The Parma Polyhedra Library Java Language Interface User's Manual[*] (version 1.1)

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

October 28, 2013

[*]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

The Parma Polyhedra Library comes equipped with an interface for the Java language. The Java interface provides access to the numerical abstractions (convex polyhedra, BD shapes, octagonal shapes, etc.) implemented by the PPL library. A general introduction to the numerical abstractions, their representation in the PPL and the operations provided by the PPL is given in the main *PPL user manual*. Here we just describe those aspects that are specific to the Java interface. In the sequel, `prefix` is the path prefix under which the library has been installed (typically `/usr` or `/usr/local`).

Overview

Here is a list of notes with general information and advice on the use of the Java interface.

- When the Parma Polyhedra Library is configured, it will automatically test for the existence of the Java system (unless configuration options are passed to disable the build of the Java interface; see configuration option `--enable-interfaces`). If Java is correctly installed in a standard location, things will be arranged so that the Java interface is built and installed (see configuration option `--with-java` if you need to specify a non-standard location for the Java system).
- The Java interface files are all installed in the directory `prefix/lib/ppl`. Since this includes shared and dynamically loaded libraries, you must make your dynamic linker/loader aware of this fact. If you use a GNU/Linux system, try the commands `man ld.so` and `man ldconfig` for more information.
- Any application using the PPL should:
  - Load the PPL interface library by calling `System.load` and passing the full path of the dynamic shared object;
  - Make sure that only the intended version(s) of the library has been loaded, e.g., by calling static method `version()` in class `parma_polyhedra_library.Parma_Polyhedra_Library`;
  - Starting from version 0.11, initialize the interface by calling static method `initialize_library()`; when all library work is done, finalize the interface by calling `finalize_library()`.
- The numerical abstract domains available to the Java user as Java classes consist of the *simple* domains, *powersets* of a simple domain and *products* of simple domains. Note that the default configuration will only enable a subset of these domains (if you need a different set of domains, see configuration option `--enable-instantiations`).
  - The simple domains are:
    - convex polyhedra, which consist of C_Polyhedron and NNC_Polyhedron;
    - weakly relational, which consist of BD_Shape_N and Octagonal_Shape_N where N is one of the numeric types signed_char, short, int, long, long_long, mpz_class, mpq_class;
    - boxes which consist of Int8_Box, Int16_Box, Int32_Box, Int64_Box, Uint8_Box, Uint16_Box, Uint32_Box, Uint64_Box, Float_Box, Double_Box, Long_Double_Box, Z_Box, Rational_Box; and
    - the Grid domain.
  - The powerset domains are Pointset_Powerset_S where S is a simple domain.
  - The product domains consist of Direct_Product_S_T, Smash_Product_S_T and Constraints_Product_S_T where S and T are simple domains.
- In the following, any of the above numerical abstract domains is called a PPL *domain* and any element of a PPL domain is called a PPL *object*.

- A Java program can create a new object for a PPL domain by using the constructors for the class corresponding to the domain.

- For a PPL object with space dimension `k`, the identifiers used for the PPL variables must lie between 0 and $k-1$ and correspond to the indices of the associated Cartesian axes. For example, when using methods that combine PPL polyhedra or add constraints or generators to a representation of a PPL polyhedron, the polyhedra referenced and any constraints or generators in the call should follow all the (space) dimension-compatibility rules stated in Section *Representations of Convex Polyhedra* of the main PPL user manual.

- As explained above, a polyhedron has a fixed topology C or NNC, that is determined at the time of its initialization. All subsequent operations on the polyhedron must respect all the topological compatibility rules stated in Section *Representations of Convex Polyhedra* of the main PPL user manual.

- A *system* of constraints (i.e., an instance of class `parma_polyhedra_library.Constraint_System`) is implemented by extending class `java.util.ArrayList` (note: `java.util.Vector` was used up to version 1.0.) As a consequence, it is possible to iterate over the constraints in the system by using corresponding inherited methods. Similarly, it is possible to modify a system of constraints by using methods such as `add`; be warned, however, that the constraint system obtained from a PPL object is a *copy* of the (C++) data structure used in the object itself: any modification will not directly affect the original PPL object; clearly, the modified constraint system can be used with appropriate methods to, e.g., build a new PPL object or modify an existing one. The same observations apply to systems of congruences and systems of (grid) generators.

