint_System (ppl_Polyhedron_t * *pph*, ppl_const_Constraint_System_t *cs*) [related]

Builds a new C polyhedron from the system of constraints `cs` and writes a handle for the newly created polyhedron at address `pph`.

The new polyhedron will inherit the space dimension of `cs`.

7.25.2.3 int ppl_new_C_Polyhedron_recycle_Constraint_System (ppl_Polyhedron_t * *pph*, ppl_Constraint_System_t *cs*) [related]

Builds a new C polyhedron recycling the system of constraints cs and writes a handle for the newly created polyhedron at address pph.

The new polyhedron will inherit the space dimension of cs.

Warning

This function modifies the constraint system referenced by cs: upon return, no assumption can be made on its value.

7.25.2.4 int ppl_new_C_Polyhedron_from_Congruence_System (ppl_Polyhedron_t * *pph*, ppl_const_Congruence_System_t *cs*) [related]

Builds a new C polyhedron from the system of congruences cs and writes a handle for the newly created polyhedron at address pph.

The new polyhedron will inherit the space dimension of cs.

7.25.2.5 int ppl_new_C_Polyhedron_recycle_Congruence_System (ppl_Polyhedron_t * *pph*, ppl_Congruence_System_t *cs*) [related]

Builds a new C polyhedron recycling the system of congruences cs and writes a handle for the newly created polyhedron at address pph.

The new polyhedron will inherit the space dimension of cs.

Warning

This function modifies the congruence system referenced by cs: upon return, no assumption can be made on its value.

7.25.2.6 int ppl_new_NNC_Polyhedron_from_NNC_Polyhedron_with_complexity (ppl_Polyhedron_t * *pph*, ppl_const_Polyhedron_t *ph*, int *complexity*) [related]

Builds an NNC polyhedron that is a copy of ph; writes a handle for the newly created polyhedron at address pph.

Note

The complexity argument is ignored.

7.25.2.7 int ppl_new_NNC_Polyhedron_from_Constraint_System (ppl_Polyhedron_t * *pph*, ppl_const_Constraint_System_t *cs*) [related]

Builds a new NNC polyhedron from the system of constraints cs and writes a handle for the newly created polyhedron at address pph.

The new polyhedron will inherit the space dimension of cs.

7.25.2.8 int ppl_new_NNC_Polyhedron_recycle_Constraint_System (ppl_Polyhedron_t * *pph*, ppl_Constraint_System_t *cs*) [related]

Builds a new NNC polyhedron recycling the system of constraints cs and writes a handle for the newly created polyhedron at address pph.

The new polyhedron will inherit the space dimension of cs.

Warning

This function modifies the constraint system referenced by cs: upon return, no assumption can be made on its value.

7.25.2.9 int ppl_new_NNC_Polyhedron_from_Congruence_System (ppl_Polyhedron_t * *pph*, ppl_const_Congruence_System_t *cs*) [related]

Builds a new NNC polyhedron from the system of congruences cs and writes a handle for the newly created polyhedron at address pph.

The new polyhedron will inherit the space dimension of cs.

7.25.2.10 int ppl_new_NNC_Polyhedron_recycle_Congruence_System (ppl_Polyhedron_t * *pph*, ppl_Congruence_System_t *cs*) [related]

Builds a new NNC polyhedron recycling the system of congruences cs and writes a handle for the newly created polyhedron at address pph.

The new polyhedron will inherit the space dimension of cs.

Warning

This function modifies the congruence system referenced by cs: upon return, no assumption can be made on its value.

7.25.2.11 int ppl_new_C_Polyhedron_from_NNC_Polyhedron_with_complexity (ppl_Polyhedron_t * *pph*, ppl_const_Polyhedron_t *ph*, int *complexity*) [related]

Builds a C polyhedron that approximates NNC_Polyhedron ph, using an algorithm whose complexity does not exceed complexity; writes a handle for the newly created polyhedron at address pph.

Note

The complexity argument, which can take values PPL_COMPLEXITY_CLASS_POLYNOMIAL, PPL_COMPLEXITY_CLASS_SIMPLEX and PPL_COMPLEXITY_CLASS_ANY, is ignored since the exact constructor has polynomial complexity.

7.25.2.12 int ppl_new_NNC_Polyhedron_from_C_Polyhedron_with_complexity (ppl_Polyhedron_t * *pph*, ppl_const_Polyhedron_t *ph*, int *complexity*) [related]

Builds an NNC polyhedron that approximates C_Polyhedron ph, using an algorithm whose complexity does not exceed complexity; writes a handle for the newly created polyhedron at address pph.

Note

The complexity argument, which can take values PPL_COMPLEXITY_CLASS_POLYNOMIAL, PPL_COMPLEXITY_CLASS_SIMPLEX and PPL_COMPLEXITY_CLASS_ANY, is ignored since the exact constructor has polynomial complexity.

