The Parma Polyhedra Library User's Manual[*] (version 0.3)

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

based on previous work also by

Sara Bonini
Andrea Pescetti
Angela Stazzone
Tatiana Zolo[‖]

February 26, 2002

[*]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".

[†]bagnara@cs.unipr.it, Department of Mathematics, University of Parma, Italy.
[‡]hill@comp.leeds.ac.uk, School of Computing, University of Leeds, U.K.
[§]ericci@cs.unipr.it, Department of Mathematics, University of Parma, Italy.
[¶]zaffanella@cs.unipr.it, Department of Mathematics, University of Parma, Italy.
[‖]zolo@cs.unipr.it, Department of Mathematics, University of Parma, Italy.

This document describes the Parma Polyhedra Library (PPL).

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

`http://www.cs.unipr.it/ppl/`

Contents

1 Convex Polyhedra and the PPL

1.1 An Introduction to Convex Polyhedra

Most of the following definitions and results are taken from:

- G. L. Nemhauser and L. A. Wolsey - Integer and Combinatorial Optimization - Wiley Interscience Series in Discrete Mathematics and Optimization, 1988.
- D. K. Wilde - A library for doing polyhedral operations - IRISA Publication interne n. 785, December 1993.
- K. Fukuda - Polyhedral Computation FAQ - Swiss Federal Institute of Technology, Lausanne and Zurich, Switzerland, October 2000.

Combination

Let $\lambda_1, \ldots, \lambda_k \in \mathbb{R}$ and $\mathbf{x}_1, \ldots, \mathbf{x}_k \in \mathbb{R}^n$. The linear combination $\sum_{j=1}^{k} \lambda_j \mathbf{x}_j$ is said to be

- a *positive combination*, if $\forall j \in \{1, \ldots, k\} : \lambda_j \geq 0$;
- an *affine combination*, if $\sum_{j=1}^{k} \lambda_j = 1$;
- a *convex combination*, if both the previous conditions hold.

Note that, when $k = 0$, we have $\sum_{j=1}^{k} \lambda_j \mathbf{x}_j = \mathbf{0}$ and $\sum_{j=1}^{k} \lambda_j = 0$. Therefore, the origin $\mathbf{0} \in \mathbb{R}^n$ can always be regarded as a positive (but not affine) linear combination of an empty set of vectors.

Scalar product

Let $\mathbf{x} = (x_0, \ldots, x_{n-1})^{\mathrm{T}}$, $\mathbf{y} = (y_0, \ldots, y_{n-1})^{\mathrm{T}} \in \mathbb{R}^n$. The *scalar product* of $\mathbf{x}$ and $\mathbf{y}$ is defined as

$$\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=0}^{n-1} x_i y_i .$$

The vectors $\mathbf{x}$ and $\mathbf{y}$ are *orthogonal* if $\langle \mathbf{x}, \mathbf{y} \rangle = 0$.

Convex hull

The *convex hull* of a set $K \subseteq \mathbb{R}^n$ is the set of all the convex combinations of the points in K. The set K is convex if it is its own convex hull.

Affine transformation

An *affine transformation* is a function mapping a point $\mathbf{x} \in \mathbb{R}^n$ to a point $\mathbf{x}' \in \mathbb{R}^m$ such that

$$\mathbf{x}' = A\mathbf{x} + \mathbf{b}$$

where $A \in \mathbb{R}^m \times \mathbb{R}^n$ and $\mathbf{b} \in \mathbb{R}^m$.

Linear independence

A finite set of points $\{\mathbf{x}_1, \ldots, \mathbf{x}_k\} \subseteq \mathbb{R}^n$ is *linearly independent* if, for all $\lambda_1, \ldots, \lambda_k \in \mathbb{R}$, the set of equations

$$\sum_{i=1}^{k} \lambda_i \mathbf{x}_i = \mathbf{0}$$

implies that, for each $i = 1, \ldots, k$, $\lambda_i = 0$.

Note that the maximum number of linearly independent points in $\mathbb{R}^n$ is n.

Proposition

If A is an $m \times n$ matrix, the maximum number of linearly independent rows of A, viewed as vectors of $\mathbb{R}^n$, equals the maximum number of linearly independent columns of A, viewed as vectors of $\mathbb{R}^m$.

Rank

The maximum number of linearly independent rows (columns) of a matrix A is the *rank* of A and is denoted by $\operatorname{rank}(A)$.

Affine independence

A finite set of points $\{\mathbf{x}_1, \ldots, \mathbf{x}_k\} \subseteq \mathbb{R}^n$ is *affinely independent* if, for all $\lambda_1, \ldots, \lambda_k \in \mathbb{R}$, the set of equations

$$\sum_{i=1}^{k} \lambda_i \mathbf{x}_i = \mathbf{0}, \quad \sum_{i=1}^{k} \lambda_i = 0$$

implies that, for each $i = 1, \ldots, k$, $\lambda_i = 0$.

Note that linear independence implies affine independence, but the converse is not true. Moreover the maximum number of affinely independent points in $\mathbb{R}^n$ is $n+1$ (e.g., n linearly independent points and the origin $\mathbf{0}$).

Polyhedron

A set $P \subseteq \mathbb{R}^n$ is called a *polyhedron* if it is the set of solutions to a finite number of linear equalities and inequalities:

$$P = \{\, \mathbf{x} \in \mathbb{R}^n \mid A\mathbf{x} = \mathbf{b}, C\mathbf{x} \geq \mathbf{d} \,\},$$

where, if m_1 is the number of linear equalities and m_2 the number of linear inequalities, $A \in \mathbb{R}^{m_1} \times \mathbb{R}^n$, $\mathbf{b} \in \mathbb{R}^{m_1}$, $C \in \mathbb{R}^{m_2} \times \mathbb{R}^n$ and $\mathbf{d} \in \mathbb{R}^{m_2}$.

In the sequel, we will simply write "equality" and "inequality" to mean "linear equality" and "linear inequality", respectively; also, we will refer to either an equality or an inequality as a *constraint*.

Constraints representation

By definition, any polyhedron $P \subseteq \mathbb{R}^n$ can be represented by a *system of constraints*: namely, the system given by the union of the equalities specified by matrix A and vector $\mathbf{b}$ and the inequalities specified by matrix C and vector $\mathbf{d}$.

Note that, if $\mathbf{c}, \mathbf{x} \in \mathbb{R}^n$ and $\lambda \in \mathbb{R}$ and the system of constraints contains the equality $\langle \mathbf{c}, \mathbf{x} \rangle = \lambda$, then this one can be replaced by the two equivalent inequalities $\langle \mathbf{c}, \mathbf{x} \rangle \geq \lambda$ and $\langle \mathbf{c}, \mathbf{x} \rangle \leq \lambda$ (i.e., $\langle -\mathbf{c}, \mathbf{x} \rangle \geq -\lambda$). It follows that a polyhedron $P \subseteq \mathbb{R}^n$ can always be represented as a system of *inequality* constraints

$$P = \{\, \mathbf{x} \in \mathbb{R}^n \mid A\mathbf{x} \geq \mathbf{b} \,\}$$

for some matrix $A \in \mathbb{R}^m \times \mathbb{R}^n$ and vector $\mathbf{b} \in \mathbb{R}^m$.

Rational polyhedron

A polyhedron $P \subseteq \mathbb{R}^n$ is said to be *rational* if there exists a matrix $A \in \mathbb{R}^{m'} \times \mathbb{R}^n$ and a vector $\mathbf{b} \in \mathbb{R}^{m'}$ with rational coefficients such that

$$P = \{\, \mathbf{x} \in \mathbb{R}^n \mid A\mathbf{x} \geq \mathbf{b} \,\}.$$

In the sequel, we will consider only rational polyhedra and assume that, if $\{\, \mathbf{x} \in \mathbb{R}^n \mid A\mathbf{x} \geq \mathbf{b} \,\}$ is a system of constraints representing a polyhedron, then A and $\mathbf{b}$ have rational coefficients.

Universe polyhedron

A polyhedron $P \subseteq \mathbb{R}^n$ is called *universe polyhedron* if it is the whole space (i.e., $P = \mathbb{R}^n$).

Polytope

A polyhedron $P \subset \mathbb{R}^n$ is *bounded* if there exists a $\lambda \in \mathbb{R}$, $\lambda \geq 0$ such that

$$P \subseteq \Bigl\{\, (x_0, \ldots, x_{n-1})^{\mathrm{T}} \in \mathbb{R}^n \Bigm| -\lambda \leq x_j \leq \lambda \text{ for } j = 0, \ldots, n-1 \,\Bigr\}.$$

A bounded polyhedron is called a *polytope*.

Proposition

A polyhedron is a closed convex set.

Polyhedron dimension

A non-empty polyhedron $P \subseteq \mathbb{R}^n$ is of *dimension* k, denoted by $\dim(P) = k$, if the maximum number of affinely independent points in P is $k+1$.

Note:

What is the dimension of the *empty* polyhedron? If the above definition is applied to an empty polyhedron, then the answer would be -1.

Vertex

A *vertex* of a polyhedron P is any point in P which cannot be expressed as a convex combination of any other distinct points in P.

Ray

Let P, P_0 be the polyhedra

$$P = \{ \mathbf{x} \in \mathbb{R}^n \mid A\mathbf{x} \geq \mathbf{b} \} \neq \varnothing \quad \text{and } P_0 = \{ \mathbf{r} \in \mathbb{R}^n \mid A\mathbf{r} \geq \mathbf{0} \}$$

where $A \in \mathbb{R}^m \times \mathbb{R}^n$, $\mathbf{b} \in \mathbb{R}^m$. Then any point $\mathbf{r} \in P_0 \setminus \{\mathbf{0}\}$ is called a *ray* of P.

A ray indicates a direction in which the polyhedron P is infinite (i.e., unbounded).

Proposition

A point $\mathbf{r} \in \mathbb{R}^n \setminus \{\mathbf{0}\}$ is a ray of a non-empty polyhedron $P \subseteq \mathbb{R}^n$ if and only if, for any point $\mathbf{x} \in P$, $(\mathbf{x} + \lambda\mathbf{r}) \in P$ for all $\lambda \in \mathbb{R}, \lambda \geq 0$.

