Possibly Not Closed Convex Polyhedra and the Parma Polyhedra Library*

Roberto Bagnara[1], Elisa Ricci[1], Enea Zaffanella[1], and Patricia M. Hill[2]

[1] Department of Mathematics, University of Parma, Italy
{bagnara,ericci,zaffanella}@cs.unipr.it
[2] School of Computing, University of Leeds, UK
hill@comp.leeds.ac.uk

Abstract. The domain of convex polyhedra is employed in several systems for the analysis and verification of hardware and software components. Current applications span imperative, functional and logic languages, synchronous languages and synchronization protocols, real-time and hybrid systems. Since the seminal work of P. Cousot and N. Halbwachs, convex polyhedra have thus played an important role in the formal methods community and several critical tasks rely on their software implementations. Despite this, existing libraries for the manipulation of convex polyhedra are still research prototypes and suffer from limitations that make their usage problematic, especially in critical applications. Furthermore, there is inadequate support for polyhedra that are not necessarily closed (NNC), i.e., polyhedra that are described by systems of constraints where strict inequalities are allowed to occur. This paper presents the Parma Polyhedra Library, a new, robust and complete implementation of NNC convex polyhedra, concentrating on the distinctive features of the library and on the novel theoretical underpinnings.

Dedicated to the memory of Hervé Le Verge

1 Introduction

Convex polyhedra are regions of some n-dimensional space that are bounded by a finite set of hyperplanes. A convex polyhedron in $\mathbb{R}^n$ describes a relation between n real-valued quantities. The seminal work of P. Cousot and N. Halbwachs [15] introduced the use of convex polyhedra as a domain of descriptions to solve, by *abstract interpretation* [13], a number of important data-flow analysis problems such as array bound checking, compile-time overflow detection, loop invariant computations and loop induction variables. Convex polyhedra are also used, among many other applications, for the analysis and verification of

* This work has been partly supported by MURST projects "Abstract Interpretation, type systems and control-flow analysis" and "Aggregate- and number-reasoning for computing: from decision algorithms to constraint programming with multisets, sets, and maps" and by EPSRC grant GR/R53401/01.

synchronous languages [6, 18] and of linear hybrid automata (an extension of finite-state machines that models time requirements) [20, 22], for the computer-aided formal verification of concurrent and reactive systems based on temporal specifications [27], for inferring argument size relationships in logic languages [5], and for the automatic parallelization of imperative programs [30]. Since the work of Cousot and Halbwachs, convex polyhedra have thus played an important role in the formal methods community and new uses continue to emerge (see, e.g., [11, 16]). As a consequence, several critical tasks, such as checking the correctness of synchronization protocols or verifying the absence of run-time errors of systems whose failure can cause serious damage, rely on the software implementations of convex polyhedra.

Traditionally, convex polyhedra are assumed to be topologically closed and described by constraint systems containing linear equations and non-strict linear inequalities. However, some tasks need to use convex polyhedra that are not necessarily closed (NNC), i.e., polyhedra that are described by constraint systems possibly containing strict linear inequalities (in addition to equations and non-strict inequalities). Strict inequalities are important, for instance, in order to directly represent non-intersecting temporal regions, as is often the case when modeling applications where synchronization protocols, asynchronous interactions and temporal constraints come into play. Recently, they have also been used for the automatic computation of linear ranking functions [11].

Prior to the release of the Parma Polyhedra Library (PPL) [2] which is the subject of this paper, four libraries for the manipulation of convex polyhedra were (and continued to be) available:[3]

1. Polylib, designed and written by H. Le Verge and D. K. Wilde [25, 32];
2. *PolyLib*, the successor of the library by Le Verge and Wilde [26];
3. New Polka, by B. Jeannet [24];
4. the polyhedra library that comes with the HyTech tool [22].