7.25.2.13 int ppl_Polyhedron_relation_with_Constraint (ppl_const_Polyhedron_t *ph*, ppl_const_Constraint_t *c*) [related]

Checks the relation between the polyhedron `ph` and the constraint `c`.

If successful, returns a non-negative integer that is obtained as the bitwise or of the bits (chosen among PPL_POLY_CON_RELATION_IS_DISJOINT PPL_POLY_CON_RELATION_STRICTLY_INTERSECTS, PPL_POLY_CON_RELATION_IS_INCLUDED, and PPL_POLY_CON_RELATION_SATURATES) that describe the relation between `ph` and `c`.

7.25.2.14 int ppl_Polyhedron_relation_with_Generator (ppl_const_Polyhedron_t *ph*, ppl_const_Generator_t *g*) [related]

Checks the relation between the polyhedron `ph` and the generator `g`.

If successful, returns a non-negative integer that is obtained as the bitwise or of the bits (only PPL_POLY_GEN_RELATION_SUBSUMES, at present) that describe the relation between `ph` and `g`.

7.25.2.15 int ppl_Polyhedron_maximize_with_point (ppl_const_Polyhedron_t *ph*, ppl_const_Linear_Expression_t *le*, ppl_Coefficient_t *sup_n*, ppl_Coefficient_t *sup_d*, int * *pmaximum*, ppl_Generator_t *point*) [related]

Returns a positive integer if `ph` is not empty and `le` is bounded from above in `ph`, in which case the supremum value and a point where `le` reaches it are computed.

Parameters

ph	The polyhedron constraining `le`;
le	The linear expression to be maximized subject to `ph`;
sup_n	Will be assigned the numerator of the supremum value;
sup_d	Will be assigned the denominator of the supremum value;
pmaximum	Will store 1 in this location if the supremum is also the maximum, will store 0 otherwise;
point	Will be assigned the point or closure point where `le` reaches the extremum value.

If `ph` is empty or `le` is not bounded from above, 0 will be returned and `sup_n`, `sup_d`, `*pmaximum` and `point` will be left untouched.

7.25.2.16 int ppl_Polyhedron_minimize_with_point (ppl_const_Polyhedron_t *ph*, ppl_const_Linear_Expression_t *le*, ppl_Coefficient_t *inf_n*, ppl_Coefficient_t *inf_d*, int * *pminimum*, ppl_Generator_t *point*) [related]

Returns a positive integer if `ph` is not empty and `le` is bounded from below in `ph`, in which case the infimum value and a point where `le` reaches it are computed.

Parameters

ph	The polyhedron constraining `le`;
le	The linear expression to be minimized subject to `ph`;
inf_n	Will be assigned the numerator of the infimum value;
inf_d	Will be assigned the denominator of the infimum value;
pminimum	Will store 1 in this location if the infimum is also the minimum, will store 0 otherwise;
point	Will be assigned the point or closure point where `le` reaches the extremum value.

If `ph` is empty or `le` is not bounded from below, 0 will be returned and `sup_n`, `sup_d`, `*pmaximum` and `point` will be left untouched.

7.25.2.17 int ppl_Polyhedron_equals_Polyhedron (ppl_const_Polyhedron_t *x*, ppl_const_Polyhedron_t *y*) [related]

Returns a positive integer if x and y are the same polyhedron; returns 0 if they are different.

Note that x and y may be topology- and/or dimension-incompatible polyhedra: in those cases, the value 0 is returned.

7.25.2.18 int ppl_Polyhedron_add_recycled_constraints (ppl_Polyhedron_t *ph*, ppl_Constraint_System_t *cs*) [related]

Adds the system of constraints cs to the system of constraints of ph.

Warning

This function modifies the constraint system referenced by cs: upon return, no assumption can be made on its value.

7.25.2.19 int ppl_Polyhedron_add_recycled_congruences (ppl_Polyhedron_t *ph*, ppl_Congruence_System_t *cs*) [related]

Adds the system of congruences cs to the polyhedron ph.

Warning

This function modifies the congruence system referenced by cs: upon return, no assumption can be made on its value.

7.25.2.20 int ppl_Polyhedron_upper_bound_assign (ppl_Polyhedron_t *x*, ppl_const_Polyhedron_t *y*) [related]

Assigns to x an upper bound of x and y.

For the domain of polyhedra, this is the same as ppl_Polyhedron_poly_hull_assign(x, y).

7.25.2.21 int ppl_Polyhedron_affine_image (ppl_Polyhedron_t *ph*, ppl_dimension_type *var*, ppl_const_Linear_Expression_t *le*, ppl_const_Coefficient_t *d*) [related]

Transforms the polyhedron ph, assigning an affine expression to the specified variable.