Extreme ray

A ray $\mathbf{r}$ of a polyhedron P is an *extreme ray* if and only if it cannot be expressed as a positive combination of any other pair $\mathbf{r}_1$ and $\mathbf{r}_2$ of rays of P, where $\mathbf{r} \neq \lambda\mathbf{r}_1$, $\mathbf{r} \neq \lambda\mathbf{r}_2$ and $\mathbf{r}_1 \neq \lambda\mathbf{r}_2$ for all $\lambda \in \mathbb{R}, \lambda \geq 0$.

Line

A *line* (or *bidirectional ray*) of a polyhedron $P \subseteq \mathbb{R}^n$ is a ray $\mathbf{l}$ of P such that $-\mathbf{l}$ is another ray of P.

Cone

A set $C \subseteq \mathbb{R}^n$ is a *cone* if

$$\mathbf{x} \in C \Rightarrow \lambda\mathbf{x} \in C \text{ for all } \lambda \in \mathbb{R}, \lambda \geq 0.$$

Polyhedral cone

The polyhedron $P = \{ \mathbf{x} \in \mathbb{R}^n \mid A\mathbf{x} \geq \mathbf{0} \}$ is a convex cone and is called *polyhedral cone*.

A polyhedral cone is either *pointed*, having the origin as its only vertex, or has no vertices at all.

Lineality space

Given a polyhedron $P = \{ \mathbf{x} \in \mathbb{R}^n \mid A\mathbf{x} \geq \mathbf{b} \}$, the *lineality space* of P is the set

$$\{ \mathbf{x} \in P \mid A\mathbf{x} = \mathbf{0} \}$$

and it is denoted by $\operatorname{lin.space}(P)$.

Minkowski's sum

Let $R, S \subseteq \mathbb{R}^n$ be two sets of vectors. Then the *Minkowski's sum* of R and S is:

$$R + S = \{ r + s \mid r \in R, s \in S \}.$$

Generators representation

A polyhedron $P \subseteq \mathbb{R}^n$ can also be represented by a finite set V of points of P, a finite set R of rays of P and a finite set L of lines of P. The elements of these three sets are the *generators* of P, in the sense that

$$P = \mathcal{V} + \mathcal{R} + \mathcal{L},$$

where the symbol '$+$' denotes the Minkowski's sum and

- $\mathcal{V}$ is the set of all the convex combinations of the points in V;
- $\mathcal{R}$ is the set of all the positive combinations of the rays in R; and

- $\mathcal{L}$ is the set of all the linear combinations of the lines in L.

Note that $\mathcal{V}$ is a polytope, $\mathcal{R}$ is a pointed cone, and $\mathcal{L}$ is $\mathrm{lin.space}(P)$. Moreover, if $V = \emptyset$, we obtain $\mathcal{V} = \emptyset$, so that $P = \emptyset$. In contrast, if both $R = L = \emptyset$, we obtain $\mathcal{R} = \mathcal{L} = \{\mathbf{0}\}$, so that $P = \mathcal{V}$.

Also note that V must contain all the vertices of P (hence the choice for its name). However, if P is a non-empty polyhedron having no vertices at all, then V necessarily contains points that are *not* vertices of P. For instance, the half-space of $\mathbb{R}^2$ corresponding to the single constraint $y \geq 0$ can be represented by taking the generators $V = \bigl\{(0,0)^{\mathrm{T}}\bigr\}$, $R = \bigl\{(0,1)^{\mathrm{T}}\bigr\}$ and $L = \bigl\{(1,0)^{\mathrm{T}}\bigr\}$. Note also that the only ray in R is *not* an extreme ray of P.

The following theorem states that, whenever a polyhedron P has a vertex, there exists a decomposition such that

- V is the set of all *vertices* of P;
- R is the set of all *extreme* rays of P; and
- $L = \emptyset$.

Minkowski's theorem

Let $P = \{\, \mathbf{x} \in \mathbb{R}^n \mid A\mathbf{x} \geq \mathbf{b} \,\}$ be a non-empty polyhedron where $\mathrm{rank}(A) = n$. Let V be the set of vertices and R the set of extreme rays of P. Let also $\mathcal{V}$ be the set of convex combinations of V and $\mathcal{R}$ the set of positive combinations of R. Then

$$P = \mathcal{V} + \mathcal{R}.$$

The conditions that P is not empty and $\mathrm{rank}(A) = n$ are equivalent to the condition that P has a vertex. (See also Nemhauser and Wolsey - Integer and Combinatorial Optimization - propositions 4.1 and 4.2 on pages 92 and 93).

Proposition

Under the same hypotheses of Minkowsky's theorem, if P is a rational polyhedron then all the vertices in V have rational coefficients and we can consider a set R of extreme rays having rational coefficients only.

The second theorem, called Weil's theorem, states that any system of generators having rational coefficients defines a rational polyhedron:

Weil's theorem

If A is a rational $m \times n$ matrix, B is a rational $m' \times n$ matrix and

$$Q = \left\{ \mathbf{x} \in \mathbb{R}^n \;\middle|\; \begin{array}{l} \mathbf{x}^{\mathrm{T}} = \mathbf{y}^{\mathrm{T}}A + \mathbf{z}^{\mathrm{T}}B, \\ \mathbf{y} = (y_0, \ldots, y_{m-1})^{\mathrm{T}} \in \mathbb{R}_+^m, \sum_{k=0}^{m-1} y_k = 1, \\ \mathbf{z} \in \mathbb{R}_+^{m'} \end{array} \right\},$$

then Q is a rational polyhedron.

In fact, since Q consists of the sum of convex combinations of the rows of A with positive combinations of the rows of B, we can think of A as the matrix of vertices and B as the matrix of rays.

Dual representation

Thus a rational polyhedron P has *dual representations*. That is, P can be represented by a system of constraints or a system of generators. Moreover, given one of the representations, there is an algorithm for computing the other.

Space dimension and dimension-compatibility

The *space dimension* of a polyhedron $P \subseteq \mathbb{R}^n$ is the dimension $n \in \mathbb{N}$ of the corresponding vector space. The space dimension of constraints, generators and other objects of the PPL are defined similarly. The space dimension of objects is important because most of the operations defined on polyhedra are well-defined if and only if the various arguments are space dimension-compatible.

The following (space) *dimension-compatibility* rules are defined:

- the polyhedra $P \subseteq \mathbb{R}^n$ and $Q \subseteq \mathbb{R}^m$ are dimension-compatible if and only if $n = m$;
- the constraint $\langle \mathbf{a}, \mathbf{x} \rangle = \lambda$ (or $\langle \mathbf{a}, \mathbf{x} \rangle \geq \lambda$), where $\mathbf{a}, \mathbf{x} \in \mathbb{R}^m$, is dimension-compatible with polyhedron $P \subseteq \mathbb{R}^n$ if and only if $m \leq n$;
- the generator $\mathbf{x} \in \mathbb{R}^m$ is dimension-compatible with polyhedron $P \subseteq \mathbb{R}^n$ if and only if $m \leq n$;
- a system of constraints (resp., generators) is dimension-compatible with a polyhedron if and only if all the constraints (resp., generators) in the system are dimension-compatible with the polyhedron.

Be careful not to confuse the dimension $k \leq n$ of a polyhedron $P \subseteq \mathbb{R}^n$ with the *space* dimension n of P, which is the dimension of the enclosing vector space. In particular, we can have $\dim(P) \neq \dim(Q)$ even though P and Q are dimension-compatible; and vice versa, P and Q may be dimension-incompatible polyhedra even though $\dim(P) = \dim(Q)$.

2 PPL Namespace Index

2.1 PPL Namespace List

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

3 PPL Compound Index

3.1 PPL Compound List

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

4 PPL Page Index

4.1 PPL Related Pages

Here is a list of all related documentation pages:

5 PPL Namespace Documentation

5.1 Parma_Polyhedra_Library Namespace Reference

The entire library is confined into this namespace.

Compounds

- class Parma_Polyhedra_Library::Variable

  A dimension of the space.
- class Parma_Polyhedra_Library::LinExpression

  A linear expression.
- class Parma_Polyhedra_Library::Constraint

  A linear equality or inequality.
- class Parma_Polyhedra_Library::ConSys

  A system of constraints.
- class Parma_Polyhedra_Library::ConSys::const_iterator
- class Parma_Polyhedra_Library::Generator

  A line, ray or vertex.
- class Parma_Polyhedra_Library::GenSys

  A system of generators.
- class Parma_Polyhedra_Library::GenSys::const_iterator
- class Parma_Polyhedra_Library::Polyhedron

  A convex polyhedron.

Enumerations

- enum GenSys_Con_Rel { NONE_SATISFIES, ALL_SATISFY, ALL_SATURATE, SOME_SATISFY }

  Describes possible relations between a system of generators and a given constraint.

Functions

- std::ostream & operator<< (std::ostream &s, GenSys_Con_Rel r)

  Output operator for GenSys_Con_Rel.

5.1.1 Enumeration Type Documentation

5.1.1.1 enum Parma_Polyhedra_Library::GenSys_Con_Rel

Enumeration values:

NONE_SATISFIES No generator satisfies the given constraint.

ALL_SATISFY All generators satisfy the given constraint, but there exists a generator not saturating it (i.e., a generator does not belong to the hyper-plane defined by the constraint.).

ALL_SATURATE All generators saturate the given constraint (i.e., they all belong to the hyper-plane defined by the constraint.).

SOME_SATISFY Some generators satisfy the given constraint (i.e., there exists both a generator satisfying the constraint and another generator which does not satisfy it.).

6 PPL Class Documentation

6.1 Parma_Polyhedra_Library::Constraint Class Reference

A linear equality or inequality.

Public Methods

- Constraint (const Constraint &c)

  Ordinary copy-constructor.

- ~Constraint ()

  Destructor.

- Constraint & operator= (const Constraint &c)

  Assignment operator.

- size_t space_dimension () const

  Returns the dimension of the vector space enclosing `*this`.