All libraries,[4] including the PPL, use the same basic technique for representing convex polyhedra (the *double description method* recalled in Section 3) and the same core algorithm for manipulating that representation (the implementation and extension of N. V. Chernikova's algorithms [8–10] by Le Verge [25], possibly with the improvements of K. Fukuda and A. Prodon [17]).

Apart from the PPL, only New Polka supports NNC polyhedra. However, this support is incomplete, incurring avoidable inefficiencies and leaving the client application with the non-trivial task of a correct interpretation of the results.

Libraries 1 and 4 may incur overflow problems whereas 2 and 3 can use unbounded integers as coefficients. Libraries 2 and 3 can be configured to use finite integral types for extra speed but this, of course, comes with the possibility of overflows.

---

[3] We restrict ourselves to those libraries that are freely available and provide the services required by applications in static analysis and computer-aided verification.

[4] We refer to the following versions that, at the time of writing, are the latest available: Polylib 2.1, *PolyLib* 5.0.4, New Polka 1.1.3c, HyTech 1.04f.

In libraries 1–4, matrices of coefficients, which are the main data structures used to represent polyhedra, cannot grow dynamically and the client application is ultimately responsible for specifying their dimensions. Since the worst case space complexity of the methods employed is exponential, in general the client application cannot make a safe and practical choice: specifying small dimensions may provoke a run-time failure; generous dimensions may waste significant amounts of memory and, again, result in unnecessary run-time failures.

The problems caused by run-time errors such as overflow and memory allocation failure could be mitigated or even solved by suitable mechanisms for error detection, handling and recovery. This requires the ability to detect the problem, releasing any affected data-structure whether it be completely or only partially constructed,and continue the computation with an alternative method (e.g., by reverting to an interval-based approximation). Library 1 detects some errors and sets an error flag whereas libraries 3 and 4 detect some errors, print an error message and abort. Library 2 detects more errors, sometimes setting a flag and sometimes printing a message and aborting.

Libraries 1–3 are free software released under the GNU General Public License (GPL, see `http://www.gnu.org/`). Thus, when faced with the need to overcome the above mentioned limitations, anyone can freely take and use them as the basis for further development. However, it appears that none of the libraries provide documentation for the interfaces and code that is adequate for an outsider to make such improvements with any real confidence. This feeling of insecurity is aggravated by the discovery of some errors and imprecisions in the theoretical sections of the documentation of some libraries (for more information see [3], which itself contains an error, now corrected in [4]). Moreover, a complete solution to issues such as error recovery and fully dynamic memory allocation requires a somewhat radical departure from the existing code bases.

For all these reasons we decided to write the PPL, a robust and complete implementation of convex polyhedra. This paper describes the library concentrating on some of its distinctive features and the novel theoretical underpinnings.

The plan of the paper is as follows: Section 2 recalls some basic notation and terminology; Section 3 introduces convex polyhedra and the double description method paying attention to common pitfalls; Section 4 presents the NNC polyhedra and describes (from a theoretical perspective) how they are handled in the PPL; Section 5 briefly describes the design and implementation of the PPL and how it addresses all the limitations we have just discussed; Section 6 concludes.

A longer version of this paper, containing the proofs of the results presented here, is available as a technical report [3].