Parameters

ph	The polyhedron that is transformed;
var	The variable to which the affine expression is assigned;
le	The numerator of the affine expression;
d	The denominator of the affine expression.

7.25.2.22 int ppl_Polyhedron_affine_preimage (ppl_Polyhedron_t *ph*, ppl_dimension_type *var*, ppl_const_Linear_Expression_t *le*, ppl_const_Coefficient_t *d*) [related]

Transforms the polyhedron ph, substituting an affine expression to the specified variable.

Parameters

ph	The polyhedron that is transformed;
var	The variable to which the affine expression is substituted;
le	The numerator of the affine expression;
d	The denominator of the affine expression.

7.25.2.23 int ppl_Polyhedron_bounded_affine_image (ppl_Polyhedron_t *ph*, ppl_dimension_type *var*, ppl_const_Linear_Expression_t *lb*, ppl_const_Linear_Expression_t *ub*, ppl_const_Coefficient_t *d*) [related]

Assigns to ph the image of ph with respect to the *generalized affine transfer relation* $\frac{\mathrm{lb}}{\mathrm{d}} \leq \mathrm{var}' \leq \frac{\mathrm{ub}}{\mathrm{d}}$.

Parameters

ph	The polyhedron that is transformed;
var	The variable bounded by the generalized affine transfer relation;
lb	The numerator of the lower bounding affine expression;
ub	The numerator of the upper bounding affine expression;
d	The (common) denominator of the lower and upper bounding affine expressions.

7.25.2.24 int ppl_Polyhedron_bounded_affine_preimage (ppl_Polyhedron_t *ph*, ppl_dimension_type *var*, ppl_const_Linear_Expression_t *lb*, ppl_const_Linear_Expression_t *ub*, ppl_const_Coefficient_t *d*) [related]

Assigns to ph the preimage of ph with respect to the *generalized affine transfer relation* $\frac{\mathrm{lb}}{\mathrm{d}} \leq \mathrm{var}' \leq \frac{\mathrm{ub}}{\mathrm{d}}$.

Parameters

ph	The polyhedron that is transformed;
var	The variable bounded by the generalized affine transfer relation;
lb	The numerator of the lower bounding affine expression;
ub	The numerator of the upper bounding affine expression;
d	The (common) denominator of the lower and upper bounding affine expressions.

7.25.2.25 int ppl_Polyhedron_generalized_affine_image (ppl_Polyhedron_t *ph*, ppl_dimension_type *var*, enum ppl_enum_Constraint_Type *relsym*, ppl_const_Linear_Expression_t *le*, ppl_const_Coefficient_t *d*) [related]

Assigns to ph the image of ph with respect to the *generalized affine transfer relation* $\mathrm{var}' \bowtie \frac{\mathrm{le}}{\mathrm{d}}$, where $\bowtie$ is the relation symbol encoded by relsym.

Parameters

ph	The polyhedron that is transformed;
var	The left hand side variable of the generalized affine transfer relation;
relsym	The relation symbol;
le	The numerator of the right hand side affine expression;
d	The denominator of the right hand side affine expression.

7.25.2.26 int ppl_Polyhedron_generalized_affine_preimage (ppl_Polyhedron_t *ph*, ppl_dimension_type *var*, enum ppl_enum_Constraint_Type *relsym*, ppl_const_Linear_Expression_t *le*, ppl_const_Coefficient_t *d*) [related]

Assigns to `ph` the preimage of `ph` with respect to the *generalized affine transfer relation* $\mathrm{var}' \bowtie \frac{\mathrm{le}}{\mathrm{d}}$, where $\bowtie$ is the relation symbol encoded by `relsym`.

Parameters

ph	The polyhedron that is transformed;
var	The left hand side variable of the generalized affine transfer relation;
relsym	The relation symbol;
le	The numerator of the right hand side affine expression;
d	The denominator of the right hand side affine expression.

7.25.2.27 int ppl_Polyhedron_generalized_affine_image_lhs_rhs (ppl_Polyhedron_t *ph*, ppl_const_Linear_Expression_t *lhs*, enum ppl_enum_Constraint_Type *relsym*, ppl_const_Linear_Expression_t *rhs*) [related]

Assigns to `ph` the image of `ph` with respect to the *generalized affine transfer relation* $\mathrm{lhs}' \bowtie \mathrm{rhs}$, where $\bowtie$ is the relation symbol encoded by `relsym`.

Parameters

ph	The polyhedron that is transformed;
lhs	The left hand side affine expression;
relsym	The relation symbol;
rhs	The right hand side affine expression.