- bool is_equality () const

  Returns `true` *if and only if* `*this` *is an equality constraint.*

- bool is_inequality () const

  Returns `true` *if and only if* `*this` *is an inequality constraint.*

- const Integer & coefficient (Variable v) const

  If the index of variable `v` *is less than the space-dimension of* `*this`*, returns the coefficient of* `v` *in* `*this`.

Exceptions:

std::invalid_argument *thrown if the index of* `v` *is greater than or equal to the space-dimension of* `*this`.

- const Integer & coefficient () const

  Returns the inhomogeneous term of `*this`.

Static Public Methods

- const Constraint & zero_dim_false ()

  The unsatisfiable (zero-dimension space) constraint $0 = 1$.

- const Constraint & zero_dim_positivity ()

  The true (zero-dimension space) constraint $0 \leq 1$, *also known as* positivity constraint.

Friends

- Constraint Parma_Polyhedra_Library::operator== (const LinExpression &e1, const LinExpression &e2)

  Returns the constraint `e1 = e2`.

- Constraint Parma_Polyhedra_Library::operator== (const LinExpression &e, const Integer &n)

  Returns the constraint `e = n`.

- Constraint Parma_Polyhedra_Library::operator== (const Integer &n, const LinExpression &e)

  Returns the constraint `n = e`.

- Constraint Parma_Polyhedra_Library::operator>= (const LinExpression &e1, const LinExpression &e2)

  Returns the constraint `e1 >= e2`.

- Constraint Parma_Polyhedra_Library::operator>= (const LinExpression &e, const Integer &n)

  Returns the constraint `e >= n`.

- Constraint Parma_Polyhedra_Library::operator>= (const Integer &n, const LinExpression &e)

  Returns the constraint `n >= e`.

- Constraint Parma_Polyhedra_Library::operator<= (const LinExpression &e1, const LinExpression &e2)

  Returns the constraint `e1 <= e2`.

- Constraint Parma_Polyhedra_Library::operator<= (const LinExpression &e, const Integer &n)

  Returns the constraint `e <= n`.

- Constraint Parma_Polyhedra_Library::operator<= (const Integer &n, const LinExpression &e)

  Returns the constraint `n <= e`.

- Constraint Parma_Polyhedra_Library::operator>> (const Constraint &c, unsigned int offset)

  Returns the constraint `c` *with variables renamed by adding* `offset` *to their Cartesian axis identifier.*

Related Functions

(Note that these are not member functions.)

- std::ostream & operator<< (std::ostream &s, const Constraint &c)
  Output operator.

6.1.1 Detailed Description

An object of the class Constraint is either:

- an equality: $\sum_{i=0}^{n-1} a_i x_i + b = 0$; or
- an inequality: $\sum_{i=0}^{n-1} a_i x_i + b \geq 0$;

where n is the dimension of the space, a_i is the integer coefficient of variable x_i and b is the integer inhomogeneous term.

How to build a constraint

Constraints are typically built by applying a relational operator to a pair of linear expressions. Available relational operators include equality (==) and non-strict inequalities (>= and <=). Strict inequalities (< and >) are not supported. The space-dimension of a constraint is defined as the maximum space-dimension of the arguments of its constructor.

In the following examples it is assumed that variables `x`, `y` and `z` are defined as follows:

```
Variable x(0);
Variable y(1);
Variable z(2);
```

Example 1

The following code builds the equality constraint $3x + 5y - z = 0$, having space-dimension 3:

```
Constraint eq_c(3*x + 5*y - z == 0);
```

The following code builds the inequality constraint $4x \geq 2y - 13$, having space-dimension 2:

```
Constraint ineq_c(4*x >= 2*y - 13);
```

The unsatisfiable constraint on the zero-dimension space $\mathbb{R}^0$ can be specified as follows:

```
Constraint false_c = Constraint::zero_dim_false();
```

An equivalent, but more involved way is the following:

```
Constraint false_c(LinExpression::zero() == 1);
```

In constrast, the following code defines an unsatisfiable constraint having space-dimension 3:

```
Constraint false_c(0*z == 1);
```

How to inspect a constraint

Several methods are provided to examine a constraint and extract all the encoded information: its space-dimension, its type (equality or inequality) and the value of its integer coefficients.

Example 2

The following code shows how it is possible to access each single coefficient of a constraint. Given an arbitrary constraint (in this case $x - 5y + 3z <= 4$), we construct a new constraint having the same coefficients but with a different relational operator (thus, in this case we want to obtain the equality constraint $x - 5y + 3z = 4$).

```
Constraint c1(x - 5*y + 3*z <= 4);
cout << "Constraint c1: " << c1 << endl;
LinExpression e;
for (int i = c1.space_dimension() - 1; i >= 0; i--)
  e += c1.coefficient(Variable(i)) * Variable(i);
e += c1.coefficient();
Constraint c2 = c1.is_equality() ? (e >= 0) : (e == 0);
cout << "Constraint c2: " << c2 << endl;
```

The actual output is the following:

```
Constraint c1: -A + 5*B - 3*C >= -4
Constraint c2: -A + 5*B - 3*C = -4
```

Note that, in general, the particular output obtained can be syntactically different from the (semantically equivalent) constraint considered.

6.2 Parma_Polyhedra_Library::ConSys Class Reference

A system of constraints.

Public Methods

- ConSys ()

  Default constructor: builds an empty system of constraints.

- ConSys (const Constraint &c)

  Builds the singleton system containing only constraint `c`.

- ConSys (const ConSys &cs)

  Ordinary copy-constructor.

- virtual ~ConSys ()

  Destructor.

- ConSys & operator= (const ConSys &y)

  Assignment operator.

- size_t space_dimension () const

  Returns the dimension of the vector space enclosing `*this`.

- void insert (const Constraint &c)

  Inserts a copy of the constraint `c` *into* `*this`*, increasing the number of dimensions if needed.*

- const_iterator begin () const

  Returns the const_iterator pointing to the first constraint, if `*this` *is not empty; otherwise, returns the past-the-end const_iterator.*

- const_iterator end () const

  Returns the past-the-end const_iterator.

Static Public Methods

- const ConSys & zero_dim_empty ()

  Returns the singleton system containing only Constraint::zero_dim_false().

6.2.1 Detailed Description

An object of the class ConSys is a system of constraints, i.e., a multiset of objects of the class Constraint. When inserting constraints in a system, dimensions are automatically adjusted so that all the constraints in the system are defined on the same vector space.

In all the examples it is assumed that variables `x` and `y` are defined as follows:

```
Variable x(0);
Variable y(1);
```

Example 1

The following code builds a system of constraints corresponding to a square in $\mathbb{R}^2$:

```
ConSys cs;
cs.insert(x >= 0);
cs.insert(x <= 3);
cs.insert(y >= 0);
cs.insert(y <= 3);
```

Note that: the constraint system is created with space dimension zero; the first and third constraint insertions increases the space dimension to 1 and 2, respectively.

Example 2

The following code builds a system of constraints corresponding to a half-strip in $\mathbb{R}^2$:

```
ConSys cs;
cs.insert(x >= 0);
cs.insert(x - y <= 0);
cs.insert(x - y + 1 >= 0);
```

Note:

After inserting a multiset of constraints in a constraint system, there are no guarantees that an *exact* copy of them can be retrieved: in general, only an *equivalent* constraint system will be available, where original constraints may have been reordered, removed (if they are trivial, duplicate or implied by other constraints), linearly combined, etc.

6.3 Parma_Polyhedra_Library::ConSys::const_iterator Class Reference

Public Methods

- const_iterator ()

  Default constructor.

- const_iterator (const const_iterator &y)

  Ordinary copy-constructor.

- virtual ~const_iterator ()

  Destructor.

- const_iterator & operator= (const const_iterator &y)

  Assignment operator.

- const Constraint & operator ∗ () const

  Dereference operator.

- const Constraint ∗ operator → () const

  Indirect member selector.

- const_iterator & operator++ ()

  Prefix increment operator.

- const_iterator operator++ (int)

  Postfix increment operator.

- bool operator== (const const_iterator &y) const

  Returns `true` *if and only if* ∗`this` *and* `y` *are identical.*

- bool operator!= (const const_iterator &y) const

  Returns `true` *if and only if* ∗`this` *and* `y` *are different.*

6.3.1 Detailed Description

A const_iterator is used to provide read-only access to each constraint contained in an object of ConSys.

Example

The following code prints the system of constraints defining the polyhedron `ph`:

```
const ConSys cs = ph.constraints();
ConSys::const_iterator iend = cs.end();
for (ConSys::const_iterator i = cs.begin(); i != iend; ++i)
  cout << *i << endl;
```

6.4 Parma_Polyhedra_Library::Generator Class Reference

A line, ray or vertex.

Public Types

- enum Type

  The generator type.

Public Methods

- Generator (const Generator &g)

  Ordinary copy-constructor.

- ∼Generator ()

  Destructor.

- Generator & operator= (const Generator &g)

  Assignment operator.

- size_t space_dimension () const

  Returns the dimension of the vector space enclosing `*this`.

- Type type () const

  Returns the generator type of `*this`.

- const Integer & coefficient (Variable v) const

  If the index of variable `v` *is less than the space-dimension of* `*this`*, returns the coefficient of* `v` *in* `*this`.

  Exceptions:

  std::invalid_argument *thrown if the index of* `v` *is greater than or equal to the space-dimension of* `*this`.

- const Integer & divisor () const

  If `*this` *is a vertex, returns its divisor.*

  Exceptions:

  std::invalid_argument *thrown if* `*this` *is not a vertex.*

- bool OK () const

  Checks if all the invariants are satisfied.

Static Public Methods

- const Generator & zero_dim_vertex ()

  Returns the origin of the zero-dimensional space $\mathbb{R}^0$.

Friends

- Generator Parma_Polyhedra_Library::line (const LinExpression &e)

  Returns the (bidirectional) line of direction `e`.

  Exceptions:

  std::invalid_argument *thrown if the homogeneous part of* `e` *represents the origin of the vector space.*

- Generator Parma_Polyhedra_Library::ray (const LinExpression &e)

  Returns the (unidirectional) ray of direction `e`.