2 GNU General Public License

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

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

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

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation 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. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (c)  YEAR  YOUR NAME.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled
"GNU Free Documentation License".
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

```
with the Invariant Sections being LIST THEIR TITLES, with the
Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

4 Module Index

4.1 Modules

Here is a list of all modules:

5 Namespace Index

5.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

6 Hierarchical Index

6.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

7 Class Index

7.1 Class List

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

8 Module Documentation

8.1 Java Language Interface

Packages

- package parma_polyhedra_library

  The PPL Java interface package.

Classes

- class parma_polyhedra_library.Artificial_Parameter_Sequence

  A sequence of artificial parameters.
- enum parma_polyhedra_library.Bounded_Integer_Type_Overflow

  Overflow behavior of bounded integer types.
- enum parma_polyhedra_library.Bounded_Integer_Type_Representation

  Representation of bounded integer types.
- enum parma_polyhedra_library.Bounded_Integer_Type_Width

  Widths of bounded integer types.
- class parma_polyhedra_library.By_Reference< T >

  An utility class implementing mutable and non-mutable call-by-reference.
- class parma_polyhedra_library.Coefficient

  A PPL coefficient.
- enum parma_polyhedra_library.Complexity_Class

  Possible Complexities.
- class parma_polyhedra_library.Congruence

  A linear congruence.
- class parma_polyhedra_library.Congruence_System

  A system of congruences.
- class parma_polyhedra_library.Constraint

  A linear equality or inequality.
- class parma_polyhedra_library.Constraint_System

  A system of constraints.
- enum parma_polyhedra_library.Control_Parameter_Name

  Names of MIP problems' control parameters.
- enum parma_polyhedra_library.Control_Parameter_Value

Possible values for MIP problem's control parameters.

- enum parma_polyhedra_library.Degenerate_Element

  Kinds of degenerate abstract elements.
- class parma_polyhedra_library.Domain_Error_Exception

  Exceptions caused by domain errors.
- class parma_polyhedra_library.Polyhedron

  The Java base class for (C and NNC) convex polyhedra.
- class parma_polyhedra_library.C_Polyhedron

  A topologically closed convex polyhedron.
- class parma_polyhedra_library.Pointset_Powerset_C_Polyhedron

  A powerset of C_Polyhedron objects.
- class parma_polyhedra_library.Pointset_Powerset_C_Polyhedron_Iterator

  An iterator class for the disjuncts of a Pointset_Powerset_C_Polyhedron.
- class parma_polyhedra_library.Generator

  A line, ray, point or closure point.
- class parma_polyhedra_library.Generator_System

  A system of generators.
- enum parma_polyhedra_library.Generator_Type

  The generator type.
- class parma_polyhedra_library.Grid_Generator

  A grid line, parameter or grid point.
- class parma_polyhedra_library.Grid_Generator_System

  A system of grid generators.
- enum parma_polyhedra_library.Grid_Generator_Type

  The grid generator type.
- class parma_polyhedra_library.Invalid_Argument_Exception

  Exceptions caused by invalid arguments.
- class parma_polyhedra_library.IO

  A class collecting I/O functions.
- class parma_polyhedra_library.Length_Error_Exception

  Exceptions caused by too big length/size values.
- class parma_polyhedra_library.Linear_Expression

  A linear expression.
- class parma_polyhedra_library.Linear_Expression_Coefficient

  A linear expression built from a coefficient.
- class parma_polyhedra_library.Linear_Expression_Difference

  The difference of two linear expressions.
- class parma_polyhedra_library.Linear_Expression_Sum

  The sum of two linear expressions.
- class parma_polyhedra_library.Linear_Expression_Times

  The product of a linear expression and a coefficient.
- class parma_polyhedra_library.Linear_Expression_Unary_Minus

  The negation of a linear expression.
- class parma_polyhedra_library.Linear_Expression_Variable

  A linear expression built from a variable.
- class parma_polyhedra_library.Logic_Error_Exception

Exceptions due to errors in low-level routines.

- class parma_polyhedra_library.MIP_Problem

  A Mixed Integer (linear) Programming problem.
- enum parma_polyhedra_library.MIP_Problem_Status

  Possible outcomes of the MIP_Problem solver.
- enum parma_polyhedra_library.Optimization_Mode

  Possible optimization modes.
- class parma_polyhedra_library.Overflow_Error_Exception

  Exceptions due to overflow errors.
- class parma_polyhedra_library.Pair< K, V >

  A pair of values of type K and V.
- class parma_polyhedra_library.Parma_Polyhedra_Library

  A class collecting library-level functions.
- class parma_polyhedra_library.Partial_Function

  A partial function on space dimension indices.
- class parma_polyhedra_library.PIP_Problem

  A Parametric Integer Programming problem.
- enum parma_polyhedra_library.PIP_Problem_Control_Parameter_Name

  Names of PIP problems' control parameters.
- enum parma_polyhedra_library.PIP_Problem_Control_Parameter_Value

  Possible values for PIP problems' control parameters.
- enum parma_polyhedra_library.PIP_Problem_Status

  Possible outcomes of the PIP_Problem solver.
- class parma_polyhedra_library.Poly_Con_Relation

  The relation between a polyhedron and a constraint.
- enum parma_polyhedra_library.Relation_Symbol

  Relation symbols.
- class parma_polyhedra_library.Timeout_Exception

  Exceptions caused by timeout expiring.
- class parma_polyhedra_library.Variable

  A dimension of the vector space.
- interface parma_polyhedra_library.Variable_Stringifier

  An interface for objects converting a Variable id to a string.

8.1.1 Detailed Description

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

9 Namespace Documentation

9.1 Package parma_polyhedra_library

The PPL Java interface package.

Classes

- class Artificial_Parameter
- class Artificial_Parameter_Sequence

  A sequence of artificial parameters.
- enum Bounded_Integer_Type_Overflow

  Overflow behavior of bounded integer types.
- enum Bounded_Integer_Type_Representation

  Representation of bounded integer types.
- enum Bounded_Integer_Type_Width

  Widths of bounded integer types.
- class By_Reference< T >

  An utility class implementing mutable and non-mutable call-by-reference.
- class Coefficient

  A PPL coefficient.
- enum Complexity_Class

  Possible Complexities.
- class Congruence

  A linear congruence.
- class Congruence_System

  A system of congruences.
- class Constraint

  A linear equality or inequality.
- class Constraint_System

  A system of constraints.
- enum Control_Parameter_Name

  Names of MIP problems' control parameters.
- enum Control_Parameter_Value

  Possible values for MIP problem's control parameters.
- enum Degenerate_Element

  Kinds of degenerate abstract elements.
- class Domain_Error_Exception

  Exceptions caused by domain errors.
- class Polyhedron

  The Java base class for (C and NNC) convex polyhedra.
- class C_Polyhedron

  A topologically closed convex polyhedron.
- class Pointset_Powerset_C_Polyhedron

  A powerset of C_Polyhedron objects.
- class Pointset_Powerset_C_Polyhedron_Iterator

  An iterator class for the disjuncts of a Pointset_Powerset_C_Polyhedron.
- class Generator

  A line, ray, point or closure point.
- class Generator_System

  A system of generators.
- enum Generator_Type

  The generator type.

- class Grid_Generator

  A grid line, parameter or grid point.
- class Grid_Generator_System

  A system of grid generators.
- enum Grid_Generator_Type

  The grid generator type.
- class Invalid_Argument_Exception

  Exceptions caused by invalid arguments.
- class IO

  A class collecting I/O functions.
- class Length_Error_Exception

  Exceptions caused by too big length/size values.
- class Linear_Expression

  A linear expression.
- class Linear_Expression_Coefficient

  A linear expression built from a coefficient.
- class Linear_Expression_Difference

  The difference of two linear expressions.
- class Linear_Expression_Sum

  The sum of two linear expressions.
- class Linear_Expression_Times

  The product of a linear expression and a coefficient.
- class Linear_Expression_Unary_Minus

  The negation of a linear expression.
- class Linear_Expression_Variable

  A linear expression built from a variable.
- class Logic_Error_Exception

  Exceptions due to errors in low-level routines.
- class MIP_Problem

  A Mixed Integer (linear) Programming problem.
- enum MIP_Problem_Status

  Possible outcomes of the MIP_Problem solver.
- enum Optimization_Mode

  Possible optimization modes.
- class Overflow_Error_Exception

  Exceptions due to overflow errors.
- class Pair< K, V >

  A pair of values of type K and V.
- class Parma_Polyhedra_Library

  A class collecting library-level functions.
- class Partial_Function

  A partial function on space dimension indices.
- class PIP_Decision_Node

  An internal node of the PIP solution tree.
- class PIP_Problem

  A Parametric Integer Programming problem.

- enum PIP_Problem_Control_Parameter_Name

  Names of PIP problems' control parameters.
- enum PIP_Problem_Control_Parameter_Value

  Possible values for PIP problems' control parameters.
- enum PIP_Problem_Status

  Possible outcomes of the PIP_Problem solver.
- class PIP_Solution_Node

  A leaf node of the PIP solution tree.
- class PIP_Tree_Node

  A node of the PIP solution tree.
- class Poly_Con_Relation

  The relation between a polyhedron and a constraint.
- class Poly_Gen_Relation

  The relation between a polyhedron and a generator.
- enum Relation_Symbol

  Relation symbols.
- class Timeout_Exception

  Exceptions caused by timeout expiring.
- class Variable

  A dimension of the vector space.
- interface Variable_Stringifier

  An interface for objects converting a Variable id to a string.
- class Variables_Set

  A java.util.TreeSet of variables' indexes.

9.1.1 Detailed Description

The PPL Java interface package. All classes, interfaces and enums related to the Parma Polyhedra Library Java interface are included in this package.

10 Class Documentation

10.1 parma_polyhedra_library.Artificial_Parameter Class Reference

Public Member Functions

- Artificial_Parameter (Linear_Expression e, Coefficient d)

  Builds an artificial parameter from a linear expression and a denominator.
- Linear_Expression linear_expression ()

  Returns the linear expression in artificial parameter `this`.
- Coefficient denominator ()

  Returns the denominator in artificial parameter `this`.
- native String ascii_dump ()

  Returns an ascii formatted internal representation of `this`.
- native String toString ()

  Returns a string representation of `this`.

10.1.1 Detailed Description

An Artificial_Parameter object represents the result of the integer division of a Linear_Expression (on the other parameters, including the previously-defined artificials) by an integer denominator (a Coefficient object). The dimensions of the artificial parameters (if any) in a tree node have consecutive indices starting from `dim+1`, where the value of `dim` is computed as follows:

- for the tree root node, `dim` is the space dimension of the PIP_Problem;
- for any other node of the tree, it is recusrively obtained by adding the value of `dim` computed for the parent node to the number of artificial parameters defined in the parent node.

> Since the numbering of dimensions for artificial parameters follows the rule above, the addition of new problem variables and/or new problem parameters to an already solved PIP_Problem object (as done when incrementally solving a problem) will result in the systematic renumbering of all the existing artificial parameters.

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

- Artificial_Parameter.java

10.2 parma_polyhedra_library.Artificial_Parameter_Sequence Class Reference

A sequence of artificial parameters.
Inherits ArrayList< Artificial_Parameter >.

Public Member Functions

- Artificial_Parameter_Sequence ()
  Default constructor: builds an empty sequence of artificial parameters.

10.2.1 Detailed Description

A sequence of artificial parameters.
An object of the class Artificial_Parameter_Sequence is a sequence of artificial parameters.
The documentation for this class was generated from the following file:

- Artificial_Parameter_Sequence.java

10.3 parma_polyhedra_library.Bounded_Integer_Type_Overflow Enum Reference

Overflow behavior of bounded integer types.

Public Attributes

- OVERFLOW_WRAPS
  On overflow, wrapping takes place.
- OVERFLOW_UNDEFINED
  On overflow, the result is undefined.

10.3.1 Detailed Description

Overflow behavior of bounded integer types.
The documentation for this enum was generated from the following file:

- Bounded_Integer_Type_Overflow.java

10.4 parma_polyhedra_library.Bounded_Integer_Type_Representation Enum Reference

Representation of bounded integer types.

Public Attributes

- UNSIGNED

  Unsigned binary.

10.4.1 Detailed Description

Representation of bounded integer types.

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

- Bounded_Integer_Type_Representation.java

10.5 parma_polyhedra_library.Bounded_Integer_Type_Width Enum Reference

Widths of bounded integer types.

Public Attributes

- BITS_8

  Minimization is requested.
- BITS_16

  16 bits.
- BITS_32

  32 bits.
- BITS_64

  64 bits.

10.5.1 Detailed Description

Widths of bounded integer types.

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

- Bounded_Integer_Type_Width.java

10.6 parma_polyhedra_library.By_Reference< T > Class Reference

An utility class implementing mutable and non-mutable call-by-reference.

Public Member Functions

- By_Reference (T object_value)

  Builds an object encapsulating `object_value`.
- void set (T y)

  Set an object to value `object_value`.
- T get ()

  Returns the value held by `this`.

10.6.1 Detailed Description

An utility class implementing mutable and non-mutable call-by-reference.

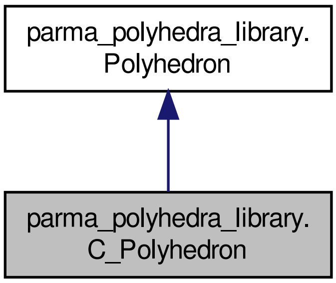
The documentation for this class was generated from the following file:

- By_Reference.java

10.7 parma_polyhedra_library.C_Polyhedron Class Reference

A topologically closed convex polyhedron.

Inheritance diagram for parma_polyhedra_library.C_Polyhedron:



Public Member Functions

Standard Constructors and Destructor

- C_Polyhedron (long d, Degenerate_Element kind)
  Builds a new C polyhedron of dimension `d`*.*
- C_Polyhedron (C_Polyhedron y)
  Builds a new C polyhedron that is copy of `y`*.*
- C_Polyhedron (C_Polyhedron y, Complexity_Class complexity)
  Builds a new C polyhedron that is a copy of `ph`*.*
- C_Polyhedron (Constraint_System cs)
  Builds a new C polyhedron from the system of constraints `cs`*.*
- C_Polyhedron (Congruence_System cgs)
  Builds a new C polyhedron from the system of congruences `cgs`*.*
- native void free ()
  Releases all resources managed by `this`*, also resetting it to a null reference.*

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.,* `Grid(C_Polyhedron y)`*,* `C_Polyhedron(BD_Shape_mpq_class y)`*, 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.*

- C_Polyhedron (NNC_Polyhedron y)

Builds a C polyhedron that is a copy of the topological closure of the NNC polyhedron `y`.
- C_Polyhedron (NNC_Polyhedron y, Complexity_Class complexity)

  Builds a C polyhedron that is a copy of the topological closure of the NNC polyhedron `y`.
- C_Polyhedron (Generator_System gs)

  Builds a new C polyhedron from the system of generators `gs`.

Other Methods

- native boolean upper_bound_assign_if_exact (C_Polyhedron y)

  If the upper bound of `this` and `y` is exact it is assigned to `this` and `true` is returned; otherwise `false` is returned.

Static Public Member Functions

- static native Pair < C_Polyhedron, Pointset_Powerset_NNC_Polyhedron > linear_partition (C_Polyhedron p, C_Polyhedron q)

  Partitions `q` with respect to `p`.

Protected Member Functions

- native void finalize ()

  Releases all resources managed by `this`.

10.7.1 Detailed Description

A topologically closed convex polyhedron.

10.7.2 Constructor & Destructor Documentation

parma_polyhedra_library.C_Polyhedron.C_Polyhedron (long *d*, Degenerate_Element *kind*) Builds a new C polyhedron of dimension `d`.

If `kind` is `EMPTY`, the newly created polyhedron will be empty; otherwise, it will be a universe polyhedron.

parma_polyhedra_library.C_Polyhedron.C_Polyhedron (C_Polyhedron *y*, Complexity_Class *complexity*) Builds a new C polyhedron that is a copy of `ph`.

The complexity argument is ignored.

parma_polyhedra_library.C_Polyhedron.C_Polyhedron (Constraint_System *cs*) Builds a new C polyhedron from the system of constraints `cs`.

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

parma_polyhedra_library.C_Polyhedron.C_Polyhedron (Congruence_System *cgs*) Builds a new C polyhedron from the system of congruences `cgs`.

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

parma_polyhedra_library.C_Polyhedron.C_Polyhedron (NNC_Polyhedron *y*, Complexity_Class *complexity*) Builds a C polyhedron that is a copy of the topological closure of the NNC polyhedron `y`.

The complexity argument is ignored, since the exact constructor has polynomial complexity.