2 Preliminaries

In this paper, all topological arguments refer to the topological space $\mathbb{R}^n$ with the standard topology. The *topological closure* of $S \subseteq \mathbb{R}^n$ is denoted by $\mathbb{C}(S)$ and defined as $\mathbb{C}(S) \stackrel{\mathrm{def}}{=} \bigcap \{ C \subseteq \mathbb{R}^n \mid S \subseteq C \text{ and } C \text{ is closed} \}$. We denote the set of all non-negative reals by $\mathbb{R}_+$. For each $i \in \{1, \dots, n\}$, v_i denotes the i-th component of the (column) vector $\boldsymbol{v} \in \mathbb{R}^n$. We denote by $\mathbf{0}$ the vector of $\mathbb{R}^n$, called *the origin*, having all components equal to zero. A vector $\boldsymbol{v} \in \mathbb{R}^n$ can also be interpreted as a matrix in $\mathbb{R}^{n\times 1}$ and manipulated accordingly using the usual definitions for addition, multiplication (both by a scalar and by another matrix), and transposition, which is denoted by $\boldsymbol{v}^{\mathrm{T}}$. The *scalar product* of $\boldsymbol{v}, \boldsymbol{w} \in \mathbb{R}^n$, denoted $\langle \boldsymbol{v}, \boldsymbol{w} \rangle$, is the real number $\boldsymbol{v}^{\mathrm{T}}\boldsymbol{w} = \sum_{i=1}^{n} v_i w_i$. For any relational operator $\bowtie \in \{=, \geq, \leq, <, >\}$, we write $\boldsymbol{v} \bowtie \boldsymbol{w}$ to denote the conjunctive proposition $\bigwedge_{i=1}^{n}(v_i \bowtie w_i)$. In contrast, $\boldsymbol{v} \neq \boldsymbol{w}$ will denote the proposition $\neg(\boldsymbol{v} = \boldsymbol{w})$. We will sometimes use the convenient notation $a \bowtie_1 b \bowtie_2 c$ to denote the conjunction $a \bowtie_1 b \wedge b \bowtie_2 c$ and we will not distinguish conjunctions of propositions from sets of propositions. For each set $S \subseteq \mathbb{R}^n$ of finite cardinality m, we denote by $\operatorname{matrix}(S) \subseteq \mathbb{R}^{n\times m}$ the set of all matrices having S as the set of their columns.

3 Convex Polyhedra and the Double Description Method

For each vector $\boldsymbol{a} \in \mathbb{R}^n$ and scalar $b \in \mathbb{R}$, where $\boldsymbol{a} \neq \mathbf{0}$, the linear (non-strict) inequality constraint $\langle \boldsymbol{a}, \boldsymbol{x} \rangle \geq b$ defines a closed affine half-space. The linear equality constraint $\langle \boldsymbol{a}, \boldsymbol{x} \rangle = b$ defines an affine hyperplane. Convex polyhedra are usually described as finite systems of linear equality and inequality constraints. When working at the theoretical level, it is simpler to express each equality constraint as the intersection of the two half-spaces $\langle \boldsymbol{a}, \boldsymbol{x} \rangle \geq b$ and $\langle -\boldsymbol{a}, \boldsymbol{x} \rangle \geq -b$.

Definition 1. (Closed polyhedron.) *The set $\mathcal{P} \subseteq \mathbb{R}^n$ is a* closed polyhedron *if and only if either $\mathcal{P}$ can be expressed as the intersection of a finite number of closed affine half-spaces of $\mathbb{R}^n$, or $n = 0$ and $\mathcal{P} = \varnothing$.*

Alternatively, the definition of a convex polyhedron can be based on some of the geometric features of the set of solutions of such a system of constraints. A vector $\boldsymbol{r} \in \mathbb{R}^n$ such that $\boldsymbol{r} \neq \mathbf{0}$ is a *ray* (or *direction of infinity*) of a non-empty polyhedron $\mathcal{P} \subseteq \mathbb{R}^n$ if, for every point $\boldsymbol{p} \in \mathcal{P}$ and every $\lambda \in \mathbb{R}_+$, it holds $\boldsymbol{p}+\lambda\boldsymbol{r} \in \mathcal{P}$; a vector $\boldsymbol{l} \in \mathbb{R}^n$ is a *line* of $\mathcal{P}$ if both $\boldsymbol{l}$ and $-\boldsymbol{l}$ are rays of $\mathcal{P}$. The empty polyhedron has no rays and no lines. As was the case for equality constraints, the theory can dispense with the use of lines by using the corresponding pair of opposite rays. The following theorem is a simple consequence of well known theorems by Minkowski and Weyl [31].