  Exceptions:

  std::invalid_argument *thrown if the homogeneous part of* `e` *represents the origin of the vector space.*

- Generator Parma_Polyhedra_Library::vertex (const LinExpression &e=LinExpression::zero(), const Integer &d=Integer_one())

  Returns the vertex at `e` / `d` *Both* `e` *and* `d` *are optional arguments, with default values LinExpression::zero() and Integer_one(), respectively.*

  Exceptions:

  std::invalid_argument *thrown if* `d` *is zero.*

Related Functions

(Note that these are not member functions.)

- std::ostream & operator<< (std::ostream &s, const Generator &g)

  Output operator.

6.4.1 Detailed Description

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

- a line $\mathbf{l} = (a_0, \ldots, a_{n-1})^{\mathrm{T}}$;
- a ray $\mathbf{r} = (a_0, \ldots, a_{n-1})^{\mathrm{T}}$;
- a vertex $\mathbf{v} = (\frac{a_0}{d}, \ldots, \frac{a_{n-1}}{d})^{\mathrm{T}}$;

where n is the dimension of the space.

A note on terminology.

As observed in the Introduction, there are cases when, in order to represent a polyhedron P using generators, we need to include in the finite set V even points of P that are *not* vertices of P. Nonetheless, accordingly to what is now an established terminology, we will call *vertex* any element of the set of generators V, even though it is not a "proper" vertex of P.

How to build a generator.

Each type of generator is built by applying the corresponding function (`line`, `ray` or `vertex`) to a linear expression, representing a direction in the space; the space-dimension of the generator is defined as the space-dimension of the corresponding linear expression. Linear expressions used to define a generator should be homogeneous (any constant term will be simply ignored). When defining a vertex, an optional Integer argument can be used as a common *divisor* for all the coefficients occurring in the provided linear expression; the default value for this argument is 1.

In all the following examples it is assumed that variables `x`, `y` and `z` are defined as follows:

```
Variable x(0);
Variable y(1);
Variable z(2);
```

Example 1

The following code builds a line with direction $x - y - z$ and having space-dimension 3:

```
Generator l = line(x - y - z);
```

As mentioned above, the constant term of the linear expression is not relevant. Thus, the following code has the same effect:

```
Generator l = line(x - y - z + 15);
```

By definition, the origin of the space is not a line, so that the following code throws an exception:

```
Generator l = line(0*x);
```

Example 2

The following code builds a ray with the same direction as the line in Example 1:

```
Generator r = ray(x - y - z);
```

As is the case for lines, when specifying a ray the constant term of the linear expression is not relevant; also, an exception is thrown when trying to built a ray from the origin of the space.

Example 3

The following code builds the vertex $\mathbf{v} = (1, 0, 2)^{\mathrm{T}} \in \mathbb{R}^3$:

```
Generator v = vertex(1*x + 0*y + 2*z);
```

The same effect can be obtained by using the following code:

```
Generator v = vertex(x + 2*z);
```

Similarly, the origin $\mathbf{0} \in \mathbb{R}^3$ can be defined using either one of the following lines of code:

```
Generator origin3 = vertex(0*x + 0*y + 0*z);
Generator origin3_alt = vertex(0*z);
```

Note however that the following code would have defined a different vertex, namely $\mathbf{0} \in \mathbb{R}^2$:

```
Generator origin2 = vertex(0*y);
```

The following two lines of code both define the only vertex having space-dimension zero, namely $\mathbf{0} \in \mathbb{R}^0$. In the second case we exploit the fact that the first argument of the function `vertex` is optional.

```
Generator origin0 = Generator::zero_dim_vertex();
Generator origin0_alt = vertex();
```

Example 4

The vertex $\mathbf{v}$ specified in Example 3 above can also be obtained with the following code, where we provide a non-default value for the second argument of the function `vertex` (the divisor):

```
Generator v = vertex(2*x + 0*y + 4*z, 2);
```

Obviously, the divisor can be usefully exploited to specify vertices having some non-integer (but rational) coordinates. For instance, the vertex $\mathbf{w} = (-1.5, 3.2, 2.1)^{\mathrm{T}} \in \mathbb{R}^3$ can be specified by the following code:

```
Generator w = vertex(-15*x + 32*y + 21*z, 10);
```

If a zero divisor is provided, an exception is thrown.

How to inspect a generator

Several methods are provided to examine a generator and extract all the encoded information: its space-dimension, its type and the value of its integer coefficients.

Example 5

The following code shows how it is possible to access each single coefficient of a generator. If `v1` is a vertex having coordinates $(a_0, \ldots, a_{n-1})^{\mathrm{T}}$, we construct the vertex `v2` having coordinates $(a_0, 2a_1, \ldots, (i+1)a_i, \ldots, na_{n-1})^{\mathrm{T}}$.

```
if (g1.type() == Generator::VERTEX) {
  cout << "Vertex g1: " << g1 << endl;
  LinExpression e;
  for (int i = g1.space_dimension() - 1; i >= 0; i--)
    e += (i + 1) * g1.coefficient(Variable(i)) * Variable(i);
  Generator g2 = vertex(e, g1.divisor());
  cout << "Vertex g2: " << g2 << endl;
}
else
  cout << "Generator g1 is not a vertex." << endl;
```

Therefore, for the vertex

```
Generator g1 = vertex(2*x - y + 3*z, 2);
```

we would obtain the following output:

```
Vertex g1: v((2*A - B + 3*C)/2)
Vertex g2: v((2*A - 2*B + 9*C)/2)
```

When working with a vertex, be careful not to confuse the notion of *coefficient* with the notion of *coordinate*: these are equivalent only when the divisor of the vertex is 1.

6.5 Parma_Polyhedra_Library::GenSys Class Reference

A system of generators.

Public Methods

- GenSys ()

  Default constructor: builds an empty system of generators.

- GenSys (const Generator &g)

  Builds the singleton system containing only generator `g`.

- GenSys (const GenSys &gs)

  Ordinary copy-constructor.

- virtual ~GenSys ()

  Destructor.

- GenSys & operator= (const GenSys &y)

  Assignment operator.

- size_t space_dimension () const

  Returns the dimension of the vector space enclosing $*$`this`.

- void insert (const Generator &g)

  Inserts a copy of the generator `g` *into* $*$`this`*, increasing the number of dimensions if needed.*

- const_iterator begin () const

  Returns the const_iterator pointing to the first generator, if `*this` *is not empty; otherwise, returns the past-the-end const_iterator.*

- const_iterator end () const

  Returns the past-the-end const_iterator.

Static Public Methods

- const GenSys & zero_dim_univ ()

  Returns the singleton system containing only Generator::zero_dim_vertex().

6.5.1 Detailed Description

An object of the class GenSys is a system of generators, i.e., a multiset of objects of the class Generator (lines, rays and vertices). When inserting generators in a system, dimensions are automatically adjusted so that all the generators in the system are defined on the same vector space. A system of generators which is meant to define a non-empty polyhedron must include at least one vertex, even if the polyhedron has no "proper" vertices: the reason is that lines and rays need a supporting point (they only specify directions).

In all the examples it is assumed that variables `x` and `y` are defined as follows:

```
Variable x(0);
Variable y(1);
```

Example 1

The following code defines the line having the same direction as the x axis (i.e., the first Cartesian axis) in $\mathbb{R}^2$:

```
GenSys gs;
gs.insert(line(x + 0*y));
```

As said above, this system of generators corresponds to an empty polyhedron, because the line has no supporting point. To define a system of generators indeed corresponding to the x axis, one can add the following code which inserts the origin of the space as a vertex:

```
gs.insert(vertex(0*x + 0*y));
```

Since dimensions are automatically adjusted, the following code obtains the same effect:

```
gs.insert(vertex(0*x));
```

In contrast, if we had added the following code, we would have defined a line parallel to the x axis and including the point $(0, 1)^{\mathrm{T}} \in \mathbb{R}^2$.

```
gs.insert(vertex(0*x + 1*y));
```

Example 2

The following code builds a ray having the same direction as the positive part of the x axis in $\mathbb{R}^2$:

```
GenSys gs;
gs.insert(ray(x + 0*y));
```

To define a system of generators indeed corresponding to the set

$$\{\,(x,0)^{\mathrm{T}} \in \mathbb{R}^2 \mid x \geq 0\,\},$$

one just has to add the origin:

```
gs.insert(vertex(0*x + 0*y));
```

Example 3

The following code builds a system of generators having four vertices and corresponding to a square in $\mathbb{R}^2$ (the same as Example 1 for the system of constraints):

```
GenSys gs;
gs.insert(vertex(0*x + 0*y));
gs.insert(vertex(0*x + 3*y));
gs.insert(vertex(3*x + 0*y));
gs.insert(vertex(3*x + 3*y));
```

Example 4

The following code builds a system of generators having two vertices and a ray, corresponding to a half-strip in $\mathbb{R}^2$ (the same as Example 2 for the system of constraints):

```
GenSys gs;
gs.insert(vertex(0*x + 0*y));
gs.insert(vertex(0*x + 1*y));
gs.insert(ray(x - y));
```

Note:

After inserting a multiset of generators in a generator system, there are no guarantees that an *exact* copy of them can be retrieved: in general, only an *equivalent* generator system will be available, where original generators may have been reordered, removed (if they are duplicate or redundant), etc.

6.6 Parma_Polyhedra_Library::GenSys::const_iterator Class Reference

Public Methods

- const_iterator ()

  Default constructor.

- const_iterator (const const_iterator &y)

  Ordinary copy-constructor.

- virtual ~const_iterator ()

  Destructor.

- const_iterator & operator= (const const_iterator &y)

  Assignment operator.

- const Generator & operator $*$ () const

  Dereference operator.

- const Generator $*$ operator $\rightarrow$ () const

  Indirect member selector.

- const_iterator & operator++ ()

  Prefix increment operator.

- const_iterator operator++ (int)

  Postfix increment operator.

- bool operator== (const const_iterator &y) const

  Returns `true` *if and only if* ∗`this` *and* `y` *are identical.*

- bool operator!= (const const_iterator &y) const

  Returns `true` *if and only if* ∗`this` *and* `y` *are different.*

6.6.1 Detailed Description

A const_iterator is used to provide read-only access to each generator contained in an object of GenSys.