parma_polyhedra_library.C_Polyhedron.C_Polyhedron (Generator_System *gs*) Builds a new C polyhedron from the system of generators `gs`.

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

10.7.3 Member Function Documentation

native boolean parma_polyhedra_library.C_Polyhedron.upper_bound_assign_if_exact (C_Polyhedron *y*) If the upper bound of `this` and `y` is exact it is assigned to `this` and `true` is returned; otherwise `false` is returned.

Exceptions

Invalid_Argument_-Exception	Thrown if `this` and `y` are dimension-incompatible.

static native Pair<C_Polyhedron, Pointset_Powerset_NNC_Polyhedron> parma_polyhedra_library.-C_Polyhedron.linear_partition (C_Polyhedron *p*, C_Polyhedron *q*) `[static]` Partitions `q` with respect to `p`.

Let `p` and `q` be two polyhedra. The function returns a pair object `r` such that

- `r.first` is the intersection of `p` and `q`;
- `r.second` has the property that all its elements are pairwise disjoint and disjoint from `p`;
- the set-theoretical union of `r.first` with all the elements of `r.second` gives `q` (i.e., `r` is the representation of a partition of `q`).

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

- Fake_Class_for_Doxygen.java

10.8 parma_polyhedra_library.Coefficient Class Reference

A PPL coefficient.

Public Member Functions

- Coefficient (int i)

  Builds a coefficient valued `i`*.*
- Coefficient (long l)

  Builds a coefficient valued `l`*.*
- Coefficient (BigInteger bi)

  Builds a coefficient valued `bi`*.*
- Coefficient (String s)

  Builds a coefficient from the decimal representation in `s`*.*
- Coefficient (Coefficient c)

  Builds a copy of `c`*.*
- String toString ()

  Returns a String representation of `this`*.*
- BigInteger getBigInteger ()

  Returns the value held by `this`*.*

Static Public Member Functions

- static native int bits ()

  Returns the number of bits of PPL coefficients; 0 if unbounded.

10.8.1 Detailed Description

A PPL coefficient.

Objects of type Coefficient are used to implement the integral valued coefficients occurring in linear expressions, constraints, generators and so on.

10.8.2 Constructor & Destructor Documentation

parma_polyhedra_library.Coefficient.Coefficient (String *s*) `[inline]` Builds a coefficient from the decimal representation in `s`.

Exceptions

java.lang.NumberFormatException	Thrown if `s` does not contain a valid decimal representation.

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

- Coefficient.java

10.9 parma_polyhedra_library.Complexity_Class Enum Reference

Possible Complexities.

Public Attributes

- POLYNOMIAL_COMPLEXITY

  Worst-case polynomial complexity.
- SIMPLEX_COMPLEXITY

  Worst-case exponential complexity but typically polynomial behavior.

10.9.1 Detailed Description

Possible Complexities.

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

- Complexity_Class.java

10.10 parma_polyhedra_library.Congruence Class Reference

A linear congruence.

Public Member Functions

- Congruence (Linear_Expression e1, Linear_Expression e2, Coefficient m)

  Returns the congruence $\mathtt{e1} = \mathtt{e2} \pmod{\mathtt{m}}$.
- Linear_Expression left_hand_side ()

  Returns the left hand side of `this`.
- Linear_Expression right_hand_side ()

  Returns the right hand side of `this`.
- Coefficient modulus ()

  Returns the relation symbol of `this`.
- native String ascii_dump ()

  Returns an ascii formatted internal representation of `this`.
- native String toString ()

  Returns a string representation of `this`.

Protected Attributes

- Coefficient mod

  The modulus of the congruence.

10.10.1 Detailed Description

A linear congruence.

An object of the class Congruence is an object represeting a congruence:

- $\mathrm{cg} = \sum_{i=0}^{n-1} a_i x_i + b = 0 \pmod{m}$

where n is the dimension of the space, a_i is the integer coefficient of variable x_i, b is the integer inhomogeneous term and m is the integer modulus; if $m = 0$, then cg represents the equality congruence $\sum_{i=0}^{n-1} a_i x_i + b = 0$ and, if $m \neq 0$, then the congruence cg is said to be a proper congruence.

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

- Congruence.java

10.11 parma_polyhedra_library.Congruence_System Class Reference

A system of congruences.

Inherits ArrayList< Congruence >.

Public Member Functions

- Congruence_System ()

  Default constructor: builds an empty system of congruences.
- native String ascii_dump ()

  Returns an ascii formatted internal representation of `this`.
- native String toString ()

  Returns a string representation of `this`.

10.11.1 Detailed Description

A system of congruences.

An object of the class Congruence_System is a system of congruences, i.e., a multiset of objects of the class Congruence.

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

- Congruence_System.java

10.12 parma_polyhedra_library.Constraint Class Reference

A linear equality or inequality.

Public Member Functions

- Constraint (Linear_Expression le1, Relation_Symbol rel_sym, Linear_Expression le2)

  Builds a constraint from two linear expressions with a specified relation symbol.
- Linear_Expression left_hand_side ()

  Returns the left hand side of `this`.
- Linear_Expression right_hand_side ()

  Returns the right hand side of `this`.
- Relation_Symbol kind ()

Returns the relation symbol of `this`*.*

- native String ascii_dump ()

  Returns an ascii formatted internal representation of `this`*.*

- native String toString ()

  Returns a string representation of `this`*.*

10.12.1 Detailed Description

A linear equality or inequality.

An object of the class Constraint is either:

- a linear equality;
- a non-strict linear inequality;
- a strict linear inequality.

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

- Constraint.java

10.13 parma_polyhedra_library.Constraint_System Class Reference

A system of constraints.

Inherits ArrayList< Constraint >.

Public Member Functions

- Constraint_System ()

  Default constructor: builds an empty system of constraints.

- native String ascii_dump ()

  Returns an ascii formatted internal representation of `this`*.*

- native String toString ()

  Returns a string representation of `this`*.*

10.13.1 Detailed Description

A system of constraints.

An object of the class Constraint_System is a system of constraints, i.e., a multiset of objects of the class Constraint.

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

- Constraint_System.java

10.14 parma_polyhedra_library.Control_Parameter_Name Enum Reference

Names of MIP problems' control parameters.

Public Attributes

- PRICING

  The pricing rule.

10.14.1 Detailed Description

Names of MIP problems' control parameters.

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

- Control_Parameter_Name.java

10.15 parma_polyhedra_library.Control_Parameter_Value Enum Reference

Possible values for MIP problem's control parameters.

Public Attributes

- PRICING_STEEPEST_EDGE_FLOAT

  Steepest edge pricing method, using floating points (default).
- PRICING_STEEPEST_EDGE_EXACT

  Steepest edge pricing method, using Coefficient.
- PRICING_TEXTBOOK

  Textbook pricing method.

10.15.1 Detailed Description

Possible values for MIP problem's control parameters.

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

- Control_Parameter_Value.java

10.16 parma_polyhedra_library.Degenerate_Element Enum Reference

Kinds of degenerate abstract elements.

Public Attributes

- UNIVERSE

  The universe element, i.e., the whole vector space.

10.16.1 Detailed Description

Kinds of degenerate abstract elements.

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

- Degenerate_Element.java

10.17 parma_polyhedra_library.Domain_Error_Exception Class Reference

Exceptions caused by domain errors.

Inherits RuntimeException.

Public Member Functions

- Domain_Error_Exception (String s)

  Constructor.

10.17.1 Detailed Description

Exceptions caused by domain errors.

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

- Domain_Error_Exception.java

10.18 parma_polyhedra_library.Generator Class Reference

A line, ray, point or closure point.

Public Member Functions

- Generator_Type type ()

  Returns the generator type.
- Linear_Expression linear_expression ()

  Returns the linear expression in `this`*.*
- Coefficient divisor ()

  If `this` *is either a point or a closure point, returns its divisor.*
- native String ascii_dump ()

  Returns an ascii formatted internal representation of `this`*.*
- native String toString ()

  Returns a string representation of `this`*.*

Static Public Member Functions

- static Generator closure_point (Linear_Expression e, Coefficient d)

  Returns the closure point at `e` */* `d`*.*
- static Generator line (Linear_Expression e)

  Returns the line of direction `e`*.*
- static Generator point (Linear_Expression e, Coefficient d)

  Returns the point at `e` */* `d`*.*
- static Generator ray (Linear_Expression e)

  Returns the ray of direction `e`*.*

10.18.1 Detailed Description

A line, ray, point or closure point.

An object of the class Generator is one of the following:

- a line;
- a ray;
- a point;
- a closure point.

10.18.2 Member Function Documentation

static Generator parma_polyhedra_library.Generator.closure_point (Linear_Expression *e*, Coefficient *d*) `[inline], [static]` Returns the closure point at `e` / `d`.

Exceptions

RuntimeErrorException	Thrown if `d` is zero.

static Generator parma_polyhedra_library.Generator.line (Linear_Expression *e*) `[inline]`, `[static]`
Returns the line of direction `e`.

Exceptions

RuntimeErrorException	Thrown if the homogeneous part of `e` represents the origin of the vector space.

static Generator parma_polyhedra_library.Generator.point (Linear_Expression *e*, Coefficient *d*) `[inline]`, `[static]` Returns the point at `e` / `d`.

Exceptions

RuntimeErrorException	Thrown if `d` is zero.

static Generator parma_polyhedra_library.Generator.ray (Linear_Expression *e*) `[inline]`, `[static]`
Returns the ray of direction `e`.

Exceptions

RuntimeErrorException	Thrown if the homogeneous part of `e` represents the origin of the vector space.

Coefficient parma_polyhedra_library.Generator.divisor ( ) `[inline]` If `this` is either a point or a closure point, returns its divisor.

Exceptions

RuntimeErrorException	Thrown if `this` is neither a point nor a closure point.

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

- Generator.java

10.19 parma_polyhedra_library.Generator_System Class Reference

A system of generators.

Inherits ArrayList< Generator >.

Public Member Functions

- Generator_System ()

  Default constructor: builds an empty system of generators.
- native String ascii_dump ()

  Returns an ascii formatted internal representation of `this`.
- native String toString ()

  Returns a string representation of `this`.

10.19.1 Detailed Description

A system of generators.

An object of the class Generator_System is a system of generators, i.e., a multiset of objects of the class Generator (lines, rays, points and closure points).

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

- Generator_System.java

10.20 parma_polyhedra_library.Generator_Type Enum Reference

The generator type.

Public Attributes

- LINE

  The generator is a line.
- RAY

  The generator is a ray.
- POINT

  The generator is a point.

10.20.1 Detailed Description

The generator type.

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

- Generator_Type.java

10.21 parma_polyhedra_library.Grid_Generator Class Reference

A grid line, parameter or grid point.

Public Member Functions

- Grid_Generator_Type type ()

  Returns the generator type.
- Linear_Expression linear_expression ()

  Returns the linear expression in `this`.
- Coefficient divisor ()

  If `this` *is either a grid point or a parameter, returns its divisor.*
- native String ascii_dump ()

  Returns an ascii formatted internal representation of `this`.
- native String toString ()

  Returns a string representation of `this`.

Static Public Member Functions

- static Grid_Generator grid_line (Linear_Expression e)

  Returns the line of direction `e`.
- static Grid_Generator parameter (Linear_Expression e, Coefficient d)

  Returns the parameter at `e` */* `d`.
- static Grid_Generator grid_point (Linear_Expression e, Coefficient d)

  Returns the point at `e` */* `d`.

10.21.1 Detailed Description

A grid line, parameter or grid point.

An object of the class Grid_Generator is one of the following:

- a grid_line;
- a parameter;
- a grid_point.

10.21.2 Member Function Documentation

static Grid_Generator parma_polyhedra_library.Grid_Generator.grid_line (Linear_Expression *e*) `[inline], [static]` Returns the line of direction `e`.

Exceptions

RuntimeErrorException	Thrown if the homogeneous part of `e` represents the origin of the vector space.

static Grid_Generator parma_polyhedra_library.Grid_Generator.parameter (Linear_Expression *e*, Coefficient *d*) `[inline], [static]` Returns the parameter at `e` / `d`.

Exceptions

RuntimeErrorException	Thrown if `d` is zero.

static Grid_Generator parma_polyhedra_library.Grid_Generator.grid_point (Linear_Expression *e*, Coefficient *d*) `[inline], [static]` Returns the point at `e` / `d`.

Exceptions

RuntimeErrorException	Thrown if `d` is zero.

Coefficient parma_polyhedra_library.Grid_Generator.divisor ( ) `[inline]` If `this` is either a grid point or a parameter, returns its divisor.

Exceptions

RuntimeErrorException	Thrown if `this` is a line.

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

- Grid_Generator.java

10.22 parma_polyhedra_library.Grid_Generator_System Class Reference

A system of grid generators.

Inherits ArrayList< Grid_Generator >.

Public Member Functions

- Grid_Generator_System ()

  Default constructor: builds an empty system of grid generators.
- native String ascii_dump ()

  Returns an ascii formatted internal representation of `this`.
- native String toString ()

  Returns a string representation of `this`.

10.22.1 Detailed Description

A system of grid generators.

An object of the class Grid_Generator_System is a system of grid generators, i.e., a multiset of objects of the class Grid_Generator.

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

- Grid_Generator_System.java

10.23 parma_polyhedra_library.Grid_Generator_Type Enum Reference

The grid generator type.

Public Attributes

- LINE

  The generator is a line.
- PARAMETER

  The generator is a parameter.

10.23.1 Detailed Description

The grid generator type.

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

- Grid_Generator_Type.java

10.24 parma_polyhedra_library.Invalid_Argument_Exception Class Reference

Exceptions caused by invalid arguments.

Inherits RuntimeException.

Public Member Functions

- Invalid_Argument_Exception (String s)

  Constructor.

10.24.1 Detailed Description

Exceptions caused by invalid arguments.

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

- Invalid_Argument_Exception.java

10.25 parma_polyhedra_library.IO Class Reference

A class collecting I/O functions.

Static Public Member Functions

- static native String wrap_string (String str, int indent_depth, int preferred_first_line_length, int preferred_line_length)

  Utility function for the wrapping of lines of text.

10.25.1 Detailed Description

A class collecting I/O functions.

10.25.2 Member Function Documentation

static native String parma_polyhedra_library.IO.wrap_string (String *str*, int *indent_depth*, int *preferred_first_line_length*, int *preferred_line_length*) `[static]` Utility function for the wrapping of lines of text.

Parameters

str	The source string holding the lines 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.

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

- IO.java

10.26 parma_polyhedra_library.Length_Error_Exception Class Reference

Exceptions caused by too big length/size values.

Inherits RuntimeException.

Public Member Functions

- Length_Error_Exception (String s)

    Constructor.

10.26.1 Detailed Description

Exceptions caused by too big length/size values.

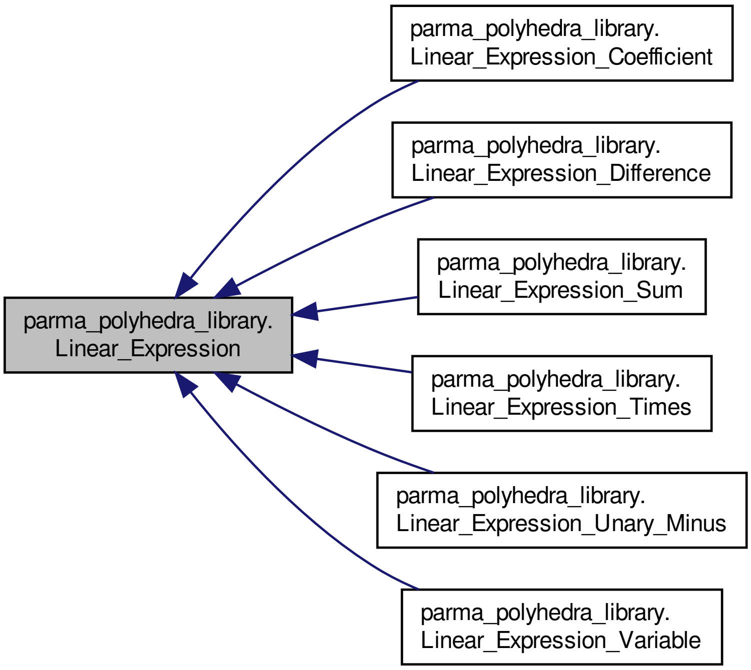
The documentation for this class was generated from the following file:

- Length_Error_Exception.java

10.27 parma_polyhedra_library.Linear_Expression Class Reference

A linear expression.

Inheritance diagram for parma_polyhedra_library.Linear_Expression:



Public Member Functions

- Linear_Expression sum (Linear_Expression y)

  Returns the sum of `this` and `y`.
- Linear_Expression subtract (Linear_Expression y)

  Returns the difference of `this` and `y`.
- Linear_Expression times (Coefficient c)

  Returns the product of `this` times `c`.
- Linear_Expression unary_minus ()

  Returns the negation of `this`.
- abstract Linear_Expression clone ()

  Returns a copy of the linear expression.
- native String ascii_dump ()

  Returns an ascii formatted internal representation of `this`.
- native String toString ()

  Returns a string representation of `this`.
- native boolean is_zero ()

  Returns `true` if and only if `this` is 0.
- native boolean all_homogeneous_terms_are_zero ()

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

10.27.1 Detailed Description

A linear expression.

An object of the class Linear_Expression represents a linear expression that can be built from a Linear_Expression_Variable, Linear_Expression_Coefficient, Linear_Expression_Sum, Linear_Expression_Difference, Linear_Expression_Unary_Minus.