7.25.2.28 int ppl_Polyhedron_generalized_affine_preimage_lhs_rhs (ppl_Polyhedron_t *ph*, ppl_const_Linear_Expression_t *lhs*, enum ppl_enum_Constraint_Type *relsym*, ppl_const_Linear_Expression_t *rhs*) [related]

Assigns to `ph` the preimage of `ph` with respect to the *generalized affine transfer relation* $\mathrm{lhs}' \bowtie \mathrm{rhs}$, where $\bowtie$ is the relation symbol encoded by `relsym`.

Parameters

ph	The polyhedron that is transformed;
lhs	The left hand side affine expression;
relsym	The relation symbol;
rhs	The right hand side affine expression.

7.25.2.29 int ppl_Polyhedron_map_space_dimensions (ppl_Polyhedron_t *ph*, ppl_dimension_type *maps[ ]*, size_t *n*) [related]

Remaps the dimensions of the vector space according to a *partial function*. This function is specified by means of the `maps` array, which has `n` entries.

The partial function is defined on dimension `i` if `i < n` and `maps[i] != ppl_not_a_dimension`; otherwise it is undefined on dimension `i`. If the function is defined on dimension `i`, then dimension `i` is mapped onto dimension `maps[i]`.

The result is undefined if `maps` does not encode a partial function with the properties described in the *specification of the mapping operator*.

7.25.2.30 int ppl_new_C_Polyhedron_from_Generator_System (ppl_Polyhedron_t * *pph*, ppl_const_Generator_System_t *gs*) [related]

Builds a new C polyhedron from the system of generators `gs` and writes a handle for the newly created polyhedron at address `pph`.

The new polyhedron will inherit the space dimension of `gs`.

7.25.2.31 int ppl_new_C_Polyhedron_recycle_Generator_System (ppl_Polyhedron_t * *pph*, ppl_Generator_System_t *gs*) [related]

Builds a new C polyhedron recycling the system of generators `gs` and writes a handle for the newly created polyhedron at address `pph`.

The new polyhedron will inherit the space dimension of `gs`.

Warning

This function modifies the generator system referenced by `gs`: upon return, no assumption can be made on its value.

7.25.2.32 int ppl_new_NNC_Polyhedron_from_Generator_System (ppl_Polyhedron_t * *pph*, ppl_const_Generator_System_t *gs*) [related]

Builds a new NNC polyhedron from the system of generators `gs` and writes a handle for the newly created polyhedron at address `pph`.

The new polyhedron will inherit the space dimension of `gs`.

7.25.2.33 int ppl_new_NNC_Polyhedron_recycle_Generator_System (ppl_Polyhedron_t * *pph*, ppl_Generator_System_t *gs*) [related]

Builds a new NNC polyhedron recycling the system of generators `gs` and writes a handle for the newly created polyhedron at address `pph`.

The new polyhedron will inherit the space dimension of `gs`.

Warning

This function modifies the generator system referenced by `gs`: upon return, no assumption can be made on its value.

7.25.2.34 int ppl_Polyhedron_add_recycled_generators (ppl_Polyhedron_t *ph*, ppl_Generator_System_t *gs*) [related]

Adds the system of generators `gs` to the system of generators of `ph`.

Warning

This function modifies the generator system referenced by `gs`: upon return, no assumption can be made on its value.

7.25.2.35 int wrap_assign (ppl_Polyhedron_t *ph*, ppl_dimension_type *ds[ ]*, size_t *n*, ppl_enum_Bounded_Integer_Type_Width *w*, ppl_enum_Bounded_Integer_Type_Representation *r*, ppl_enum_Bounded_Integer_Type_Overflow *o*, const ppl_const_Constraint_System_t * *pcs*, unsigned *complexity_threshold*, int *wrap_individually*) [related]

Assigns to ph the polyhedron obtained from ph by "wrapping" the vector space defined by the first n space dimensions in ds[].

Parameters

ph	The polyhedron that is transformed;
ds[ ]	Specifies the space dimensions to be wrapped.
n	The first n space dimensions in the array ds[] will be wrapped.
w	The width of the bounded integer type corresponding to all the dimensions to be wrapped.
r	The representation of the bounded integer type corresponding to all the dimensions to be wrapped.
o	The overflow behavior of the bounded integer type corresponding to all the dimensions to be wrapped.
pcs	Possibly null pointer to a constraint system whose space dimensions are the first n dimensions in ds[]. If *pcs depends on variables not in vars, the behavior is undefined. When non-null, the constraint system is assumed to represent the conditional or looping construct guard with respect to which wrapping is performed. Since wrapping requires the computation of upper bounds and due to non-distributivity of constraint refinement over upper bounds, passing a constraint system in this way can be more precise than refining the result of the wrapping operation with the constraints in cs.
complexity_threshold	A precision parameter where higher values result in possibly improved precision.
wrap_individually	Non-zero if the dimensions should be wrapped individually (something that results in much greater efficiency to the detriment of precision).

The documentation for this interface was generated from the following file:

- C_interface.dox

Index