Example

The following code prints the system of generators of the polyhedron `ph`:

```
const GenSys gs = ph.generators();
GenSys::const_iterator iend = gs.end();
for (GenSys::const_iterator i = gs.begin(); i != iend; ++i)
  cout << *i << endl;
```

The same effect can be obtained more concisely by using more features of the STL:

```
const GenSys gs = ph.generators();
copy(gs.begin(), gs.end(), ostream_iterator<Generator>(cout, "\n"));
```

6.7 Parma_Polyhedra_Library::LinExpression Class Reference

A linear expression.

Public Methods

- LinExpression ()

  Default constructor: returns a copy of LinExpression::zero().

- LinExpression (const LinExpression &e)

  Ordinary copy-constructor.

- virtual ∼LinExpression ()

  Destructor.

- LinExpression (const Integer &n)

  Constructor: builds the linear expression corresponding to the inhomogeneous term `n`.

- LinExpression (const Variable &v)

  Constructor: builds the linear expression corresponding to the variable `v`.

- size_t space_dimension () const

  Returns the dimension of the vector space enclosing ∗`this`.

Static Public Methods

- const LinExpression & zero ()

  Returns the (zero-dimension space) constant 0.

Friends

- LinExpression Parma_Polyhedra_Library::operator+ (const LinExpression &e1, const LinExpression &e2)

  Returns the linear expression `e1 + e2`.

- LinExpression Parma_Polyhedra_Library::operator+ (const Integer &n, const LinExpression &e)

  Returns the linear expression `n + e`.

- LinExpression Parma_Polyhedra_Library::operator+ (const LinExpression &e, const Integer &n)

  Returns the linear expression `e + n`.

- LinExpression Parma_Polyhedra_Library::operator- (const LinExpression &e)

  Returns the linear expression `- e`.

- LinExpression Parma_Polyhedra_Library::operator- (const LinExpression &e1, const LinExpression &e2)

  Returns the linear expression `e1 - e2`.

- LinExpression Parma_Polyhedra_Library::operator- (const Integer &n, const LinExpression &e)

  Returns the linear expression `n - e`.

- LinExpression Parma_Polyhedra_Library::operator- (const LinExpression &e, const Integer &n)

  Returns the linear expression `e - n`.

- LinExpression Parma_Polyhedra_Library::operator * (const Integer &n, const LinExpression &e)

  Returns the linear expression `n * e`.

- LinExpression Parma_Polyhedra_Library::operator * (const LinExpression &e, const Integer &n)

  Returns the linear expression `e * n`.

- LinExpression & Parma_Polyhedra_Library::operator+= (LinExpression &e1, const LinExpression &e2)

  Returns the linear expression `e1 + e2` *and assigns it to* `e1`.

- LinExpression & Parma_Polyhedra_Library::operator+= (LinExpression &e, const Variable &v)

  Returns the linear expression `e + v` *and assigns it to* `e`.

- LinExpression & Parma_Polyhedra_Library::operator+= (LinExpression &e, const Integer &n)

  Returns the linear expression `e + n` *and assigns it to* `e`.

6.7.1 Detailed Description

An object of the class LinExpression represents the linear expression

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

where n is the dimension of the space, each a_i is the integer coefficient of the `i` -th variable x_i and b is the integer for the inhomogeneous term.

How to build a linear expression.

Linear expressions are the basic blocks for defining both constraints (i.e., linear equalities or inequalities) and generators (i.e., lines, rays and vertices). A full set of functions is defined to provide a convenient interface for building complex linear expressions starting from simpler ones and from objects of the classes Variable and Integer: available operators include unary negation, binary addition and subtraction, as well as multiplication by an Integer. The space-dimension of a linear expression is defined as the maximum space-dimension of the arguments used to build it: in particular, the space-dimension of a Variable `x` is defined as `x.id()+1`, whereas all the objects of the class Integer have space-dimension zero.

Example

The following code builds the linear expression $4x - 2y - z + 14$, having space-dimension 3:

```
LinExpression e = 4*x - 2*y - z + 14;
```

Another way to build the same linear expression is:

```
LinExpression e1 = 4*x;
LinExpression e2 = 2*y;
LinExpression e3 = z;
LinExpression e = LinExpression(14);
e += e1 - e2 - e3;
```

Note that `e1`, `e2` and `e3` have space-dimension 1, 2 and 3, respectively; also, in the fourth line of code, `e` is created with space-dimension zero and then extended to space-dimension 3.

6.8 Parma_Polyhedra_Library::Polyhedron Class Reference

A convex polyhedron.

Public Types

- enum Degenerate_Kind { UNIVERSE, EMPTY }

  Kinds of degenerate polyhedra.

Public Methods

- Polyhedron (const Polyhedron &y)

  Ordinary copy-constructor.

- Polyhedron (size_t num_dimensions=0, Degenerate_Kind kind=UNIVERSE)

Builds either the universe or the empty polyhedron of dimension `num_dimensions`. *Both parameters are optional: by default, a 0-dimension space universe polyhedron is built.*

- Polyhedron (ConSys &cs)

  Builds a polyhedron from a system of constraints. The polyhedron inherits the space dimension of the constraint system.

  Parameters:

  cs *The system of constraints defining the polyhedron. It is not declared* `const` *because it can be modified.*

- Polyhedron (GenSys &gs)

  Builds a polyhedron from a system of generators. The polyhedron inherits the space dimension of the generator system.

  Parameters:

  gs *The system of generators defining the polyhedron. It is not declared* `const` *because it can be modified.*

  Exceptions:

  std::invalid_argument *thrown if the system of generators is not empty but has no vertices.*

- Polyhedron & operator= (const Polyhedron &y)

  The assignment operator. (Note that ∗`this` *and* `y` *can be dimension-incompatible.).*

- size_t space_dimension () const

  Returns the dimension of the vector space enclosing ∗`this`.

- void intersection_assign_and_minimize (const Polyhedron &y)

  Intersects ∗`this` *with polyhedron* `y` *and assigns the result to* ∗`this`. *The result is not guaranteed to be minimized.*

  Exceptions:

  std::invalid_argument *thrown if* ∗`this` *and* `y` *are dimension-incompatible.*

- void intersection_assign (const Polyhedron &y)

  Intersects ∗`this` *with polyhedron* `y` *and assigns the result to* ∗`this` *without minimizing the result.*

  Exceptions:

  std::invalid_argument *thrown if* ∗`this` *and* `y` *are dimension-incompatible.*

- void convex_hull_assign_and_minimize (const Polyhedron &y)

  Assigns to ∗`this` *the convex hull of the set-theoretic union* ∗`this` *and* `y`, *minimizing the result.*

  Exceptions:

  std::invalid_argument *thrown if* ∗`this` *and* `y` *are dimension-incompatible.*

- void convex_hull_assign (const Polyhedron &y)

  Assigns to ∗`this` *the convex hull of the set-theoretic union* ∗`this` *and* `y`. *The result is not guaranteed to be minimized.*

  Exceptions:

  std::invalid_argument *thrown if* ∗`this` *and* `y` *are dimension-incompatible.*

- void convex_difference_assign_and_minimize (const Polyhedron &y)

  Assigns to ∗`this` *the convex hull of the set-theoretic difference* ∗`this` *and* `y`, *minimizing the result.*

  Exceptions:

  std::invalid_argument *thrown if* ∗`this` *and* `y` *are dimension-incompatible.*

- void convex_difference_assign (const Polyhedron &y)

Assigns to `*this` *the convex hull of the set-theoretic difference* `*this` *and* `y`*. The result is not guaranteed to be minimized.*

Exceptions:

std::invalid_argument *thrown if* `*this` *and* `y` *are dimension-incompatible.*

- GenSys_Con_Rel satisfies (const Constraint &c)

  Returns the relation between the generators of `*this` *and the constraint* `c`*.*

  Exceptions:

  std::invalid_argument *thrown if* `*this` *and constraint* `c` *are dimension-incompatible.*

- bool includes (const Generator &g)

  Tests the inclusion of the generator `g` *in the polyhedron* `*this`*.*

  Exceptions:

  std::invalid_argument *thrown if* `*this` *and constraint* `g` *are dimension-incompatible.*

- void widening_assign (const Polyhedron &y)

  Computes the widening between `*this` *and* `y` *and assigns the result to* `*this`*.*

  Parameters:

  y *The polyhedron that* must *be contained in* `*this`*.*

  Exceptions:

  std::invalid_argument *thrown if* `*this` *and* `y` *are dimension-incompatible.*

- void limited_widening_assign (const Polyhedron &y, ConSys &cs)

  Limits the widening between `*this` *and* `y` *by* `cs` *and assigns the result to* `*this`*.*

  Parameters:

  y *The polyhedron that* must *be contained in* `*this`*.*

  cs *The system of constraints that limits the widened polyhedron. It is not declared* `const` *because it can be modified.*

  Exceptions:

  std::invalid_argument *thrown if* `*this`*,* `y` *and* `cs` *are dimension-incompatible.*

- const ConSys & constraints () const

  Returns the system of constraints.

- const GenSys & generators () const

  Returns the system of generators.

- void insert (const Constraint &c)

  Inserts a copy of constraint `c` *into the system of constraints of* `*this`*.*

  Exceptions:

  std::invalid_argument *thrown if* `*this` *and constraint* `c` *are dimension-incompatible.*

- void insert (const Generator &g)

  Inserts a copy of generator `g` *into the system of generators of* `*this`*.*

  Exceptions:

  std::invalid_argument *thrown if* `*this` *and generator* `g` *are dimension-incompatible or if a ray/line is inserted in an empty polyhedron.*

- void affine_image (const Variable &v, const LinExpression &expr, const Integer &denominator=Integer_one())

  Transforms the polyhedron `*this`*, assigning an affine expression to the specified variable.*

  Parameters:

  v *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:

std::invalid_argument *thrown if* `denominator` *is zero or if* `expr` *and* `*this` *are dimension-incompatible or if* `v` *is not a dimension of* `*this`.