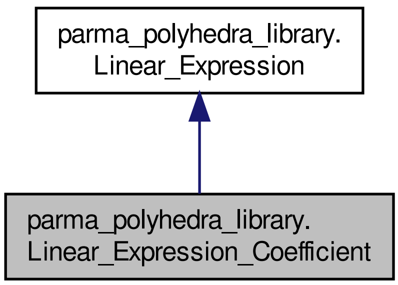
The documentation for this class was generated from the following file:

- Linear_Expression.java

10.28 parma_polyhedra_library.Linear_Expression_Coefficient Class Reference

A linear expression built from a coefficient.

Inheritance diagram for parma_polyhedra_library.Linear_Expression_Coefficient:



Public Member Functions

- Linear_Expression_Coefficient (Coefficient c)

  Builds the object corresponding to a copy of the coefficient `c`*.*
- Coefficient argument ()

  Returns coefficient representing the linear expression.
- Linear_Expression_Coefficient clone ()

  Builds a copy of this.

Protected Attributes

- Coefficient coeff

  The coefficient representing the linear expression.

10.28.1 Detailed Description

A linear expression built from a coefficient.

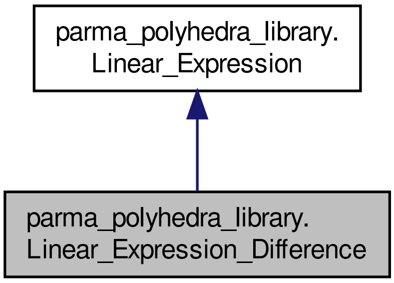
The documentation for this class was generated from the following file:

- Linear_Expression_Coefficient.java

10.29 parma_polyhedra_library.Linear_Expression_Difference Class Reference

The difference of two linear expressions.

Inheritance diagram for parma_polyhedra_library.Linear_Expression_Difference:



Public Member Functions

- Linear_Expression_Difference (Linear_Expression x, Linear_Expression y)

  Builds an object that represents the difference of the copy `x` *and* `y`*.*
- Linear_Expression left_hand_side ()

  Returns the left hand side of `this`*.*
- Linear_Expression right_hand_side ()

  Returns the left hand side of `this`*.*
- Linear_Expression_Difference clone ()

  Builds a copy of this.

Protected Attributes

- Linear_Expression lhs

  The value of the left hand side of `this`*.*
- Linear_Expression rhs

  The value of the right hand side of `this`*.*

10.29.1 Detailed Description

The difference of two linear expressions.

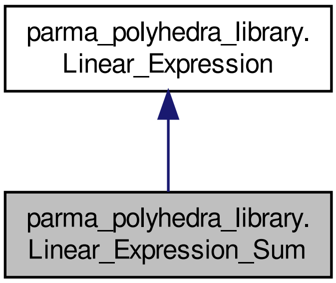
The documentation for this class was generated from the following file:

- Linear_Expression_Difference.java

10.30 parma_polyhedra_library.Linear_Expression_Sum Class Reference

The sum of two linear expressions.

Inheritance diagram for parma_polyhedra_library.Linear_Expression_Sum:



Public Member Functions

- Linear_Expression_Sum (Linear_Expression x, Linear_Expression y)

  Builds an object that represents the sum of the copy of `x` *and* `y`.
- Linear_Expression left_hand_side ()

  Returns the left hand side of `this`.
- Linear_Expression right_hand_side ()

  Returns the right hand side of `this`.
- Linear_Expression_Sum clone ()

  Builds a copy of this.

Protected Attributes

- Linear_Expression lhs

  The value of the left hand side of `this`.
- Linear_Expression rhs

  The value of the right hand side of `this`.

10.30.1 Detailed Description

The sum of two linear expressions.

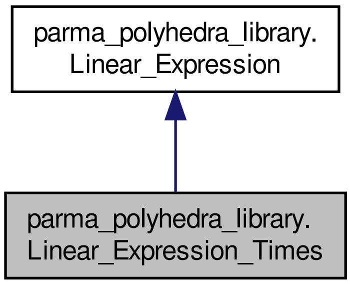
The documentation for this class was generated from the following file:

- Linear_Expression_Sum.java

10.31 parma_polyhedra_library.Linear_Expression_Times Class Reference

The product of a linear expression and a coefficient.

Inheritance diagram for parma_polyhedra_library.Linear_Expression_Times:



Public Member Functions

- Linear_Expression_Times (Coefficient c, Variable v)

  Builds an object cloning the input arguments.
- Linear_Expression_Times (Coefficient c, Linear_Expression l)

  Builds an object cloning the input arguments.
- Linear_Expression_Times (Linear_Expression l, Coefficient c)

  Builds an object cloning the input arguments.
- Coefficient coefficient ()

  Returns the coefficient of `this`*.*
- Linear_Expression linear_expression ()

  Returns the linear expression subobject of `this`*.*
- Linear_Expression_Times clone ()

  Builds a copy of this.

Protected Attributes

- Coefficient coeff

  The value of the coefficient.
- Linear_Expression lin_expr

  The value of the inner linear expression.

10.31.1 Detailed Description

The product of a linear expression and a coefficient.

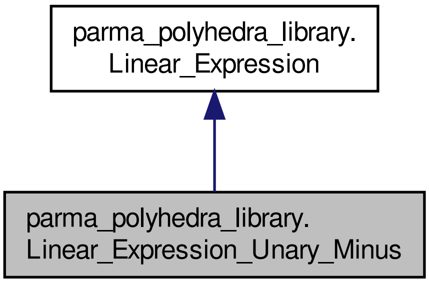
The documentation for this class was generated from the following file:

- Linear_Expression_Times.java

10.32 parma_polyhedra_library.Linear_Expression_Unary_Minus Class Reference

The negation of a linear expression.

Inheritance diagram for parma_polyhedra_library.Linear_Expression_Unary_Minus:



Public Member Functions

- Linear_Expression_Unary_Minus (Linear_Expression x)

  Builds an object that represents the negation of the copy `x`*.*

- Linear_Expression argument ()

  Returns the value that `this` *negates.*

- Linear_Expression_Unary_Minus clone ()

  Builds a copy of this.

Protected Attributes

- Linear_Expression arg

  The value that `this` *negates.*

10.32.1 Detailed Description

The negation of a linear expression.

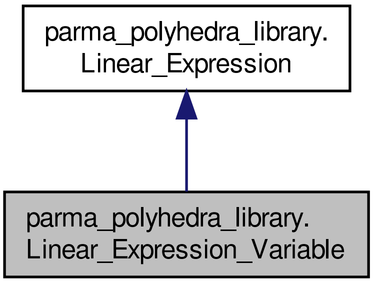
The documentation for this class was generated from the following file:

- Linear_Expression_Unary_Minus.java

10.33 parma_polyhedra_library.Linear_Expression_Variable Class Reference

A linear expression built from a variable.

Inheritance diagram for parma_polyhedra_library.Linear_Expression_Variable:



Public Member Functions

- Linear_Expression_Variable (Variable v)

  Builds the object associated to the copy of v*.*
- Variable argument ()

  Returns the variable representing the linear expression.
- Linear_Expression_Variable clone ()

  Builds a copy of this.

10.33.1 Detailed Description

A linear expression built from a variable.

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

- Linear_Expression_Variable.java

10.34 parma_polyhedra_library.Logic_Error_Exception Class Reference

Exceptions due to errors in low-level routines.

Inherits RuntimeException.

Public Member Functions

- Logic_Error_Exception (String s)

  Constructor.

10.34.1 Detailed Description

Exceptions due to errors in low-level routines.

These exceptions may be generated, for instance, by the inability of querying/controlling the FPU behavior with respect to rounding modes.

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

- Logic_Error_Exception.java

10.35 parma_polyhedra_library.MIP_Problem Class Reference

A Mixed Integer (linear) Programming problem.
Inherits parma_polyhedra_library.PPL_Object.

Public Member Functions

Functions that Do Not Modify the MIP_Problem

- native long max_space_dimension ()
  Returns the maximum space dimension an MIP_Problem can handle.
- native long space_dimension ()
  Returns the space dimension of the MIP problem.
- native Variables_Set integer_space_dimensions ()
  Returns a set containing all the variables' indexes constrained to be integral.
- native Constraint_System constraints ()
  Returns the constraints .
- native Linear_Expression objective_function ()
  Returns the objective function.
- native Optimization_Mode optimization_mode ()
  Returns the optimization mode.
- native String ascii_dump ()
  Returns an ascii formatted internal representation of `this`.
- native String toString ()
  Returns a string representation of `this`.
- native long total_memory_in_bytes ()
  Returns the total size in bytes of the memory occupied by the underlying C++ object.
- native boolean OK ()
  Checks if all the invariants are satisfied.

Functions that May Modify the MIP_Problem

- native void clear ()
  Resets `this` *to be equal to the trivial MIP problem.*
- native void add_space_dimensions_and_embed (long m)
  Adds `m` *new space dimensions and embeds the old MIP problem in the new vector space.*
- native void add_to_integer_space_dimensions (Variables_Set i_vars)
  Sets the variables whose indexes are in set `i_vars` *to be integer space dimensions.*
- native void add_constraint (Constraint c)
  Adds a copy of constraint `c` *to the MIP problem.*
- native void add_constraints (Constraint_System cs)
  Adds a copy of the constraints in `cs` *to the MIP problem.*
- native void set_objective_function (Linear_Expression obj)
  Sets the objective function to `obj`.
- native void set_optimization_mode (Optimization_Mode mode)
  Sets the optimization mode to `mode`.

Computing the Solution of the MIP_Problem

- native boolean is_satisfiable ()
  Checks satisfiability of `this`.
- native MIP_Problem_Status solve ()
  Optimizes the MIP problem.

- native void evaluate_objective_function (Generator evaluating_point, Coefficient num, Coefficient den)

  Sets `num` *and* `den` *so that* $\frac{num}{den}$ *is the result of evaluating the objective function on* `evaluating_point`.
- native Generator feasible_point ()

  Returns a feasible point for `this`, *if it exists.*
- native Generator optimizing_point ()

  Returns an optimal point for `this`, *if it exists.*
- native void optimal_value (Coefficient num, Coefficient den)

  Sets `num` *and* `den` *so that* $\frac{num}{den}$ *is the solution of the optimization problem.*

Querying/Setting Control Parameters

- native Control_Parameter_Value get_control_parameter (Control_Parameter_Name name)

  Returns the value of control parameter `name`.
- native void set_control_parameter (Control_Parameter_Value value)

  Sets control parameter `value`.

Constructors and Destructor

- MIP_Problem (long dim)

  Builds a trivial MIP problem.
- MIP_Problem (long dim, Constraint_System cs, Linear_Expression obj, Optimization_Mode mode)

  Builds an MIP problem having space dimension `dim` *from the constraint system* `cs`, *the objective function* `obj` *and optimization mode* `mode`.
- MIP_Problem (MIP_Problem y)

  Builds a copy of `y`.
- native void free ()

  Releases all resources managed by `this`, *also resetting it to a null reference.*
- native void finalize ()

  Releases all resources managed by `this`.

Additional Inherited Members

10.35.1 Detailed Description

A Mixed Integer (linear) Programming problem.

An object of this class encodes a mixed integer (linear) programming problem. The MIP problem is specified by providing:

- the dimension of the vector space;
- the feasible region, by means of a finite set of linear equality and non-strict inequality constraints;
- the subset of the unknown variables that range over the integers (the other variables implicitly ranging over the reals);
- the objective function, described by a Linear_Expression;
- the optimization mode (either maximization or minimization).

The class provides support for the (incremental) solution of the MIP problem based on variations of the revised simplex method and on branch-and-bound techniques. The result of the resolution process is expressed in terms of an enumeration, encoding the feasibility and the unboundedness of the optimization problem. The class supports simple feasibility tests (i.e., no optimization), as well as the extraction of an optimal (resp., feasible) point, provided the MIP_Problem is optimizable (resp., feasible).

By exploiting the incremental nature of the solver, it is possible to reuse part of the computational work already done when solving variants of a given MIP_Problem: currently, incremental resolution supports the addition of space dimensions, the addition of constraints, the change of objective function and the change of optimization mode.

10.35.2 Constructor & Destructor Documentation

parma_polyhedra_library.MIP_Problem.MIP_Problem (long *dim*) `[inline]` Builds a trivial MIP problem.

A trivial MIP problem requires to maximize the objective function 0 on a vector space under no constraints at all: the origin of the vector space is an optimal solution.

Parameters

dim	The dimension of the vector space enclosing `this`.

Exceptions

Length_Error_Exception	Thrown if `dim` exceeds `max_space_dimension()`.

parma_polyhedra_library.MIP_Problem.MIP_Problem (long *dim*, Constraint_System *cs*, Linear_Expression *obj*, Optimization_Mode *mode*) `[inline]` Builds an MIP problem having space dimension `dim` from the constraint system `cs`, the objective function `obj` and optimization mode `mode`.

Parameters

dim	The dimension of the vector space enclosing `this`.
cs	The constraint system defining the feasible region.
obj	The objective function.
mode	The optimization mode.

Exceptions

Length_Error_Exception	Thrown if `dim` exceeds `max_space_dimension()`.
Invalid_Argument_Exception	Thrown if the constraint system contains any strict inequality or if the space dimension of the constraint system (resp., the objective function) is strictly greater than `dim`.

10.35.3 Member Function Documentation

native void parma_polyhedra_library.MIP_Problem.clear ( ) Resets `this` to be equal to the trivial MIP problem.

The space dimension is reset to 0.

native void parma_polyhedra_library.MIP_Problem.add_space_dimensions_and_embed (long *m*) Adds `m` new space dimensions and embeds the old MIP problem in the new vector space.

Parameters

m	The number of dimensions to add.

Exceptions

Length_Error_Exception	Thrown if adding `m` new space dimensions would cause the vector space to exceed dimension `max_space_dimension()`.

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

native void parma_polyhedra_library.MIP_Problem.add_to_integer_space_dimensions (Variables_-Set *i_vars*) Sets the variables whose indexes are in set `i_vars` to be integer space dimensions.

Exceptions

Invalid_Argument_-Exception	Thrown if some index in `i_vars` does not correspond to a space dimension in `this`.

native void parma_polyhedra_library.MIP_Problem.add_constraint (Constraint *c*) Adds a copy of constraint `c` to the MIP problem.
Exceptions

Invalid_Argument_-Exception	Thrown if the constraint `c` is a strict inequality or if its space dimension is strictly greater than the space dimension of `this`.

native void parma_polyhedra_library.MIP_Problem.add_constraints (Constraint_System *cs*) Adds a copy of the constraints in `cs` to the MIP problem.
Exceptions

Invalid_Argument_-Exception	Thrown if the constraint system `cs` contains any strict inequality or if its space dimension is strictly greater than the space dimension of `this`.

native void parma_polyhedra_library.MIP_Problem.set_objective_function (Linear_Expression *obj*) Sets the objective function to `obj`.
Exceptions

Invalid_Argument_-Exception	Thrown if the space dimension of `obj` is strictly greater than the space dimension of `this`.

native boolean parma_polyhedra_library.MIP_Problem.is_satisfiable ( ) Checks satisfiability of `this`.

Returns

`true` if and only if the MIP problem is satisfiable.

native MIP_Problem_Status parma_polyhedra_library.MIP_Problem.solve ( ) Optimizes the MIP problem.

Returns

An MIP_Problem_Status flag indicating the outcome of the optimization attempt (unfeasible, unbounded or optimized problem).

native void parma_polyhedra_library.MIP_Problem.evaluate_objective_function (Generator *evaluating-_point*, Coefficient *num*, Coefficient *den*) Sets `num` and `den` so that $\frac{num}{den}$ is the result of evaluating the objective function on `evaluating_point`.
Parameters

evaluating_-point	The point on which the objective function will be evaluated.

num	On exit will contain the numerator of the evaluated value.
den	On exit will contain the denominator of the evaluated value.

Exceptions

Invalid_Argument_-Exception	Thrown if `this` and `evaluating_point` are dimension-incompatible or if the generator `evaluating_point` is not a point.

native Generator parma_polyhedra_library.MIP_Problem.feasible_point ( ) Returns a feasible point for `this`, if it exists.
Exceptions

Domain_Error_Exception	Thrown if the MIP problem is not satisfiable.

native Generator parma_polyhedra_library.MIP_Problem.optimizing_point ( ) Returns an optimal point for `this`, if it exists.
Exceptions

Domain_Error_Exception	Thrown if `this` doesn't not have an optimizing point, i.e., if the MIP problem is unbounded or not satisfiable.

native void parma_polyhedra_library.MIP_Problem.optimal_value (Coefficient *num*, Coefficient *den*) Sets `num` and `den` so that $\frac{num}{den}$ is the solution of the optimization problem.
Exceptions

Domain_Error_Exception	Thrown if `this` doesn't not have an optimizing point, i.e., if the MIP problem is unbounded or not satisfiable.

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

- MIP_Problem.java

10.36 parma_polyhedra_library.MIP_Problem_Status Enum Reference

Possible outcomes of the MIP_Problem solver.

Public Attributes

- UNFEASIBLE_MIP_PROBLEM

  The problem is unfeasible.
- UNBOUNDED_MIP_PROBLEM

  The problem is unbounded.

10.36.1 Detailed Description

Possible outcomes of the MIP_Problem solver.
The documentation for this enum was generated from the following file:

- MIP_Problem_Status.java

10.37 parma_polyhedra_library.Optimization_Mode Enum Reference

Possible optimization modes.

Public Attributes

- MINIMIZATION

  Minimization is requested.

10.37.1 Detailed Description

Possible optimization modes.

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

- Optimization_Mode.java

10.38 parma_polyhedra_library.Overflow_Error_Exception Class Reference

Exceptions due to overflow errors.

Inherits RuntimeException.

Public Member Functions

- Overflow_Error_Exception (String s)

  Constructor.

10.38.1 Detailed Description

Exceptions due to overflow errors.

These exceptions can be obtained when the library has been configured to use integer coefficients having bounded size.

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

- Overflow_Error_Exception.java

10.39 parma_polyhedra_library.Pair< K, V > Class Reference

A pair of values of type K and V.

Public Member Functions

- K getFirst ()

  Returns the object of type K.
- V getSecond ()

  Returns the object of type V.