Theorem 1. *The set $\mathcal{P} \subseteq \mathbb{R}^n$ is a closed polyhedron if and only if there exist finite sets $R, P \subseteq \mathbb{R}^n$ of cardinality k and ℓ, respectively, such that $\mathbf{0} \notin R$ and, for any matrices $K \in \mathbb{R}^{n\times k}$ and $L \in \mathbb{R}^{n\times \ell}$ where $K \in \operatorname{matrix}(R)$ and $L \in \operatorname{matrix}(P)$,*

$$\mathcal{P} = \left\{ K\boldsymbol{\mu} + L\boldsymbol{\nu} \in \mathbb{R}^n \;\middle|\; \boldsymbol{\mu} \in \mathbb{R}_+^k, \boldsymbol{\nu} \in \mathbb{R}_+^\ell, \textstyle\sum_{i=1}^{\ell} \nu_i = 1 \right\}.$$

When $\mathcal{P} \neq \varnothing$, we say that $\mathcal{P}$ is described by the *generator system* $\mathcal{G} = (R, P)$. In particular, the vectors of R and P are rays and points of $\mathcal{P}$, respectively. Informally speaking, if no "supporting point" is provided then an empty polyhedron is obtained; formally, $\mathcal{P} = \varnothing$ if and only if $P = \varnothing$. By convention, the empty system (i.e., the system with $R = \varnothing$ and $P = \varnothing$) is the only generator system for the empty polyhedron. It is worth stressing that, in general, the vectors in R and P are not the *extreme rays* and the *vertices* of the polyhedron [3]: for instance, any half-space of $\mathbb{R}^2$ has two extreme rays and no vertices, but any generator system describing it will contain at least three rays and one point.

The combination of the two approaches outlined above is the basis of the *Double Description* (DD) method [29], which exploits the duality principle to compute each representation starting from the other one, possibly minimizing the descriptions. We will write $\mathrm{con}(\mathcal{C})$ and $\mathrm{gen}(\mathcal{G})$ to denote the polyhedra described by the finite constraint system $\mathcal{C}$ and generator system $\mathcal{G}$, respectively.

Definition 2. (DD pair and minimal forms.) *If* $\mathrm{con}(\mathcal{C}) = \mathrm{gen}(\mathcal{G}) = \mathcal{P}$*, then* $(\mathcal{C}, \mathcal{G})$ *is said to be a* DD pair *for* $\mathcal{P}$*, and we write* $(\mathcal{C}, \mathcal{G}) \equiv \mathcal{P}$*. We say that*

- $\mathcal{C}$ *is in* minimal form *if there does not exist* $\mathcal{C}' \subset \mathcal{C}$ *such that* $\mathrm{con}(\mathcal{C}') = \mathcal{P}$*;*
- $\mathcal{G} = (R, P)$ *is in* minimal form *if there does not exist* $\mathcal{G}' = (R', P')$ *such that* $\mathcal{G}' \neq \mathcal{G}$*,* $R' \subseteq R$*,* $P' \subseteq P$ *and* $\mathrm{gen}(\mathcal{G}') = \mathcal{P}$*;*
- *the DD pair* $(\mathcal{C}, \mathcal{G})$ *is in* minimal form *if* $\mathcal{C}$ *and* $\mathcal{G}$ *are both in minimal form.*

The set of all closed polyhedra on the vector space $\mathbb{R}^n$, denoted $\mathbb{CP}_n$, can be partially ordered by set-inclusion to form a lattice having the empty set and $\mathbb{R}^n$ as the bottom and top element, respectively. The binary meet operation is thus given by set-intersection, which is easily implemented by taking the union of the constraint systems representing the two arguments. The binary join operation, denoted $\uplus$ and called *convex polyhedral hull* (poly-hull, for short), is implemented by taking the union of the two arguments' generator systems. Note that, in general, the poly-hull of two polyhedra is different from their convex hull [31].