- void affine_preimage (const Variable &v, const LinExpression &expr, const Integer &denominator=Integer_one())

  Transforms the polyhedrons `*this`*, substituting an affine expression for the specified variable. (It is the inverse operation of* `affine_image`*.)*

  Parameters:

  v *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:

  std::invalid_argument *thrown if* `denominator` *is zero or if* `expr` *and* `*this` *are dimension-incompatible or if* `v` *is not a dimension of* `*this`.

- bool OK (bool check_not_empty=true) const

  Checks if all the invariants are satisfied.

  Parameters:

  check_not_empty `true` *if it must be checked whether the system of constraint is satisfiable.*

  Returns:

  `true` *if the polyhedron satisfies all the invariants stated in the PPL,* `false` *otherwise.*

- void add_dimensions_and_embed (size_t dim)

  Adds new dimensions and embeds the old polyhedron into the new space.

  Parameters:

  dim *The number of dimensions to add.*

- void add_dimensions_and_project (size_t dim)

  Adds new dimensions to the polyhedron and does not embed it in the new space.

  Parameters:

  dim *The number of dimensions to add.*

- void remove_dimensions (const std::set< Variable > &to_be_removed)

  Removes the specified dimensions.

  Parameters:

  to_be_removed *The set of variables to remove.*

- void remove_higher_dimensions (size_t new_dimension)

  Removes all dimensions higher than a threshold.

  Parameters:

  new_dimension *The dimension of the resulting polyhedron after all higher dimensions have been removed.*

- bool add_constraints_and_minimize (ConSys &cs)

  Adds the specified constraints and computes a new polyhedron.

  Parameters:

  cs *The constraints that will be added to the current system of constraints. This parameter is not declared* `const` *because it can be modified.*

  Returns:

  `false` *if the resulting polyhedron is empty.*

  Exceptions:

  std::invalid_argument *thrown if* `*this` *and* `cs` *are dimension-incompatible.*

- void add_constraints (ConSys &cs)

  Adds the specified constraints without minimizing.

  Parameters:

  cs *The constraints that will be added to the current system of constraints. This parameter is not declared* `const` *because it can be modified.*

  Exceptions:

  std::invalid_argument *thrown if* `*this` *and* `cs` *are dimension-incompatible.*

- void add_dimensions_and_constraints (ConSys &cs)

  First increases the space dimension of `*this` *by adding* `cs.space_dimension()` *new dimensions; then adds to the system of constraints of* `*this` *a renamed-apart version of the constraints in 'cs'.*

- void add_generators_and_minimize (GenSys &gs)

  Adds the specified generators.

  Parameters:

  gs *The generators that will be added to the current system of generators. The parameter is not declared* `const` *because it can be modified.*

  Exceptions:

  std::invalid_argument *thrown if* `*this` *and* `gs` *are dimension-incompatible or if* `*this` *is empty and the the system of generators* `gs` *is not empty, but has no vertices.*

- void add_generators (GenSys &gs)

  Adds the specified generators without minimizing.

  Parameters:

  gs *The generators that will be added to the current system of generators. This parameter is not declared* `const` *because it can be modified.*

  Exceptions:

  std::invalid_argument *thrown if* `*this` *and* `gs` *are dimension-incompatible or if* `*this` *is empty and the system of generators* `gs` *is not empty, but has no vertices.*

- bool check_empty () const

  Returns `true` *if and only if* `*this` *is an empty polyhedron.*

- bool check_universe () const

  Returns `true` *if and only if* `*this` *is a universe polyhedron.*

- void swap (Polyhedron &y)

  Swaps `*this` *with polyhedron* `y`*. (Note that* `*this` *and* `y` *can be dimension-incompatible.).*

- bool is_empty () const

  Returns `true` *if and only if* `*this` *is an empty polyhedron.*

Friends

- bool Parma_Polyhedra_Library::operator<= (const Polyhedron &x, const Polyhedron &y)

  Returns `true` *if and only if polyhedron* `x` *is contained in polyhedron* `y`*.*

  Exceptions:

  std::invalid_argument *thrown if* `x` *and* `y` *are dimension-incompatible.*

- std::ostream & Parma_Polyhedra_Library::operator<< (std::ostream &s, const Polyhedron &p)

Output operator.

- std::istream & Parma_Polyhedra_Library::operator>> (std::istream &s, Polyhedron &p)

  Input operator.

Related Functions

(Note that these are not member functions.)

- bool operator== (const Polyhedron &x, const Polyhedron &y)

  Returns `true` *if and only if* `x` *and* `y` *are the same polyhedron.*

  Exceptions:

  std::invalid_argument *thrown if* `x` *and* `y` *are dimension-incompatible.*

- bool operator!= (const Polyhedron &x, const Polyhedron &y)

  Returns `true` *if and only if* `x` *and* `y` *are different polyhedra.*

  Exceptions:

  std::invalid_argument *thrown if* `x` *and* `y` *are dimension-incompatible.*

- bool operator< (const Polyhedron &x, const Polyhedron &y)

  Returns `true` *if and only if* `x` *is strictly contained in* `y`.

  Exceptions:

  std::invalid_argument *thrown if* `x` *and* `y` *are dimension-incompatible.*

- bool operator> (const Polyhedron &x, const Polyhedron &y)

  Returns `true` *if and only if* `x` *strictly contains* `y`.

  Exceptions:

  std::invalid_argument *thrown if* `x` *and* `y` *are dimension-incompatible.*

- bool operator>= (const Polyhedron &x, const Polyhedron &y)

  Returns `true` *if and only if* `x` *contains* `y`.

  Exceptions:

  std::invalid_argument *thrown if* `x` *and* `y` *are dimension-incompatible.*

6.8.1 Detailed Description

An object of the class Polyhedron represents a convex polyhedron in the vector space $\mathbb{R}^n$.

The dimension $n \in \mathbb{N}$ of the enclosing vector space is a key attribute of the polyhedron:

- all polyhedra, the empty ones included, are endowed with a specific space dimension;
- most operations working on a polyhedron and another object (i.e., another polyhedron, a constraint or generator, a set of variables, etc.) will throw an exception if the polyhedron and the object are dimension-incompatible (see the dimension-compatibility rules in the Introduction);
- the only ways to change the space dimension of a polyhedron are:
  - *explicit* calls to operators provided for that purpose;
  - standard copy, assignment and swap operators.

Note that two polyhedra can be defined on the zero-dimension space: the empty polyhedron and the universe polyhedron R^0.

A polyhedron can be specified as either a finite system of constraints or a finite system of generators (see Minkowski's theorem in the Introduction) and it is always possible to obtain either representation. That is, if we know the system of constraints, we can obtain from this the system of generators that define the same polyhedron and vice versa. These systems can contain redundant members: in this case we say that they are not in the minimal form.

In all the examples it is assumed that variables `x` and `y` are defined (where they are used) as follows:

```
Variable x(0);
Variable y(1);
```

Example 1

The following code builds a polyhedron corresponding to a square in $\mathbb{R}^2$, given as a system of constraints:

```
ConSys cs;
cs.insert(x >= 0);
cs.insert(x <= 3);
cs.insert(y >= 0);
cs.insert(y <= 3);
Polyhedron ph(cs);
```

The following code builds the same polyhedron as above, but starting from a system of generators specifying the four vertices of the square:

```
GenSys gs;
gs.insert(vertex(0*x + 0*y));
gs.insert(vertex(0*x + 3*y));
gs.insert(vertex(3*x + 0*y));
gs.insert(vertex(3*x + 3*y));
Polyhedron ph(gs);
```

Example 2

The following code builds an unbounded polyhedron corresponding to a half-strip in $\mathbb{R}^2$, given as a system of constraints:

```
ConSys cs;
cs.insert(x >= 0);
cs.insert(x - y <= 0);
cs.insert(x - y + 1 >= 0);
Polyhedron ph(cs);
```

The following code builds the same polyhedron as above, but starting from the system of generators specifying the two vertices of the polyhedron and one ray:

```
GenSys gs;
gs.insert(vertex(0*x + 0*y));
gs.insert(vertex(0*x + y));
gs.insert(ray(x - y));
Polyhedron ph(gs);
```

Example 3

The following code builds the polyhedron corresponding to an half-plane by adding a single constraint to the universe polyhedron in $\mathbb{R}^2$:

```
Polyhedron ph(2);
ph.insert(y >= 0);
```

The following code builds the same polyhedron as above, but starting from the empty polyhedron in the space $\mathbb{R}^2$ and inserting the appropriate generators (a vertex, a ray and a line).

```
Polyhedron ph(2, Polyhedron::EMPTY);
ph.insert(vertex(0*x + 0*y));
ph.insert(ray(y));
ph.insert(line(x));
```

Note that, even if the above polyhedron has no “proper” vertex, we must add one, because otherwise the result of the Minkowsky’s sum would be an empty polyhedron. To avoid subtle errors related to the minimization process, it is required that the first generator inserted in an empty polyhedron is a vertex (otherwise, an exception is thrown).

Example 4

The following code shows the use of the function `add_dimensions_and_embed`:

```
Polyhedron ph(1);
ph.insert(x == 2);
ph.add_dimensions_and_embed(1);
```

We build the universe polyhedron in the 1-dimension space $\mathbb{R}$. Then we add a single equality constraint, thus obtaining the polyhedron corresponding to the singleton set $\{2\} \subseteq \mathbb{R}$. After the last line of code, the resulting polyhedron is

$$\left\{ (2, x_1)^{\mathrm{T}} \in \mathbb{R}^2 \;\middle|\; x_1 \in \mathbb{R} \right\}.$$

Example 5

The following code shows the use of the function `add_dimensions_and_project`:

```
Polyhedron ph(1);
ph.insert(x == 2);
ph.add_dimensions_and_project(1);
```

The first two lines of code are the same as in Example 4 for `add_dimensions_and_embed`. After the last line of code, the resulting polyhedron is the singleton set $\left\{(2,0)^{\mathrm{T}}\right\} \subseteq \mathbb{R}^2$.