10.39.1 Detailed Description

A pair of values of type K and V.

An object of this class holds an ordered pair of values of type K and V.

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

- Pair.java

10.40 parma_polyhedra_library.Parma_Polyhedra_Library Class Reference

A class collecting library-level functions.

Static Public Member Functions

Library initialization and finalization

- static native void initialize_library ()
  Initializes the Parma Polyhedra Library.
- static native void finalize_library ()
  Finalizes the Parma Polyhedra Library.

Version Checking

- static native int version_major ()
  Returns the major number of the PPL version.
- static native int version_minor ()
  Returns the minor number of the PPL version.
- static native int version_revision ()
  Returns the revision number of the PPL version.
- static native int version_beta ()
  Returns the beta number of the PPL version.
- static native String version ()
  Returns a string containing the PPL version.
- static native String banner ()
  Returns a string containing the PPL banner.

Floating-point rounding and precision settings.

- static native void set_rounding_for_PPL ()
  Sets the FPU rounding mode so that the PPL abstractions based on floating point numbers work correctly.
- static native void restore_pre_PPL_rounding ()
  Sets the FPU rounding mode as it was before initialization of the PPL.
- static native int irrational_precision ()
  Returns the precision parameter for irrational calculations.
- static native void set_irrational_precision (int p)
  Sets the precision parameter used for irrational calculations.

Timeout handling

- static native void set_timeout (int csecs)
  Sets the timeout for computations whose completion could require an exponential amount of time.
- static native void reset_timeout ()
  Resets the timeout time so that the computation is not interrupted.
- static native void set_deterministic_timeout (int unscaled_weight, int scale)
  Sets a threshold for computations whose completion could require an exponential amount of time.
- static native void reset_deterministic_timeout ()
  Resets the deterministic timeout so that the computation is not interrupted.

10.40.1 Detailed Description

A class collecting library-level functions.

10.40.2 Member Function Documentation

static native void parma_polyhedra_library.Parma_Polyhedra_Library.initialize_library ( ) `[static]`
Initializes the Parma Polyhedra Library.

This method must be called after loading the library and before calling any other method from any other PPL package class.

static native void parma_polyhedra_library.Parma_Polyhedra_Library.finalize_library ( ) `[static]`
Finalizes the Parma Polyhedra Library.

This method must be called when work with the library is done. After finalization, no other library method can be called (except those in class Parma_Polyhedra_Library), unless the library is re-initialized by calling `initialize_library()`.

static native String parma_polyhedra_library.Parma_Polyhedra_Library.banner ( ) `[static]`
Returns a 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.

static native void parma_polyhedra_library.Parma_Polyhedra_Library.set_rounding_for_PPL ( ) `[static]` 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.

static native void parma_polyhedra_library.Parma_Polyhedra_Library.restore_pre_PPL_rounding ( ) `[static]` 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.

static native void parma_polyhedra_library.Parma_Polyhedra_Library.set_irrational_precision (int *p*) `[static]` 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`.

static native void parma_polyhedra_library.Parma_Polyhedra_Library.set_timeout (int *csecs*) `[static]`
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, by throwing a `Timeout_Exception` object. Otherwise, if the computation completes without being interrupted, then the timeout should be reset by calling `reset_timeout()`.

static native void parma_polyhedra_library.Parma_Polyhedra_Library.set_deterministic_timeout (int *unscaled_weight*, int *scale*) `[static]` Sets a threshold for computations whose completion could require an exponential amount of time.

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, by throwing a `Timeout_Exception` object. Otherwise, if the computation completes without being interrupted, then the deterministic timeout should be reset by calling `reset_deterministic_timeout()`.

Parameters

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

Exceptions

Invalid_Argument_-Exception	Thrown if the computation of the weight threshold exceeds the maximum allowed value.

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.

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

- Parma_Polyhedra_Library.java

10.41 parma_polyhedra_library.Partial_Function Class Reference

A partial function on space dimension indices.

Inherits parma_polyhedra_library.PPL_Object.

Public Member Functions

- Partial_Function ()

  Builds the empty map.
- native void insert (long i, long j)

  Inserts mapping from `i` to `j`.
- native boolean has_empty_codomain ()

  Returns `true` if and only if the partial function has an empty codomain (i.e., it is always undefined).
- native long max_in_codomain ()

  Returns the maximum value that belongs to the codomain of the partial function.
- native long maps (long i)

  If the partial function is defined on index `i`, returns its value.
- native void free ()

  Releases all resources managed by `this`, also resetting it to a null reference.

Protected Member Functions

- native void finalize ()

  Releases all resources managed by `this`.

10.41.1 Detailed Description

A partial function on space dimension indices.

This class is used in order to specify how space dimensions should be mapped by methods named `map_space_dimensions`.

10.41.2 Member Function Documentation

native boolean parma_polyhedra_library.Partial_Function.has_empty_codomain ( ) Returns `true` if and only if the partial function has an empty codomain (i.e., it is always undefined).

This method will always be called before the other methods of the interface. Moreover, if `true` is returned, then none of the other interface methods will be called.

native long parma_polyhedra_library.Partial_Function.maps (long *i*) If the partial function is defined on index `i`, returns its value.

The function returns a negative value if the partial function is not defined on domain value `i`.

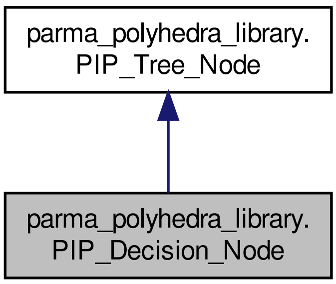
The documentation for this class was generated from the following file:

- Partial_Function.java

10.42 parma_polyhedra_library.PIP_Decision_Node Class Reference

An internal node of the PIP solution tree.

Inheritance diagram for parma_polyhedra_library.PIP_Decision_Node:



Public Member Functions

- native PIP_Tree_Node child_node (boolean branch)

  Returns the true branch (if `branch` is true) or the false branch (if `branch` is false) of `this`.

Additional Inherited Members

10.42.1 Detailed Description

An internal node of the PIP solution tree.

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

- PIP_Decision_Node.java

10.43 parma_polyhedra_library.PIP_Problem Class Reference

A Parametric Integer Programming problem.

Inherits parma_polyhedra_library.PPL_Object.

Public Member Functions

- PIP_Problem (long dim)

  Builds a trivial PIP problem.
- PIP_Problem (long dim, Constraint_System cs, Variables_Set params)

  Builds a PIP problem from a sequence of constraints.
- PIP_Problem (PIP_Problem y)

  Builds a copy of `y`*.*
- native void free ()

  Releases all resources managed by `this`*, also resetting it to a null reference.*

Functions that Do Not Modify the PIP_Problem

- native long max_space_dimension ()

  Returns the maximum space dimension an PIP_Problem can handle.
- native long space_dimension ()

  Returns the space dimension of the PIP problem.
- native long number_of_parameter_space_dimensions ()

  Returns the number of parameter space dimensions of the PIP problem.
- native Variables_Set parameter_space_dimensions ()

  Returns all the parameter space dimensions of problem `pip`*.*
- native long get_big_parameter_dimension ()

  Returns the big parameter dimension of PIP problem `pip`*.*
- native long number_of_constraints ()

  Returns the number of constraints defining the feasible region of `pip`*.*
- native Constraint constraint_at_index (long dim)

  Returns the `i-th` *constraint defining the feasible region of the PIP problem* `pip`*.*
- native Constraint_System constraints ()

  Returns the constraints .
- native String ascii_dump ()

  Returns an ascii formatted internal representation of `this`*.*
- native String toString ()

  Returns a string representation of `this`*.*
- native long total_memory_in_bytes ()

  Returns the size in bytes of the memory occupied by the underlying C++ object.
- native long external_memory_in_bytes ()

  Returns the size in bytes of the memory managed by the underlying C++ object.
- native boolean OK ()

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

Functions that May Modify the PIP_Problem

- native void clear ()

  Resets `this` *to be equal to the trivial PIP problem.*
- native void add_space_dimensions_and_embed (long pip_vars, long pip_params)

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

- native void add_to_parameter_space_dimensions (Variables_Set vars)

  Sets the space dimensions in `vars` *to be parameter dimensions of the PIP problem.*
- native void set_big_parameter_dimension (long d)

  Sets the big parameter dimension of PIP problem to `d`*.*
- native void add_constraint (Constraint c)

  Adds a copy of constraint `c` *to the PIP problem.*
- native void add_constraints (Constraint_System cs)

  Adds a copy of the constraints in `cs` *to the PIP problem.*

Computing the Solution of the PIP_Problem

- native boolean is_satisfiable ()

  Checks satisfiability of `this`*.*
- native PIP_Problem_Status solve ()

  Optimizes the PIP problem.
- native PIP_Tree_Node solution ()

  Returns a solution for the PIP problem, if it exists.
- native PIP_Tree_Node optimizing_solution ()

  Returns an optimizing solution for the PIP problem, if it exists.

Querying/Setting Control Parameters

- native
  PIP_Problem_Control_Parameter_Value get_pip_problem_control_parameter (PIP_Problem_Control_Parameter_Name name)

  Returns the value of control parameter `name`*.*
- native void set_pip_problem_control_parameter (PIP_Problem_Control_Parameter_Value value)

  Sets control parameter `value`*.*

Protected Member Functions

- native void finalize ()

  Releases all resources managed by `this`*.*

10.43.1 Detailed Description

A Parametric Integer Programming problem.

An object of this class encodes a parametric integer (linear) programming problem. The PIP problem is specified by providing:

- the dimension of the vector space;
- the subset of those dimensions of the vector space that are interpreted as integer parameters (the other space dimensions are interpreted as non-parameter integer variables);
- a finite set of linear equality and (strict or non-strict) inequality constraints involving variables and/or parameters; these constraints are used to define:
  - the *feasible region*, if they involve one or more problem variable (and maybe some parameters);
  - the *initial context*, if they only involve the parameters;
- optionally, the so-called *big parameter*, i.e., a problem parameter to be considered arbitrarily big.

Note that all problem variables and problem parameters are assumed to take non-negative integer values, so that there is no need to specify non-negativity constraints.

The class provides support for the (incremental) solution of the PIP problem based on variations of the revised simplex method and on Gomory cut generation techniques.

The solution for a PIP problem is the lexicographic minimum of the integer points of the feasible region, expressed in terms of the parameters. As the problem to be solved only involves non-negative variables and parameters, the problem will always be either unfeasible or optimizable.

As the feasibility and the solution value of a PIP problem depend on the values of the parameters, the solution is a binary decision tree, dividing the context parameter set into subsets. The tree nodes are of two kinds:

- *Decision* nodes. These are internal tree nodes encoding one or more linear tests on the parameters; if all the tests are satisfied, then the solution is the node's *true* child; otherwise, the solution is the node's *false* child;
- *Solution* nodes. These are leaf nodes in the tree, encoding the solution of the problem in the current context subset, where each variable is defined in terms of a linear expression of the parameters. Solution nodes also optionally embed a set of parameter constraints: if all these constraints are satisfied, the solution is described by the node, otherwise the problem has no solution.

It may happen that a decision node has no *false* child. This means that there is no solution if at least one of the corresponding constraints is not satisfied. Decision nodes having two or more linear tests on the parameters cannot have a *false* child. Decision nodes always have a *true* child.

Both kinds of tree nodes may also contain the definition of extra parameters which are artificially introduced by the solver to enforce an integral solution. Such artificial parameters are defined by the integer division of a linear expression on the parameters by an integer coefficient.

By exploiting the incremental nature of the solver, it is possible to reuse part of the computational work already done when solving variants of a given PIP_Problem: currently, incremental resolution supports the addition of space dimensions, the addition of parameters and the addition of constraints.

10.43.2 Constructor & Destructor Documentation

parma_polyhedra_library.PIP_Problem.PIP_Problem (long *dim*) `[inline]` Builds a trivial PIP problem.

A trivial PIP problem requires to compute the lexicographic minimum on a vector space under no constraints and with no parameters: due to the implicit non-negativity constraints, the origin of the vector space is an optimal solution.

Parameters

dim	The dimension of the vector space enclosing `this` (optional argument with default value 0).

Exceptions

Length_Error_Exception	Thrown if `dim` exceeds `max_space_dimension()`.

parma_polyhedra_library.PIP_Problem.PIP_Problem (long *dim*, Constraint_System *cs*, Variables_Set *params*) `[inline]` Builds a PIP problem from a sequence of constraints.

Builds a PIP problem having space dimension `dim` from the constraint system cs; the dimensions `vars` are interpreted as parameters.

10.43.3 Member Function Documentation

native void parma_polyhedra_library.PIP_Problem.clear ( ) Resets `this` to be equal to the trivial PIP problem.

The space dimension is reset to 0.

native void parma_polyhedra_library.PIP_Problem.add_space_dimensions_and_embed (long *pip_vars*, long *pip_params*) Adds `pip_vars + pip_params` new space dimensions and embeds the PIP problem in the new vector space.

Parameters

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.

native void parma_polyhedra_library.PIP_Problem.add_constraint (Constraint *c*) Adds a copy of constraint `c` to the PIP problem.

Exceptions

Invalid_Argument_Exception	Thrown if the constraint `c` is a strict inequality or if its space dimension is strictly greater than the space dimension of `this`.

native void parma_polyhedra_library.PIP_Problem.add_constraints (Constraint_System *cs*) Adds a copy of the constraints in `cs` to the PIP problem.

Exceptions

Invalid_Argument_Exception	Thrown if the constraint system `cs` contains any strict inequality or if its space dimension is strictly greater than the space dimension of `this`.

native boolean parma_polyhedra_library.PIP_Problem.is_satisfiable ( ) Checks satisfiability of `this`.

Returns

`true` if and only if the PIP problem is satisfiable.

native PIP_Problem_Status parma_polyhedra_library.PIP_Problem.solve ( ) Optimizes the PIP problem.

Solves the PIP problem, returning an exit status.

Returns

`UNFEASIBLE_PIP_PROBLEM` if the PIP problem is not satisfiable; `OPTIMIZED_PIP_PROBLEM` if the PIP problem admits an optimal solution.

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

- PIP_Problem.java

10.44 parma_polyhedra_library.PIP_Problem_Control_Parameter_Name Enum Reference

Names of PIP problems' control parameters.

Public Attributes

- CUTTING_STRATEGY

  The cutting strategy rule.
- PIVOT_ROW_STRATEGY

  The pivot row strategy rule.

10.44.1 Detailed Description

Names of PIP problems' control parameters.

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

- PIP_Problem_Control_Parameter_Name.java

10.45 parma_polyhedra_library.PIP_Problem_Control_Parameter_Value Enum Reference

Possible values for PIP problems' control parameters.

Public Attributes

- CUTTING_STRATEGY_FIRST

  Choose the first non-integer row.
- CUTTING_STRATEGY_DEEPEST

  Choose row which generates the deepest cut.
- CUTTING_STRATEGY_ALL

  Always generate all possible cuts.
- PIVOT_ROW_STRATEGY_FIRST

  Choose the first row with negative parameter sign.
- PIVOT_ROW_STRATEGY_MAX_COLUMN

  Choose the row which generates the lexico-maximal pivot column.

10.45.1 Detailed Description

Possible values for PIP problems' control parameters.

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

- PIP_Problem_Control_Parameter_Value.java