Example 6

The following code shows the use of the function `affine_image`:

```
Polyhedron ph(2, Polyhedron::EMPTY);
ph.insert(vertex(0*x + 0*y));
ph.insert(vertex(0*x + 3*y));
ph.insert(vertex(3*x + 0*y));
ph.insert(vertex(3*x + 3*y));
LinExpression coeff = x + 4;
ph.affine_image(x, coeff);
```

In this example the starting polyhedron is a square in $\mathbb{R}^2$, the considered variable is x and the affine expression is $x + 4$. The resulting polyhedron is the same square translated towards right. Moreover, if the affine transformation for the same variable `x` is $x + y$:

```
LinExpression coeff = x + y;
```

the resulting polyhedron is a parallelogram with the height equal to the side of the square and the oblique sides parallel to the line $x - y$. Instead, if we do not use an invertible transformation for the same variable; for example, the affine expression y:

```
LinExpression coeff = y;
```

the resulting polyhedron is a diagonal of the square.

Example 7

The following code shows the use of the function `affine_preimage`:

```
Polyhedron ph(2);
ph.insert(x >= 0);
ph.insert(x <= 3);
ph.insert(y >= 0);
ph.insert(y <= 3);
LinExpression coeff = x + 4;
ph.affine_preimage(x, coeff);
```

In this example the starting polyhedron, `var` and the affine expression and the denominator are the same as in Example 6, while the resulting polyhedron is again the same square, but translated towards left. Moreover, if the affine transformation for `x` is $x + y$

```
LinExpression coeff = x + y;
```

the resulting polyhedron is a parallelogram with the height equal to the side of the square and the oblique sides parallel to the line $x + y$. Instead, if we do not use an invertible transformation for the same variable `x`, for example, the affine expression y:

```
LinExpression coeff = y;
```

the resulting polyhedron is a line that corresponds to the y axis.

Example 8

For this example we use also the variables:

```
Variable z(2);
Variable w(3);
```

The following code shows the use of the function `remove_dimensions`:

```
GenSys gs;
gs.insert(vertex(3*x + y +0*z + 2*w));
Polyhedron ph(gs);
set<Variable> to_be_removed;
to_be_removed.insert(y);
to_be_removed.insert(z);
ph.remove_dimensions(to_be_removed);
```

The starting polyhedron is the singleton set $\{(3, 1, 0, 2)^{\mathrm{T}}\} \subseteq \mathbb{R}^4$, while the resulting polyhedron is $\{(3, 2)^{\mathrm{T}}\} \subseteq \mathbb{R}^2$. Be careful when removing dimensions *incrementally*: since dimensions are automatically renamed after each application of the `remove_dimensions` operator, unexpected results can be obtained. For instance, by using the following code we would obtain a different result:

```
set<Variable> to_be_removed1;
to_be_removed1.insert(y);
ph.remove_dimensions(to_be_removed1);
set<Variable> to_be_removed2;
to_be_removed2.insert(z);
ph.remove_dimensions(to_be_removed2);
```

In this case, the result is the polyhedron $\{(3, 0)^{\mathrm{T}}\} \subseteq \mathbb{R}^2$: when removing the set of dimensions `to_be_removed2` we are actually removing variable w of the original polyhedron. For the same reason, the operator `remove_dimensions` is not idempotent: removing twice the same set of dimensions is never a no-op.

6.8.2 Member Enumeration Documentation

6.8.2.1 enum Parma_Polyhedra_Library::Polyhedron::Degenerate_Kind

Enumeration values:

UNIVERSE The universe polyhedron, i.e., the whole vector space.

EMPTY The empty polyhedron, i.e., the empty set.

6.9 Parma_Polyhedra_Library::Variable Class Reference

A dimension of the space.

Public Methods

- Variable (unsigned int id)

  Constructor: `id` *is the index of the Cartesian axis.*

- unsigned int id () const

  Returns the index of the Cartesian axis.

Related Functions

(Note that these are not member functions.)

- std::ostream & operator<< (std::ostream &s, const Parma_Polyhedra_Library::Variable &v)

  Output operator.

- bool operator< (const Variable &v, const Variable &w)

  Defines a total ordering on variables.

6.9.1 Detailed Description

An object of the class Variable represents a dimension of the space, that is one of the Cartesian axes. Variables are used as base 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).

Note that the "meaning" of an object of the class Variable is completely specified by the integer index provided to its constructor: be careful not to be mislead by C++ language variable names. For instance, in the following example the linear expressions `e1` and `e2` are equivalent, since the two variables `x` and `z` denote the same Cartesian axis.

```
Variable x(0);
Variable y(1);
Variable z(0);
LinExpression e1 = x + y;
LinExpression e2 = y + z;
```

7 PPL Page Documentation

7.1 Prolog Interface

7.1.1 Introduction

This Prolog library is an interface to the PPL and provides Prolog operations for creating and manipulating the PPL polyhedra.

7.1.2 System-Dependent Features

CIAO Prolog

Support for CIAO Prolog is under development and will be available in a future release.

GNU Prolog

Support for GNU Prolog is under development and will be available in a future release.

SICStus Prolog

In order to use the library you should load `ppl_sicstus.pl`.

SWI Prolog

Support for SWI Prolog is under development and will be available in a future release.

7.1.3 System-Independent Features

The PPL predicates provided for the Prolog interface are specified below.

The specification uses the following grammar rules:

```
VarId      -->   non-negative integer  variable identifier

PPL_Var    -->   '$VAR'(VarId)         PPL variable

LinExpr    -->    PPL_Var              PPL variable
                | number               integer
                | + LinExpr            unary plus
                | - LinExpr            unary minus
                | LinExpr + LinExpr    addition
                | LinExpr - LinExpr    subtraction
                | number * LinExpr     multiplication
                | LinExpr * number     multiplication

Constraint -->    LinExpr = LinExpr    equation
                | LinExpr =< LinExpr   nonstrict inequation
                | LinExpr >= LinExpr   nonstrict inequation

Generator -->     vertex(LinExpr)      vertex
                | vertex(LinExpr,Int)  vertex
                                       (Int is the denominator so that the
```

```
                              vertex is defined by Expr/Int)
           | ray(LinExpr)         ray
           | line(LinExpr)        line
```

There are a few general rules that need to be followed when using the PPL predicates.

- Argument positions labeled as `+Address` must be addresses in memory. It is up to the programmer to ensure that these addresses refer to a PPL polyhedron.
- A free variable may be bound to an address of a PPL polyhedron by using either `ppl_new_polyhedron/2` or `ppl_copy_polyhedron/2`.
- Memory occupied by a PPL polyhedron should be released as soon as it is no longer required. This can be done by executing `ppl_delete_polyhedron/1`. To understand why this is important, consider a Prolog program and a variable that is bound to a Herbrand term. When the variable dies (goes out of scope) or is uninstantiated (on backtracking) the term it is bound to is amenable to garbage collection. But this only applies for the standard domain of the language: Herbrand terms. In Prolog+PPL, addresses of PPL polyhedra are just integers. When variables bound to addresses (i.e. integers) die or are uninstantiated, these variables will be garbage-collected, but the polyhedra to which the addresses refer will not be released.
- For a PPL polyhedron with space dimension `k`, the variables used for defining the constraints and the generators must have the form, `'$VAR'(0), ..., '$VAR'(k-1).` Note that the variable identifiers must be strictly less than `k`.
- When using the predicates that combine PPL polyhedra or insert constraints or generators into a PPL polyhedron, the polyhedra referenced and any constraints or generators in the call should follow all the rules stated in the dimension-compatibility paragraph in the Introduction.
- Note that the vertex specified in the grammar rule for `Generator` is not necessarily a vertex of a polyhedron. As observed in the Introduction, a set of generators representing a polyhedron P often have to include points in P that are *not* vertices of P. Hence, it is convenient to use *vertex* to mean any element of the set of generators V even though it may not be a "proper" vertex of P.

See the specifications of individual predicates for examples and more information regarding these issues.

`ppl_new_polyhedron(-Address, +Integer)`

Creates a new universe polyhedron with `Integer` dimensions with reference `Address`. Thus the query

```
| ?- ppl_new_polyhedron(X, 3).
```

creates the polyhedron defining the 3-dimensional vector space $\mathbb{R}^3$ with `X`, the reference address.

The first argument must be a free variable and if it is already instantiated to an address, it will fail. E.g., the following will fail:

```
| ?- ppl_new_polyhedron(X,3),
     ppl_new_polyhedron(X,3).
```

`ppl_new_empty_polyhedron(-Address, +Integer)`

Creates a new empty polyhedron with `Integer` dimensions with reference address `Address`. Thus the query

```
| ?- ppl_new_empty_polyhedron(X, 3).
```

creates an empty polyhedron embedded in $\mathbb{R}^3$ with `X` the reference address.

The first argument must be a free variable.

`ppl_copy_polyhedron(+Address1, -Address2)`

The polyhedron referenced by `Address1` is copied to `Address2`.

`ppl_delete_polyhedron(+Address)`

Deletes the polyhedron referenced by `Address`.

If `Address` is not an address of a polyhedron, execution is aborted. E.g., the following aborts:

```
| ?-ppl_new_polyhedron(X,3),
    ppl_delete_polyhedron(X),
    ppl_delete_polyhedron(X).
```

`ppl_space_dimension(+Address, -Integer+)`

There must be a polyhedron `P` referenced by `Address`. Returns in `Integer` the space dimension of `P`.

If `Address` is not an address of a polyhedron, execution succeeds with a large but unspecified dimension. E.g.,

```
| ?- ppl_new_polyhedron(X,5),
     ppl_delete_polyhedron(X),
     ppl_space_dimension(X,K).

  K = 136896952,
  X = -32966754 ?
```

`ppl_insert_constraint(+Address, +Constraint)`

Adds the constraint `Constraint` to the polyhedron referenced by `Address`. Thus after the query

```
| ?- A = '$VAR'(0), B = '$VAR'(1), C = '$VAR'(2),
     ppl_new_polyhedron(X, 3),
     ppl_insert_constraint(X, 4*A + B - 2*C >= 5).
```

the polyhedron referenced by `X` is defined to be the set of points in the vector space $\mathbb{R}^3$ satisfying the constraint $4x + y - 2z >= 5$.

The constraint `Constraint` and the polyhedron referenced by `Address` must be dimensional compatible. This means that the identifiers for the variables in `Constraint` must be strictly less than the space dimension of the polyhedron. E.g.,

```
| ?- ppl_new_polyhedron(X,3),
     ppl_insert_constraint(X,'$VAR'(3) = -12).
{ERROR: 'PPL::Polyhedron::insert(c):
this->space_dimension() == 3, y->space_dimension() == 4'}
```

ppl_insert_generator(+Address, +Generator)

Adds the generator `Generator` to the polyhedron `P` referenced by `Address`. Thus after the query

```
| ?- A = '$VAR'(0), B = '$VAR'(1), C = '$VAR'(2),
     ppl_new_polyhedron(X, 3),
     ppl_insert_generator(X, vertex(-100*A - 5*B, 8)).
```

the polyhedron referenced by `X` is defined to be single vertex $(-12.5, -0.625, 0)^{\mathrm{T}}$ in the vector space $\mathbb{R}^3$.

As for `ppl_insert_constraint`, the identifiers for the variables in `Generator` must be strictly less than the space dimension of the polyhedron referenced by `Address`.

ppl_insert_constraints(+Address, +List_of_Constraints)

Adds the constraints in list `List_of_Constraints` to the polyhedron referenced by `Address`. Constraints are not minimized and a query will succeed even when `List_of_Constraints` is unsatisfiable. E.g.,

```
| ?- A = '$VAR'(0), B = '$VAR'(1),
     ppl_new_polyhedron(X, 2),
     ppl_insert_constraints(X, [4*A + B >= 3, A = 1]),
     ppl_get_constraints(X,CS).

A = A,
B = B,
X = -32975498,
CS = [4*A+1*B>=3,1*A=1] ?

yes
| ?- A = '$VAR'(0), B = '$VAR'(1),
     ppl_new_polyhedron(X, 2),
     ppl_insert_constraints(X, [4*A + B >= 3, A = 1]),
     ppl_get_constraints(X,CS).

A = A,
B = B,
X = -32975104,
CS = [4*A+1*B>=3,1*A=1],
GS = [vertex(1*A+ -1*B),ray(1*B)] ?

yes
```

ppl_add_constraints_and_minimize(+Address, +List_of_Constraints)

Adds the constraints in list `List_of_Constraints` to the polyhedron referenced by `Address`. This will fail if the resulting polyhedron is empty. E.g.,

```
| ?- A = '$VAR'(0), B = '$VAR'(1),
     ppl_new_polyhedron(X, 2),
     ppl_add_constraints_and_minimize(X, [4*A + B >= 3, A = 1]),
     ppl_get_constraints(X,CS).
```

```
A = A,
B = B,
X = -33291612,
CS = [1*B>= -1,1*A=1] ?

yes
| ?- A = '$VAR'(0), B = '$VAR'(1),
    ppl_new_polyhedron(X, 2),
    ppl_add_constraints_and_minimize(X, [4*A + B >= 3, A = 0, B =< 0]),
    ppl_get_constraints(X,CS).
| ?-
no
```

ppl_insert_generators(+Address, +List_of_Generators)

Adds the generators in list `List_of_Generators` to the polyhedron referenced by `Address` in list order.

Note that, as explained in the paragraph on generator representation in the Introduction, a non-empty polyhedron must always have a vertex as as one of its generators. Thus care must be taken to ensure that before calling this predicate that either the polyhedron referenced by `Address` is non-empty or that whenever `List_of_Generators` is non-empty the first element defines a vertex.

ppl_remove_dimensions(+Address, +List_of_PPL_Vars)

The dimensions corresponding to the identifiers of the variables in list `List_of_PPL_Vars` are removed from the polyhedron referenced by `Address`. E.g.,

```
| ?- A='$VAR'(0), B = '$VAR'(1), C = '$VAR'(2),
     ppl_new_polyhedron(X, 3),
     ppl_remove_dimensions(X, [B]),
     ppl_space_dimension(X,K),
     ppl_get_generators(X,GS).
A = A,
B = B,
C = C,
K = 2,
X = -32974400,
GS = [vertex(0),line(1*A),line(1*B),line(0)] ?
```

Note that as can be seen from this example, the identifiers for the remaining variables are renumbered so that they are consecutive and the maximum index is less than the number of dimensions.

ppl_remove_higher_dimensions(+Address, +Integer))

Projects the polyhedron referenced by address `Address` onto the first `Integer` dimensions. Thus, if the polyhedron `P` at the `Address` has space dimension k, `Integer` must be less than or equal to k. E.g.,

```
| ?- ppl_new_polyhedron(X,5),
     ppl_remove_higher_dimensions(X,3),
     ppl_space_dimension(X,K).

K = 3,
X = -33292540 ?

yes
```

```
| ?- ppl_new_polyhedron(X,5),
     ppl_remove_higher_dimensions(X,6),
     ppl_space_dimension(X,K).
{ERROR: 'void PPL::Polyhedron::remove_higher_dimensions(nd):
this->space_dimension() == 5, requested dimension == 6'}
```

ppl_add_dimensions_and_embed(+Address, +Integer)

Adds `Integer` new dimensions and embeds the old polyhedron referenced by `Address` in the new space. E.g.,

```
| ?- ppl_new_polyhedron(X,0),
     ppl_add_dimensions_and_embed(X,2),
     ppl_get_constraints(X,CS),
     ppl_get_generators(X,GS).

X = -32775690,
CS = [],
GS = [vertex(0),line(1*A),line(1*B)] ?

yes
```

ppl_add_dimensions_and_project(+Address, +Integer)

Adds `Integer` new dimensions and does not embed the old polyhedron referenced by `Address` in the new space. E.g.,

```
| ?- ppl_new_polyhedron(X,0),
     ppl_add_dimensions_and_project(X,2),
     ppl_get_constraints(X,CS),
     ppl_get_generators(X,GS).

X = -32920674,
CS = [1*A=0,1*B=0],
GS = [vertex(0)] ?

yes
```

ppl_check_empty(+Address)

Succeeds if and only if the polyhedron referenced by `Address` is empty.

ppl_get_constraints(+Address, -List_of_Constraints)

Binds `List_of_Constraints` to the system of constraints defining the polyhedron at `Address`. E.g.,

```
| ?- A = '$VAR'(0), B = '$VAR'(1), C = '$VAR'(2),
     ppl_new_polyhedron(X, 3),
     ppl_insert_constraint(X, 4*A+B-2*C >= 5),
     ppl_get_constraints(X, CS),
     write(CS).
[4*A+1*B+ -2*C>=5]
A = A,
B = B,
C = C,
X = -32975760,
CS = [4*A+1*B+ -2*C>=5] ?
```

`ppl_get_generators(+Address, -List_of_Generators)`

Binds `List_of_Generators` to the system of generators defining the polyhedron at `Address`. E.g.,

```
| ?- A = '$VAR'(0), B = '$VAR'(1), C = '$VAR'(2),
     ppl_new_polyhedron(X, 3),
     ppl_insert_constraint(X, 4*A+B-2*C >= 5),
     ppl_get_generators(X, GS),
     write(GS).
A = A,
B = B,
C = C,
X = -32975734,
GS = [ray(-1*C),vertex(-5*C,2),line(-2*B+ -1*C),line(-1*A+ -2*C)] ?
```

`ppl_intersection_assign(+Address_1, +Address_2)`

Computes the intersection of the polyhedra referenced by `Address_1` and `Address_2` and places the result at `Address_1`.

`ppl_convex_hull_assign(+Address_1, +Address_2)`

Computes the convex hull of the polyhedra referenced by `Address_1` and `Address_2` and places the result at `Address_1`.

`ppl_convex_difference_assign(+Address_1, +Address_2)`

Computes the convex hull of the set-theoretic difference of the polyhedra referenced by `Address_1` and `Address_2` and places the result at `Address_1`.

`ppl_widening_assign(+Address_1, +Address_2)`

Computes the widening between the polyhedra referenced by `Address_1` and `Address_2` and places the result at `Address_1`.

7.2 GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Preamble

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

Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) 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 this service 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 make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. 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.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

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

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The ”Program”, below, refers to any such program or work, and a ”work based on the Program” means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term ”modification”.) Each licensee is addressed as ”you”.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute 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 and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

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

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

- **a)** You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
- **b)** You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
- **c)** If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

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

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

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

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

- **a)** Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- **b)** Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- **c)** Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

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

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program 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.

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

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

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

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

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

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

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

9. The Free Software Foundation may publish revised and/or new versions of the 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 a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

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

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, 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.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE 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.

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 convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
 one line to give the program's name and an idea of what it does.
Copyright (C)  yyyy   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 2
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, write to the Free Software
Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
```

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

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

```
Gnomovision version 69, Copyright (C)  year  name of author
```

```
Gnomovision 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 <SAMP>‘show w’</SAMP> and <SAMP>‘show c’</SAMP> should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than <SAMP>‘show w’</SAMP> and <SAMP>‘show c’</SAMP>; they could even be mouse-clicks or menu items–whatever suits your program.

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

```
Yoyodyne, Inc., hereby disclaims all copyright
interest in the program ‘Gnomovision’
(which makes passes at compilers) written
by James Hacker.

 signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice
```

This 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 Library General Public License instead of this License.

7.3 GNU Free Documentation License

Version 1.1, March 2000

```
Copyright (C) 2000  Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.
```

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written 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 that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The ”Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as ”you”.

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. (For example, 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.

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 ”Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and 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 has been designed to thwart or discourage subsequent modification by readers is not Transparent. 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 designed for human modification. Opaque formats include PostScript, PDF, 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 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.

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 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 publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. 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 less than five).
- **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”, and 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.** In any section entitled ”Acknowledgements” or ”Dedications”, preserve the section's title, 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 as ”Endorsements” or to conflict in title with any Invariant Section.

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.

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, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an ”aggregate”, and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they 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 quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around 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 provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

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 1.1
or any later version published by the Free Software Foundation;
with the Invariant Sections being LIST THEIR TITLES, with the
Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.
A copy of the license is included in the section entitled "GNU
Free Documentation License".
```

If you have no Invariant Sections, write ”with no Invariant Sections” instead of saying which ones are invariant. If you have no Front-Cover Texts, write ”no Front-Cover Texts” instead of ”Front-Cover Texts being LIST”; likewise for Back-Cover Texts.

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.

Index