10.46 parma_polyhedra_library.PIP_Problem_Status Enum Reference

Possible outcomes of the PIP_Problem solver.

Public Attributes

- UNFEASIBLE_PIP_PROBLEM

  The problem is unsatisfiable.

10.46.1 Detailed Description

Possible outcomes of the PIP_Problem solver.

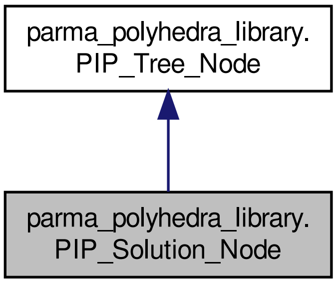
The documentation for this enum was generated from the following file:

- PIP_Problem_Status.java

10.47 parma_polyhedra_library.PIP_Solution_Node Class Reference

A leaf node of the PIP solution tree.

Inheritance diagram for parma_polyhedra_library.PIP_Solution_Node:



Public Member Functions

- native Linear_Expression parametric_values (Variable var)

  Returns the parametric expression of the values of variable `var` in solution node `this`.

Additional Inherited Members

10.47.1 Detailed Description

A leaf node of the PIP solution tree.

10.47.2 Member Function Documentation

native Linear_Expression parma_polyhedra_library.PIP_Solution_Node.parametric_values (Variable *var*) Returns the parametric expression of the values of variable `var` in solution node `this`.

The returned parametric expression will only refer to (problem or artificial) parameters.

Parameters

var	The variable being queried.

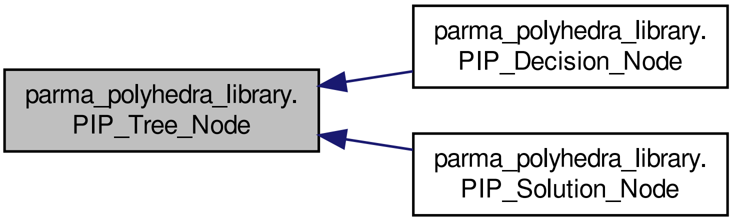
The documentation for this class was generated from the following file:

- PIP_Solution_Node.java

10.48 parma_polyhedra_library.PIP_Tree_Node Class Reference

A node of the PIP solution tree.

Inheritance diagram for parma_polyhedra_library.PIP_Tree_Node:



Public Member Functions

- native PIP_Solution_Node as_solution ()

  Returns the solution node if `this` is a solution node, and 0 otherwise.

- native PIP_Decision_Node as_decision ()

  Returns the decision node if `this` is a decision node, and 0 otherwise.

- native boolean OK ()

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

- native long number_of_artificials ()

  Returns the number of artificial parameters in the PIP_Tree_Node.

- native
  Artificial_Parameter_Sequence artificials ()

  Returns the sequence of (Java) artificial parameters in the PIP_Tree_Node.

- native Constraint_System constraints ()

  Returns the system of parameter constraints controlling the PIP_Tree_Node.

- native String toString ()

  Returns a string representation of `this`.

Additional Inherited Members

10.48.1 Detailed Description

A node of the PIP solution tree.

This is the base class for the nodes of the binary trees representing the solutions of PIP problems. From this one, two classes are derived:

- PIP_Decision_Node, for the internal nodes of the tree;
- PIP_Solution_Node, for the leaves of the tree.

10.48.2 Member Function Documentation

native Constraint_System parma_polyhedra_library.PIP_Tree_Node.constraints ( ) Returns the system of parameter constraints controlling the PIP_Tree_Node.

The indices in the constraints are the same as the original variables and parameters. Coefficients in indices corresponding to variables always are zero.

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

- PIP_Tree_Node.java

10.49 parma_polyhedra_library.Pointset_Powerset_C_Polyhedron Class Reference

A powerset of C_Polyhedron objects.

Inherits parma_polyhedra_library.PPL_Object.

Public Member Functions

Ad Hoc Functions for Pointset_Powerset domains

- native void omega_reduce ()
  Drops from the sequence of disjuncts in `this` *all the non-maximal elements, so that a non-redundant powerset if obtained.*
- native long size ()
  Returns the number of disjuncts.
- native boolean geometrically_covers (Pointset_Powerset_C_Polyhedron y)
  Returns `true` *if and only if* `this` *geometrically covers* `y`*.*
- native boolean geometrically_equals (Pointset_Powerset_C_Polyhedron y)
  Returns `true` *if and only if* `this` *is geometrically equal to* `y`*.*
- native
  Pointset_Powerset_C_Polyhedron_Iterator begin_iterator ()
  Returns an iterator referring to the beginning of the sequence of disjuncts of `this`*.*
- native
  Pointset_Powerset_C_Polyhedron_Iterator end_iterator ()
  Returns an iterator referring to past the end of the sequence of disjuncts of `this`*.*
- native void add_disjunct (C_Polyhedron d)
  Adds to `this` *a copy of disjunct* `d`*.*
- native void drop_disjunct (Pointset_Powerset_C_Polyhedron_Iterator iter)
  Drops from `this` *the disjunct referred by* `iter`*; returns an iterator referring to the disjunct following the dropped one.*
- native void drop_disjuncts (Pointset_Powerset_C_Polyhedron_Iterator first, Pointset_Powerset_C_Polyhedron_Iterator last)
  Drops from `this` *all the disjuncts from* `first` *to* `last` *(excluded).*
- native void pairwise_reduce ()
  Modifies `this` *by (recursively) merging together the pairs of disjuncts whose upper-bound is the same as their set-theoretical union.*

Additional Inherited Members

10.49.1 Detailed Description

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

10.49.2 Member Function Documentation

native long parma_polyhedra_library.Pointset_Powerset_C_Polyhedron.size ( ) Returns the number of disjuncts.

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

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

- Fake_Class_for_Doxygen.java

10.50 parma_polyhedra_library.Pointset_Powerset_C_Polyhedron_Iterator Class Reference

An iterator class for the disjuncts of a Pointset_Powerset_C_Polyhedron.

Inherits parma_polyhedra_library.PPL_Object.

Public Member Functions

- Pointset_Powerset_C_Polyhedron_Iterator (Pointset_Powerset_C_Polyhedron_Iterator y)

  Builds a copy of iterator `y`*.*
- native boolean equals (Pointset_Powerset_C_Polyhedron_Iterator itr)

  Returns `true` *if and only if* `this` *and* `itr` *are equal.*
- native void next ()

  Modifies `this` *so that it refers to the next disjunct.*
- native void prev ()

  Modifies `this` *so that it refers to the previous disjunct.*
- native C_Polyhedron get_disjunct ()

  Returns the disjunct referenced by `this`*.*
- native void free ()

  Releases resources and resets `this` *to a null reference.*

Protected Member Functions

- native void finalize ()

  Releases the resources managed by `this`*.*

10.50.1 Detailed Description

An iterator class for the disjuncts of a Pointset_Powerset_C_Polyhedron.

10.50.2 Member Function Documentation

native C_Polyhedron parma_polyhedra_library.Pointset_Powerset_C_Polyhedron_Iterator.get_disjunct ( ) Returns the disjunct referenced by `this`.

Warning

> On exit, the C_Polyhedron disjunct is still owned by the powerset object: any function call on the owning powerset object may invalidate it. Moreover, the disjunct is meant to be immutable and should not be modified in any way (its resources will be released when deleting the owning powerset). If really needed, the disjunct may be copied into a new object, which will be under control of the user.

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

- Fake_Class_for_Doxygen.java

10.51 parma_polyhedra_library.Poly_Con_Relation Class Reference

The relation between a polyhedron and a constraint.

Public Member Functions

- Poly_Con_Relation (int val)

  Constructs from a integer value.
- boolean implies (Poly_Con_Relation y)

  True if and only if `this` *implies* `y`*.*

Static Public Member Functions

- static Poly_Con_Relation nothing ()

  The assertion that says nothing.
- static Poly_Con_Relation is_disjoint ()

  The polyhedron and the set of points satisfying the constraint are disjoint.
- static Poly_Con_Relation strictly_intersects ()

  The polyhedron intersects the set of points satisfying the constraint, but it is not included in it.
- static Poly_Con_Relation is_included ()

  The polyhedron is included in the set of points satisfying the constraint.
- static Poly_Con_Relation saturates ()

  The polyhedron is included in the set of points saturating the constraint.

10.51.1 Detailed Description

The relation between a polyhedron and a constraint.

This class implements conjunctions of assertions on the relation between a polyhedron and a constraint.

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

- Poly_Con_Relation.java

10.52 parma_polyhedra_library.Poly_Gen_Relation Class Reference

The relation between a polyhedron and a generator.

Public Member Functions

- Poly_Gen_Relation (int val)

  Constructs from a integer value.
- boolean implies (Poly_Gen_Relation y)

  True if and only if `this` *implies* `y`*.*

Static Public Member Functions

- static Poly_Gen_Relation nothing ()

  The assertion that says nothing.
- static Poly_Gen_Relation subsumes ()

  Adding the generator would not change the polyhedron.

10.52.1 Detailed Description

The relation between a polyhedron and a generator.

This class implements conjunctions of assertions on the relation between a polyhedron and a generator.

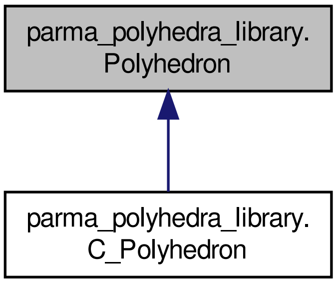
The documentation for this class was generated from the following file:

- Poly_Gen_Relation.java

10.53 parma_polyhedra_library.Polyhedron Class Reference

The Java base class for (C and NNC) convex polyhedra.

Inheritance diagram for parma_polyhedra_library.Polyhedron:



Public Member Functions

Member Functions that Do Not Modify the Polyhedron

- native long space_dimension ()

  Returns the dimension of the vector space enclosing `this`*.*
- native long affine_dimension ()

  Returns 0*, if* `this` *is empty; otherwise, returns the affine dimension of* `this`*.*
- native Constraint_System constraints ()

  Returns the system of constraints.
- native Congruence_System congruences ()

  Returns a system of (equality) congruences satisfied by `this`*.*
- native Constraint_System minimized_constraints ()

  Returns the system of constraints, with no redundant constraint.
- native Congruence_System minimized_congruences ()

  Returns a system of (equality) congruences satisfied by `this`*, with no redundant congruences and having the same affine dimension as* `this`*.*
- native boolean is_empty ()

  Returns `true` *if and only if* `this` *is an empty polyhedron.*
- native boolean is_universe ()

  Returns `true` *if and only if* `this` *is a universe polyhedron.*
- native boolean is_bounded ()

  Returns `true` *if and only if* `this` *is a bounded polyhedron.*
- native boolean is_discrete ()

Returns `true` *if and only if* `this` *is discrete.*

- native boolean is_topologically_closed ()

  Returns `true` *if and only if* `this` *is a topologically closed subset of the vector space.*
- native boolean contains_integer_point ()

  Returns `true` *if and only if* `this` *contains at least one integer point.*
- native boolean constrains (Variable var)

  Returns `true` *if and only if* `var` *is constrained in* `this`.
- native boolean bounds_from_above (Linear_Expression expr)

  Returns `true` *if and only if* `expr` *is bounded from above in* `this`.
- native boolean bounds_from_below (Linear_Expression expr)

  Returns `true` *if and only if* `expr` *is bounded from below in* `this`.
- native boolean maximize (Linear_Expression expr, Coefficient sup_n, Coefficient sup_d, By_Reference< Boolean > maximum)

  Returns `true` *if and only if* `this` *is not empty and* `expr` *is bounded from above in* `this`, *in which case the supremum value is computed.*
- native boolean minimize (Linear_Expression expr, Coefficient inf_n, Coefficient inf_d, By_Reference< Boolean > minimum)

  Returns `true` *if and only if* `this` *is not empty and* `expr` *is bounded from below in* `this`, *in which case the infimum value is computed.*
- native boolean maximize (Linear_Expression expr, Coefficient sup_n, Coefficient sup_d, By_Reference< Boolean > maximum, Generator g)

  Returns `true` *if and only if* `this` *is not empty and* `expr` *is bounded from above in* `this`, *in which case the supremum value and a point where* `expr` *reaches it are computed.*
- native boolean minimize (Linear_Expression expr, Coefficient inf_n, Coefficient inf_d, By_Reference< Boolean > minimum, Generator g)

  Returns `true` *if and only if* `this` *is not empty and* `expr` *is bounded from below in* `this`, *in which case the infimum value and a point where* `expr` *reaches it are computed.*
- native Poly_Con_Relation relation_with (Constraint c)

  Returns the relations holding between the polyhedron `this` *and the constraint* `c`.
- native Poly_Gen_Relation relation_with (Generator c)

  Returns the relations holding between the polyhedron `this` *and the generator* `g`.
- native Poly_Con_Relation relation_with (Congruence c)

  Returns the relations holding between the polyhedron `this` *and the congruence* `c`.
- native boolean contains (Polyhedron y)

  Returns `true` *if and only if* `this` *contains* `y`.
- native boolean strictly_contains (Polyhedron y)

  Returns `true` *if and only if* `this` *strictly contains* `y`.
- native boolean is_disjoint_from (Polyhedron y)

  Returns `true` *if and only if* `this` *and* `y` *are disjoint.*
- native boolean equals (Polyhedron y)

  Returns `true` *if and only if* `this` *and* `y` *are equal.*
- boolean equals (Object y)

  Returns `true` *if and only if* `this` *and* `y` *are equal.*
- native int hashCode ()

  Returns a hash code for `this`.
- native long external_memory_in_bytes ()

  Returns the size in bytes of the memory managed by `this`.
- native long total_memory_in_bytes ()

  Returns the total size in bytes of the memory occupied by `this`.
- native String toString ()

  Returns a string representing `this`.
- native String ascii_dump ()

  Returns a string containing a low-level representation of `this`.
- native boolean OK ()

Checks if all the invariants are satisfied.

Space Dimension Preserving Member Functions that May Modify the Polyhedron

- native void add_constraint (Constraint c)

  Adds a copy of constraint `c` to the system of constraints of `this` (without minimizing the result).
- native void add_congruence (Congruence cg)

  Adds a copy of congruence `cg` to `this`, if `cg` can be exactly represented by a polyhedron.
- native void add_constraints (Constraint_System cs)

  Adds a copy of the constraints in `cs` to the system of constraints of `this` (without minimizing the result).
- native void add_congruences (Congruence_System cgs)

  Adds a copy of the congruences in `cgs` to `this`, if all the congruences can be exactly represented by a polyhedron.
- native void refine_with_constraint (Constraint c)

  Uses a copy of constraint `c` to refine `this`.
- native void refine_with_congruence (Congruence cg)

  Uses a copy of congruence `cg` to refine `this`.
- native void refine_with_constraints (Constraint_System cs)

  Uses a copy of the constraints in `cs` to refine `this`.
- native void refine_with_congruences (Congruence_System cgs)

  Uses a copy of the congruences in `cgs` to refine `this`.
- native void intersection_assign (Polyhedron y)

  Assigns to `this` the intersection of `this` and `y`. The result is not guaranteed to be minimized.
- native void upper_bound_assign (Polyhedron y)

  Assigns to `this` the upper bound of `this` and `y`.
- native void difference_assign (Polyhedron y)

  Assigns to `this` the poly-difference of `this` and `y`. The result is not guaranteed to be minimized.
- native void time_elapse_assign (Polyhedron y)

  Assigns to `this` the result of computing the time-elapse between `this` and `y`.
- native void topological_closure_assign ()

  Assigns to `this` its topological closure.
- native boolean simplify_using_context_assign (Polyhedron y)

  Assigns to `this` a meet-preserving simplification of `this` with respect to `y`. If `false` is returned, then the intersection is empty.
- native void affine_image (Variable var, Linear_Expression expr, Coefficient denominator)

  Assigns to `this` the affine image of `this` under the function mapping variable `var` to the affine expression specified by `expr` and `denominator`.
- native void affine_preimage (Variable var, Linear_Expression expr, Coefficient denominator)

  Assigns to `this` the affine preimage of `this` under the function mapping variable `var` to the affine expression specified by `expr` and `denominator`.
- native void bounded_affine_image (Variable var, Linear_Expression lb_expr, Linear_Expression ub_expr, Coefficient denominator)

  Assigns to `this` the image of `this` with respect to the bounded affine relation $\frac{\mathrm{lb\_expr}}{\mathrm{denominator}} \leq \mathrm{var}' \leq \frac{\mathrm{ub\_expr}}{\mathrm{denominator}}$.
- native void bounded_affine_preimage (Variable var, Linear_Expression lb_expr, Linear_Expression ub_expr, Coefficient denominator)

  Assigns to `this` the preimage of `this` with respect to the bounded affine relation $\frac{\mathrm{lb\_expr}}{\mathrm{denominator}} \leq \mathrm{var}' \leq \frac{\mathrm{ub\_expr}}{\mathrm{denominator}}$.
- native void generalized_affine_image (Variable var, Relation_Symbol relsym, Linear_Expression expr, Coefficient denominator)

  Assigns to `this` the image of `this` with respect to the generalized affine relation $\mathrm{var}' \bowtie \frac{\mathrm{expr}}{\mathrm{denominator}}$, *where* $\bowtie$ *is the relation symbol encoded by `relsym`.*
- native void generalized_affine_preimage (Variable var, Relation_Symbol relsym, Linear_Expression expr, Coefficient denominator)

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

- native void generalized_affine_image (Linear_Expression lhs, Relation_Symbol relsym, Linear_Expression rhs)

  Assigns to `this` *the image of* `this` *with respect to the generalized affine relation* $\mathrm{lhs}' \bowtie \mathrm{rhs}$, *where* $\bowtie$ *is the relation symbol encoded by* `relsym`*.*
- native void generalized_affine_preimage (Linear_Expression lhs, Relation_Symbol relsym, Linear_Expression rhs)

  Assigns to `this` *the preimage of* `this` *with respect to the generalized affine relation* $\mathrm{lhs}' \bowtie \mathrm{rhs}$, *where* $\bowtie$ *is the relation symbol encoded by* `relsym`*.*
- native void unconstrain_space_dimension (Variable var)

  Computes the cylindrification of `this` *with respect to space dimension* `var`*, assigning the result to* `this`*.*
- native void unconstrain_space_dimensions (Variables_Set vars)

  Computes the cylindrification of `this` *with respect to the set of space dimensions* `vars`*, assigning the result to* `this`*.*
- native void widening_assign (Polyhedron y, By_Reference< Integer > tp)

  Assigns to `this` *the result of computing the H79-widening between* `this` *and* `y`*.*

Member Functions that May Modify the Dimension of the Vector Space

- native void swap (Polyhedron y)

  Swaps `this` *with polyhedron* `y`*. (*`this` *and* `y` *can be dimension-incompatible.)*
- native void add_space_dimensions_and_embed (long m)

  Adds `m` *new space dimensions and embeds the old polyhedron in the new vector space.*
- native void add_space_dimensions_and_project (long m)

  Adds `m` *new space dimensions to the polyhedron and does not embed it in the new vector space.*
- native void concatenate_assign (Polyhedron y)

  Assigns to `this` *the concatenation of* `this` *and* `y`*, taken in this order.*
- native void remove_space_dimensions (Variables_Set vars)

  Removes all the specified dimensions from the vector space.
- native void remove_higher_space_dimensions (long new_dimension)

  Removes the higher dimensions of the vector space so that the resulting space will have dimension `new_dimension`*.*
- native void expand_space_dimension (Variable var, long m)

  Creates `m` *copies of the space dimension corresponding to* `var`*.*
- native void fold_space_dimensions (Variables_Set vars, Variable dest)

  Folds the space dimensions in `vars` *into* `dest`*.*
- native void map_space_dimensions (Partial_Function pfunc)

  Remaps the dimensions of the vector space according to a partial function.

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.

- native Generator_System generators ()

  Returns the system of generators.
- native Generator_System minimized_generators ()

  Returns the system of generators, with no redundant generator.
- native void add_generator (Generator g)

  Adds a copy of generator `g` *to the system of generators of* `this` *(without minimizing the result).*
- native void add_generators (Generator_System gs)

  Adds a copy of the generators in `gs` *to the system of generators of* `this` *(without minimizing the result).*
- native void poly_hull_assign (Polyhedron y)

  Same as upper_bound_assign.

- native void poly_difference_assign (Polyhedron y)

  Same as difference_assign.
- native void BHRZ03_widening_assign (Polyhedron y, By_Reference< Integer > tp)

  Assigns to `this` *the result of computing the BHRZ03-widening between* `this` *and* `y`*.*
- native void H79_widening_assign (Polyhedron y, By_Reference< Integer > tp)

  Assigns to `this` *the result of computing the H79-widening between* `this` *and* `y`*.*
- native void limited_BHRZ03_extrapolation_assign (Polyhedron y, Constraint_System cs, By_Reference< Integer > tp)

  Improves the result of the BHRZ03-widening computation by also enforcing those constraints in `cs` *that are satisfied by all the points of* `this`*.*
- native void limited_H79_extrapolation_assign (Polyhedron y, Constraint_System cs, By_Reference< Integer > tp)

  Improves the result of the H79-widening computation by also enforcing those constraints in `cs` *that are satisfied by all the points of* `this`*.*
- native void bounded_BHRZ03_extrapolation_assign (Polyhedron y, Constraint_System cs, By_Reference< Integer > tp)

  Improves the result of the BHRZ03-widening computation by also enforcing those constraints in `cs` *that are satisfied by all the points of* `this`*, plus all the constraints of the form* $\pm x \leq r$ *and* $\pm x < r$*, with* $r \in \mathbb{Q}$*, that are satisfied by all the points of* `this`*.*
- native void bounded_H79_extrapolation_assign (Polyhedron y, Constraint_System cs, By_Reference< Integer > tp)

  Improves the result of the H79-widening computation by also enforcing those constraints in `cs` *that are satisfied by all the points of* `this`*, plus all the constraints of the form* $\pm x \leq r$ *and* $\pm x < r$*, with* $r \in \mathbb{Q}$*, that are satisfied by all the points of* `this`*.*

Additional Inherited Members

10.53.1 Detailed Description

The Java base class for (C and NNC) convex polyhedra.

The base class Polyhedron provides declarations for most of the methods common to classes C_Polyhedron and NNC_Polyhedron. Note that the user should always use the derived classes. Moreover, C and NNC polyhedra can not be freely interchanged: as specified in the main manual, most library functions require their arguments to be topologically compatible.

10.53.2 Member Function Documentation

native boolean parma_polyhedra_library.Polyhedron.constrains (Variable *var*) Returns `true` if and only if `var` is constrained in `this`.

Exceptions

Invalid_Argument_Exception	Thrown if `var` is not a space dimension of `this`.

native boolean parma_polyhedra_library.Polyhedron.bounds_from_above (Linear_Expression *expr*) Returns `true` if and only if `expr` is bounded from above in `this`.

Exceptions

Invalid_Argument_Exception	Thrown if `expr` and `this` are dimension-incompatible.

native boolean parma_polyhedra_library.Polyhedron.bounds_from_below (Linear_Expression *expr*) Returns `true` if and only if `expr` is bounded from below in `this`.

Exceptions

Invalid_Argument_-Exception	Thrown if `expr` and `this` are dimension-incompatible.

native boolean parma_polyhedra_library.Polyhedron.maximize (Linear_Expression *expr*, Coefficient *sup_n*, Coefficient *sup_d*, By_Reference< Boolean > *maximum*) Returns `true` if and only if `this` is not empty and `expr` is bounded from above in `this`, in which case the supremum value is computed.

Parameters

expr	The linear expression to be maximized subject to `this`;
sup_n	The numerator of the supremum value;
sup_d	The denominator of the supremum value;
maximum	`true` if and only if the supremum is also the maximum value.

Exceptions

Invalid_Argument_-Exception	Thrown if `expr` and `this` are dimension-incompatible.

If `this` is empty or `expr` is not bounded from above, `false` is returned and `sup_n`, `sup_d` and `maximum` are left untouched.

native boolean parma_polyhedra_library.Polyhedron.minimize (Linear_Expression *expr*, Coefficient *inf_n*, Coefficient *inf_d*, By_Reference< Boolean > *minimum*) Returns `true` if and only if `this` is not empty and `expr` is bounded from below in `this`, in which case the infimum value is computed.

Parameters

expr	The linear expression to be minimized subject to `this`;
inf_n	The numerator of the infimum value;
inf_d	The denominator of the infimum value;
minimum	`true` if and only if the infimum is also the minimum value.

Exceptions

Invalid_Argument_-Exception	Thrown if `expr` and `this` are dimension-incompatible.

If `this` is empty or `expr` is not bounded from below, `false` is returned and `inf_n`, `inf_d` and `minimum` are left untouched.

native boolean parma_polyhedra_library.Polyhedron.maximize (Linear_Expression *expr*, Coefficient *sup_n*, Coefficient *sup_d*, By_Reference< Boolean > *maximum*, Generator *g*) Returns `true` if and only if `this` is not empty and `expr` is bounded from above in `this`, in which case the supremum value and a point where `expr` reaches it are computed.

Parameters

expr	The linear expression to be maximized subject to `this`;
sup_n	The numerator of the supremum value;
sup_d	The denominator of the supremum value;

maximum	`true` if and only if the supremum is also the maximum value;
g	When maximization succeeds, will be assigned the point or closure point where `expr` reaches its supremum value.

Exceptions

Invalid_Argument_-Exception	Thrown if `expr` and `this` are dimension-incompatible.

If `this` is empty or `expr` is not bounded from above, `false` is returned and `sup_n`, `sup_d`, `maximum` and `g` are left untouched.

native boolean parma_polyhedra_library.Polyhedron.minimize (Linear_Expression *expr*, Coefficient *inf_n*, Coefficient *inf_d*, By_Reference< Boolean > *minimum*, Generator *g*) Returns `true` if and only if `this` is not empty and `expr` is bounded from below in `this`, in which case the infimum value and a point where `expr` reaches it are computed.

Parameters

expr	The linear expression to be minimized subject to `this`;
inf_n	The numerator of the infimum value;
inf_d	The denominator of the infimum value;
minimum	`true` if and only if the infimum is also the minimum value;
g	When minimization succeeds, will be assigned a point or closure point where `expr` reaches its infimum value.

Exceptions

Invalid_Argument_-Exception	Thrown if `expr` and `this` are dimension-incompatible.

If `this` is empty or `expr` is not bounded from below, `false` is returned and `inf_n`, `inf_d`, `minimum` and `g` are left untouched.

native Poly_Con_Relation parma_polyhedra_library.Polyhedron.relation_with (Constraint *c*) Returns the relations holding between the polyhedron `this` and the constraint `c`.

Exceptions

Invalid_Argument_-Exception	Thrown if `this` and constraint `c` are dimension-incompatible.

native Poly_Gen_Relation parma_polyhedra_library.Polyhedron.relation_with (Generator *c*) Returns the relations holding between the polyhedron `this` and the generator `g`.

Exceptions

Invalid_Argument_-Exception	Thrown if `this` and generator `g` are dimension-incompatible.

native Poly_Con_Relation parma_polyhedra_library.Polyhedron.relation_with (Congruence *c*) Returns the relations holding between the polyhedron `this` and the congruence `c`.

Exceptions

Invalid_Argument_-Exception	Thrown if this and congruence c are dimension-incompatible.

native boolean parma_polyhedra_library.Polyhedron.contains (Polyhedron *y*) Returns true if and only if this contains y.

Exceptions

Invalid_Argument_-Exception	Thrown if this and y are topology-incompatible or dimension-incompatible.

native boolean parma_polyhedra_library.Polyhedron.strictly_contains (Polyhedron *y*) Returns true if and only if this strictly contains y.

Exceptions

Invalid_Argument_-Exception	Thrown if this and y are topology-incompatible or dimension-incompatible.

native boolean parma_polyhedra_library.Polyhedron.is_disjoint_from (Polyhedron *y*) Returns true if and only if this and y are disjoint.

Exceptions

Invalid_Argument_-Exception	Thrown if x and y are topology-incompatible or dimension-incompatible.

native int parma_polyhedra_library.Polyhedron.hashCode ( ) Returns a hash code for this.

If x and y are such that x == y, then x.hash_code() == y.hash_code().

native String parma_polyhedra_library.Polyhedron.ascii_dump ( ) Returns a string containing a low-level representation of this.

Useful for debugging purposes.

native void parma_polyhedra_library.Polyhedron.add_constraint (Constraint *c*) Adds a copy of constraint c to the system of constraints of this (without minimizing the result).

Parameters

c	The constraint that will be added to the system of constraints of this.

Exceptions

Invalid_Argument_-Exception	Thrown if this and constraint c are topology-incompatible or dimension-incompatible.

native void parma_polyhedra_library.Polyhedron.add_congruence (Congruence *cg*) Adds a copy of congruence cg to this, if cg can be exactly represented by a polyhedron.

Exceptions

Invalid_Argument_-Exception	Thrown if this and congruence cg are dimension-incompatible, of if cg is a proper congruence which is neither a tautology, nor a contradiction.

native void parma_polyhedra_library.Polyhedron.add_constraints (Constraint_System *cs*) Adds a copy of the constraints in cs to the system of constraints of this (without minimizing the result).

Parameters

cs	Contains the constraints that will be added to the system of constraints of `this`.

Exceptions

Invalid_Argument_-Exception	Thrown if `this` and `cs` are topology-incompatible or dimension-incompatible.

native void parma_polyhedra_library.Polyhedron.add_congruences (Congruence_System *cgs*) Adds a copy of the congruences in `cgs` to `this`, if all the congruences can be exactly represented by a polyhedron.

Parameters

cgs	The congruences to be added.

Exceptions

Invalid_Argument_-Exception	Thrown if `this` and `cgs` are dimension-incompatible, of if there exists in `cgs` a proper congruence which is neither a tautology, nor a contradiction.

native void parma_polyhedra_library.Polyhedron.refine_with_constraint (Constraint *c*) Uses a copy of constraint `c` to refine `this`.

Exceptions

Invalid_Argument_-Exception	Thrown if `this` and constraint `c` are dimension-incompatible.

native void parma_polyhedra_library.Polyhedron.refine_with_congruence (Congruence *cg*) Uses a copy of congruence `cg` to refine `this`.

Exceptions

Invalid_Argument_-Exception	Thrown if `this` and congruence `cg` are dimension-incompatible.

native void parma_polyhedra_library.Polyhedron.refine_with_constraints (Constraint_System *cs*) Uses a copy of the constraints in `cs` to refine `this`.

Parameters

cs	Contains the constraints used to refine the system of constraints of `this`.

Exceptions

Invalid_Argument_-Exception	Thrown if `this` and `cs` are dimension-incompatible.

native void parma_polyhedra_library.Polyhedron.refine_with_congruences (Congruence_System *cgs*) Uses a copy of the congruences in `cgs` to refine `this`.

Parameters

cgs	Contains the congruences used to refine the system of constraints of `this`.

Exceptions

Invalid_Argument_-Exception	Thrown if `this` and `cgs` are dimension-incompatible.

native void parma_polyhedra_library.Polyhedron.intersection_assign (Polyhedron *y*) Assigns to `this` the intersection of `this` and `y`. The result is not guaranteed to be minimized.
Exceptions

Invalid_Argument_-Exception	Thrown if `this` and `y` are topology-incompatible or dimension-incompatible.

native void parma_polyhedra_library.Polyhedron.upper_bound_assign (Polyhedron *y*) Assigns to `this` the upper bound of `this` and `y`.
Exceptions

Invalid_Argument_-Exception	Thrown if `this` and `y` are topology-incompatible or dimension-incompatible.

native void parma_polyhedra_library.Polyhedron.difference_assign (Polyhedron *y*) Assigns to `this` the *poly-difference* of `this` and `y`. The result is not guaranteed to be minimized.
Exceptions

Invalid_Argument_-Exception	Thrown if `this` and `y` are topology-incompatible or dimension-incompatible.

native void parma_polyhedra_library.Polyhedron.time_elapse_assign (Polyhedron *y*) Assigns to `this` the result of computing the *time-elapse* between `this` and `y`.
Exceptions

Invalid_Argument_-Exception	Thrown if `this` and `y` are topology-incompatible or dimension-incompatible.

native boolean parma_polyhedra_library.Polyhedron.simplify_using_context_assign (Polyhedron *y*) Assigns to `this` a *meet-preserving simplification* of `this` with respect to `y`. If `false` is returned, then the intersection is empty.
Exceptions

Invalid_Argument_-Exception	Thrown if `this` and `y` are topology-incompatible or dimension-incompatible.

native void parma_polyhedra_library.Polyhedron.affine_image (Variable *var*, Linear_Expression *expr*, Coefficient *denominator*) Assigns to `this` the *affine image* of `this` under the function mapping variable `var` to the affine expression specified by `expr` and `denominator`.

Parameters

var	The variable to which the affine expression is assigned;
expr	The numerator of the affine expression;
denominator	The denominator of the affine expression (optional argument with default value 1).

Exceptions

Invalid_Argument_-Exception	Thrown if `denominator` is zero or if `expr` and `this` are dimension-incompatible or if `var` is not a space dimension of `this`.

native void parma_polyhedra_library.Polyhedron.affine_preimage (Variable *var*, Linear_Expression *expr*, Coefficient *denominator*) Assigns to `this` the *affine preimage* of `this` under the function mapping variable `var` to the affine expression specified by `expr` and `denominator`.

Parameters

var	The variable to which the affine expression is substituted;
expr	The numerator of the affine expression;
denominator	The denominator of the affine expression (optional argument with default value 1).

Exceptions

Invalid_Argument_-Exception	Thrown if `denominator` is zero or if `expr` and `this` are dimension-incompatible or if `var` is not a space dimension of `this`.

native void parma_polyhedra_library.Polyhedron.bounded_affine_image (Variable *var*, Linear_-Expression *lb_expr*, Linear_Expression *ub_expr*, Coefficient *denominator*) Assigns to `this` the image of `this` with respect to the *bounded affine relation* $\frac{\mathrm{lb\_expr}}{\mathrm{denominator}} \leq \mathrm{var}' \leq \frac{\mathrm{ub\_expr}}{\mathrm{denominator}}$.

Parameters

var	The variable updated by the affine relation;
lb_expr	The numerator of the lower bounding affine expression;
ub_expr	The numerator of the upper bounding affine expression;
denominator	The (common) denominator for the lower and upper bounding affine expressions (optional argument with default value 1).

Exceptions

Invalid_Argument_-Exception	Thrown if `denominator` is zero or if `lb_expr` (resp., `ub_expr`) and `this` are dimension-incompatible or if `var` is not a space dimension of `this`.

native void parma_polyhedra_library.Polyhedron.bounded_affine_preimage (Variable *var*, Linear-_Expression *lb_expr*, Linear_Expression *ub_expr*, Coefficient *denominator*) Assigns to `this` the preimage of `this` with respect to the *bounded affine relation* $\frac{\mathrm{lb\_expr}}{\mathrm{denominator}} \leq \mathrm{var}' \leq \frac{\mathrm{ub\_expr}}{\mathrm{denominator}}$.

Parameters

var	The variable updated by the affine relation;
lb_expr	The numerator of the lower bounding affine expression;
ub_expr	The numerator of the upper bounding affine expression;

denominator	The (common) denominator for the lower and upper bounding affine expressions (optional argument with default value 1).

Exceptions

Invalid_Argument_-Exception	Thrown if `denominator` is zero or if `lb_expr` (resp., `ub_expr`) and `this` are dimension-incompatible or if `var` is not a space dimension of `this`.

native void parma_polyhedra_library.Polyhedron.generalized_affine_image (Variable *var,* Relation_Symbol *relsym,* Linear_Expression *expr,* Coefficient *denominator*) Assigns to `this` the image of `this` with respect to the *generalized affine relation* $\mathrm{var}' \bowtie \frac{\mathrm{expr}}{\mathrm{denominator}}$, where $\bowtie$ is the relation symbol encoded by `relsym`.

Parameters

var	The left hand side variable of the generalized affine relation;
relsym	The relation symbol;
expr	The numerator of the right hand side affine expression;
denominator	The denominator of the right hand side affine expression (optional argument with default value 1).

Exceptions

Invalid_Argument_-Exception	Thrown if `denominator` is zero or if `expr` and `this` are dimension-incompatible or if `var` is not a space dimension of `this` or if `this` is a C_Polyhedron and `relsym` is a strict relation symbol.

native void parma_polyhedra_library.Polyhedron.generalized_affine_preimage (Variable *var,* Relation_Symbol *relsym,* Linear_Expression *expr,* Coefficient *denominator*) Assigns to `this` the preimage of `this` with respect to the *generalized affine relation* $\mathrm{var}' \bowtie \frac{\mathrm{expr}}{\mathrm{denominator}}$, where $\bowtie$ is the relation symbol encoded by `relsym`.

Parameters

var	The left hand side variable of the generalized affine relation;
relsym	The relation symbol;
expr	The numerator of the right hand side affine expression;
denominator	The denominator of the right hand side affine expression (optional argument with default value 1).

Exceptions

Invalid_Argument_-Exception	Thrown if `denominator` is zero or if `expr` and `this` are dimension-incompatible or if `var` is not a space dimension of `this` or if `this` is a C_Polyhedron and `relsym` is a strict relation symbol.

native void parma_polyhedra_library.Polyhedron.generalized_affine_image (Linear_Expression *lhs,* Relation_Symbol *relsym,* Linear_Expression *rhs*) Assigns to `this` the image of `this` with respect to the *generalized affine relation* $\mathrm{lhs}' \bowtie \mathrm{rhs}$, where $\bowtie$ is the relation symbol encoded by `relsym`.

Parameters

lhs	The left hand side affine expression;
relsym	The relation symbol;
rhs	The right hand side affine expression.

Exceptions

Invalid_Argument_-Exception	Thrown if `this` is dimension-incompatible with `lhs` or `rhs` or if `this` is a C_Polyhedron and `relsym` is a strict relation symbol.

native void parma_polyhedra_library.Polyhedron.generalized_affine_preimage (Linear_Expression *lhs*, Relation_Symbol *relsym*, Linear_Expression *rhs*) Assigns to `this` the preimage of `this` with respect to the *generalized affine relation* $\mathrm{lhs}' \bowtie \mathrm{rhs}$, where $\bowtie$ is the relation symbol encoded by `relsym`.

Parameters

lhs	The left hand side affine expression;
relsym	The relation symbol;
rhs	The right hand side affine expression.

Exceptions

Invalid_Argument_-Exception	Thrown if `this` is dimension-incompatible with `lhs` or `rhs` or if `this` is a C_Polyhedron and `relsym` is a strict relation symbol.

native void parma_polyhedra_library.Polyhedron.unconstrain_space_dimension (Variable *var*) Computes the *cylindrification* of `this` with respect to space dimension `var`, assigning the result to `this`.

Parameters

var	The space dimension that will be unconstrained.

Exceptions

Invalid_Argument_-Exception	Thrown if `var` is not a space dimension of `this`.

native void parma_polyhedra_library.Polyhedron.unconstrain_space_dimensions (Variables_Set *vars*) Computes the *cylindrification* of `this` with respect to the set of space dimensions `vars`, assigning the result to `this`.

Parameters

vars	The set of space dimension that will be unconstrained.

Exceptions

Invalid_Argument_-Exception	Thrown if `this` is dimension-incompatible with one of the Variable objects contained in `vars`.

native void parma_polyhedra_library.Polyhedron.widening_assign (Polyhedron *y*, By_Reference< Integer > *tp*) Assigns to `this` the result of computing the *H79-widening* between `this` and `y`.

Parameters

y	A polyhedron that *must* be contained in `this`;

tp	A reference to an unsigned variable storing the number of available tokens (to be used when applying the *widening with tokens* delay technique).

Exceptions

Invalid_Argument_-Exception	Thrown if `this` and `y` are topology-incompatible or dimension-incompatible.

native void parma_polyhedra_library.Polyhedron.swap (Polyhedron *y*) Swaps `this` with polyhedron `y`. (`this` and `y` can be dimension-incompatible.)

Exceptions

Invalid_Argument_-Exception	Thrown if `x` and `y` are topology-incompatible.

native void parma_polyhedra_library.Polyhedron.add_space_dimensions_and_embed (long *m*) Adds `m` new space dimensions and embeds the old polyhedron in the new vector space.

Parameters

m	The number of dimensions to add.

Exceptions

Length_Error_Exception	Thrown if adding `m` new space dimensions would cause the vector space to exceed dimension `max_space_dimension()`.

native void parma_polyhedra_library.Polyhedron.add_space_dimensions_and_project (long *m*) Adds `m` new space dimensions to the polyhedron and does not embed it in the new vector space.

Parameters

m	The number of space dimensions to add.

Exceptions

Length_Error_Exception	Thrown if adding `m` new space dimensions would cause the vector space to exceed dimension `max_space_dimension()`.

native void parma_polyhedra_library.Polyhedron.concatenate_assign (Polyhedron *y*) Assigns to `this` the *concatenation* of `this` and `y`, taken in this order.

Exceptions

Invalid_Argument_-Exception	Thrown if `this` and `y` are topology-incompatible.
Length_Error_Exception	Thrown if the concatenation would cause the vector space to exceed dimension `max_space_dimension()`.

native void parma_polyhedra_library.Polyhedron.remove_space_dimensions (Variables_Set *vars*) Removes all the specified dimensions from the vector space.

Parameters

vars	The set of Variable objects corresponding to the space dimensions to be removed.

Exceptions

Invalid_Argument_-Exception	Thrown if `this` is dimension-incompatible with one of the Variable objects contained in `vars`.

native void parma_polyhedra_library.Polyhedron.remove_higher_space_dimensions (long *new_dimension*) Removes the higher dimensions of the vector space so that the resulting space will have dimension `new_dimension`.

Exceptions

Invalid_Argument_-Exception	Thrown if `new_dimensions` is greater than the space dimension of `this`.

native void parma_polyhedra_library.Polyhedron.expand_space_dimension (Variable *var*, long *m*) Creates `m` copies of the space dimension corresponding to `var`.

Parameters

var	The variable corresponding to the space dimension to be replicated;
m	The number of replicas to be created.

Exceptions

Invalid_Argument_-Exception	Thrown if `var` does not correspond to a dimension of the vector space.
Length_Error_Exception	Thrown if adding `m` new space dimensions would cause the vector space to exceed dimension `max_space_dimension()`.

native void parma_polyhedra_library.Polyhedron.fold_space_dimensions (Variables_Set *vars*, Variable *dest*) Folds the space dimensions in `vars` into `dest`.

Parameters

vars	The set of Variable objects corresponding to the space dimensions to be folded;
dest	The variable corresponding to the space dimension that is the destination of the folding operation.

Exceptions

Invalid_Argument_-Exception	Thrown if `this` is dimension-incompatible with `dest` or with one of the Variable objects contained in `vars`. Also thrown if `dest` is contained in `vars`.

native void parma_polyhedra_library.Polyhedron.map_space_dimensions (Partial_Function *pfunc*) Remaps the dimensions of the vector space according to a *partial function*.

Parameters

pfunc	The partial function specifying the destiny of each space dimension.

native void parma_polyhedra_library.Polyhedron.add_generator (Generator *g*) Adds a copy of generator `g` to the system of generators of `this` (without minimizing the result).

Exceptions

Invalid_Argument_-Exception	Thrown if `this` and generator `g` are topology-incompatible or dimension-incompatible, or if `this` is an empty polyhedron and `g` is not a point.

native void parma_polyhedra_library.Polyhedron.add_generators (Generator_System *gs*) Adds a copy of the generators in `gs` to the system of generators of `this` (without minimizing the result).
Parameters

gs	Contains the generators that will be added to the system of generators of `this`.

Exceptions

Invalid_Argument_-Exception	Thrown if `this` and `gs` are topology-incompatible or dimension-incompatible, or if `this` is empty and the system of generators `gs` is not empty, but has no points.

native void parma_polyhedra_library.Polyhedron.BHRZ03_widening_assign (Polyhedron *y*, By_-Reference< Integer > *tp*) Assigns to `this` the result of computing the *BHRZ03-widening* between `this` and `y`.
Parameters

y	A polyhedron that *must* be contained in `this`;
tp	A reference to an unsigned variable storing the number of available tokens (to be used when applying the *widening with tokens* delay technique).

Exceptions

Invalid_Argument_-Exception	Thrown if `this` and `y` are topology-incompatible or dimension-incompatible.

native void parma_polyhedra_library.Polyhedron.H79_widening_assign (Polyhedron *y*, By_Reference< Integer > *tp*) Assigns to `this` the result of computing the *H79-widening* between `this` and `y`.
Parameters

y	A polyhedron that *must* be contained in `this`;
tp	A reference to an unsigned variable storing the number of available tokens (to be used when applying the *widening with tokens* delay technique).

Exceptions

Invalid_Argument_-Exception	Thrown if `this` and `y` are topology-incompatible or dimension-incompatible.

native void parma_polyhedra_library.Polyhedron.limited_BHRZ03_extrapolation_assign (Polyhedron *y*, Constraint_System *cs*, By_Reference< Integer > *tp*) Improves the result of the *BHRZ03-widening* computation by also enforcing those constraints in `cs` that are satisfied by all the points of `this`.
Parameters

y	A polyhedron that *must* be contained in `this`;
cs	The system of constraints used to improve the widened polyhedron;
tp	A reference to an unsigned variable storing the number of available tokens (to be used when applying the *widening with tokens* delay technique).

Exceptions

Invalid_Argument_-Exception	Thrown if `this`, `y` and `cs` are topology-incompatible or dimension-incompatible.

native void parma_polyhedra_library.Polyhedron.limited_H79_extrapolation_assign (Polyhedron *y*, Constraint_System *cs*, By_Reference< Integer > *tp*) Improves the result of the *H79-widening* computation by also enforcing those constraints in `cs` that are satisfied by all the points of `this`.

Parameters

y	A polyhedron that *must* be contained in `this`;
cs	The system of constraints used to improve the widened polyhedron;
tp	A reference to an unsigned variable storing the number of available tokens (to be used when applying the *widening with tokens* delay technique).

Exceptions

Invalid_Argument_-Exception	Thrown if `this`, `y` and `cs` are topology-incompatible or dimension-incompatible.

native void parma_polyhedra_library.Polyhedron.bounded_BHRZ03_extrapolation_assign (Polyhedron *y*, Constraint_System *cs*, By_Reference< Integer > *tp*) Improves the result of the *BHRZ03-widening* computation by also enforcing those constraints in `cs` that are satisfied by all the points of `this`, plus all the constraints of the form $\pm x \leq r$ and $\pm x < r$, with $r \in \mathbb{Q}$, that are satisfied by all the points of `this`.

Parameters

y	A polyhedron that *must* be contained in `this`;
cs	The system of constraints used to improve the widened polyhedron;
tp	A reference to an unsigned variable storing the number of available tokens (to be used when applying the *widening with tokens* delay technique).

Exceptions

Invalid_Argument_-Exception	Thrown if `this`, `y` and `cs` are topology-incompatible or dimension-incompatible.

native void parma_polyhedra_library.Polyhedron.bounded_H79_extrapolation_assign (Polyhedron *y*, Constraint_System *cs*, By_Reference< Integer > *tp*) Improves the result of the *H79-widening* computation by also enforcing those constraints in `cs` that are satisfied by all the points of `this`, plus all the constraints of the form $\pm x \leq r$ and $\pm x < r$, with $r \in \mathbb{Q}$, that are satisfied by all the points of `this`.

Parameters

y	A polyhedron that *must* be contained in `this`;
cs	The system of constraints used to improve the widened polyhedron;

tp	A reference to an unsigned variable storing the number of available tokens (to be used when applying the *widening with tokens* delay technique).

Exceptions

Invalid_Argument_-Exception	Thrown if `this`, `y` and `cs` are topology-incompatible or dimension-incompatible.

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

- Fake_Class_for_Doxygen.java

10.54 parma_polyhedra_library.Relation_Symbol Enum Reference

Relation symbols.

Public Attributes

- LESS_THAN

  Less than.
- LESS_OR_EQUAL

  Less than or equal to.
- EQUAL

  Equal to.
- GREATER_OR_EQUAL

  Greater than or equal to.
- GREATER_THAN

  Greater than.

10.54.1 Detailed Description

Relation symbols.
The documentation for this enum was generated from the following file:

- Relation_Symbol.java

10.55 parma_polyhedra_library.Timeout_Exception Class Reference

Exceptions caused by timeout expiring.
Inherits RuntimeException.

Public Member Functions

- Timeout_Exception (String s)

  Constructor.

10.55.1 Detailed Description

Exceptions caused by timeout expiring.
The documentation for this class was generated from the following file:

- Timeout_Exception.java

10.56 parma_polyhedra_library.Variable Class Reference

A dimension of the vector space.

Inherits Comparable< Variable >.

Public Member Functions

- Variable (long i)

  Builds the variable corresponding to the Cartesian axis of index `i`.
- long id ()

  Returns the index of the Cartesian axis associated to `this`.
- int compareTo (Variable v)

  Returns a negative number if `this` *is smaller than* `v`*, a zero if* `this` *equals* `v`*, a positive number if* `v` *is greater than* `v`.

Static Public Member Functions

- static native void setStringifier (Variable_Stringifier vs)

  Sets the variable stringifier object.

10.56.1 Detailed Description

A dimension of the vector space.

An object of the class Variable represents a dimension of the space, that is one of the Cartesian axes. Variables are used as basic blocks in order to build more complex linear expressions. Each variable is identified by a non-negative integer, representing the index of the corresponding Cartesian axis (the first axis has index 0).

10.56.2 Constructor & Destructor Documentation

parma_polyhedra_library.Variable.Variable (long *i*) `[inline]` Builds the variable corresponding to the Cartesian axis of index `i`.

Exceptions

RuntimeErrorException	Thrown if `i` is has negative value.

10.56.3 Member Function Documentation

static native void parma_polyhedra_library.Variable.setStringifier (Variable_Stringifier *vs*) `[static]`
Sets the variable stringifier object.

A variable stringifier object provides customization for the `toString` method; if no variable stringifier object is set (or if it is set to a null refeerence value), the PPL default variable output function will be used.

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

- Variable.java

10.57 parma_polyhedra_library.Variable_Stringifier Interface Reference

An interface for objects converting a Variable id to a string.

10.57.1 Detailed Description

An interface for objects converting a Variable id to a string.

If customized variable output is required, the user should implement this interface and pass a corresponding instance to Variable's static method `setStringifier`.

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

- Variable_Stringifier.java

10.58 parma_polyhedra_library.Variables_Set Class Reference

A java.util.TreeSet of variables' indexes.

Inherits TreeSet< Variable >.

Public Member Functions

- Variables_Set ()

  Builds the empty set of variable indexes.

10.58.1 Detailed Description

A java.util.TreeSet of variables' indexes.

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

- Variables_Set.java

Index