4 Handling Not Necessarily Closed Polyhedra

For each vector $\boldsymbol{a} \in \mathbb{R}^n$ and scalar $b \in \mathbb{R}$, where $\boldsymbol{a} \neq \boldsymbol{0}$, the linear strict inequality constraint $\langle \boldsymbol{a}, \boldsymbol{x} \rangle > b$ defines an open affine half-space. By allowing strict inequalities to occur in the system of constraints, it is possible to define polyhedra that are not necessarily closed (NNC polyhedra, for short).

Definition 3. (NNC polyhedron.) *The set* $\mathcal{P} \subseteq \mathbb{R}^n$ *is a* NNC polyhedron *if and only if either* $\mathcal{P}$ *can be expressed as the intersection of a finite number of (not necessarily closed) affine half-spaces of* $\mathbb{R}^n$*, or* $n = 0$ *and* $\mathcal{P} = \varnothing$*.*

We denote by $\mathbb{P}_n$ the set of all NNC polyhedra on the vector space $\mathbb{R}^n$. Obviously, we have $\mathbb{CP}_n \subseteq \mathbb{P}_n$ (note that $\mathbb{CP}_n = \mathbb{P}_n$ if and only if $n = 0$). When partially ordered by set-inclusion, $\mathbb{P}_n$ is a lattice and $\mathbb{CP}_n$ is a sublattice of $\mathbb{P}_n$.

To the best of the authors' knowledge, the first software library (based on the DD method) allowing for the computation over the domain $\mathbb{P}_n$ was the Polka library [20], where each NNC polyhedron $\mathcal{P} \in \mathbb{P}_n$ is embedded into a closed polyhedron $\mathcal{R} \in \mathbb{CP}_{n+1}$. The additional dimension of the vector space, usually labeled by the letter ϵ, encodes the topological closeness of each affine half-space in the constraint description for $\mathcal{P}$. Namely, if $\mathcal{P} = \mathrm{con}(\mathcal{C})$, where

$$\mathcal{C} = \bigl\{ \langle \boldsymbol{a}_i, \boldsymbol{x} \rangle \bowtie_i b_i \bigm| i \in \{1, \dots, m\}, \boldsymbol{a}_i \in \mathbb{R}^n, \bowtie_i \in \{\geq, >\}, b_i \in \mathbb{R} \bigr\},$$

then the representation polyhedron is defined as $\mathcal{R} = \mathrm{con}\bigl(\mathrm{con\_repr}(\mathcal{C})\bigr)$, where

$$\begin{aligned} \mathrm{con\_repr}(\mathcal{C}) \stackrel{\mathrm{def}}{=} \bigl\{0 \leq \epsilon \leq 1\bigr\} & \\ \cup \bigl\{ \langle \boldsymbol{a}_i, \boldsymbol{x} \rangle - 1 \cdot \epsilon \geq b_i \bigm| i \in \{1, \dots, m\}, \bowtie_i \in \{>\} \bigr\} & \\ \cup \bigl\{ \langle \boldsymbol{a}_i, \boldsymbol{x} \rangle + 0 \cdot \epsilon \geq b_i \bigm| i \in \{1, \dots, m\}, \bowtie_i \in \{\geq\} \bigr\}. & \end{aligned}$$

It should be stressed that the choice of the value -1 for the coefficients of the additional variable ϵ in the constraints representing strict inequalities is rather arbitrary: any other negative value will do. Similarly, the side constraint $\epsilon \leq 1$ could be replaced by any other *ϵ-upper-bound* constraint, i.e., by any constraint $\epsilon \leq \delta$ such that $\delta > 0$. Different, though equivalent, constraint systems $\mathcal{C}_j$ describing $\mathcal{P}$ may be embedded into different representation polyhedra $\mathcal{R}_j = \mathrm{con}\bigl(\mathrm{con\_repr}(\mathcal{C}_j)\bigr)$. We shall abuse notation by writing $\mathcal{R} = \mathrm{con\_repr}(\mathcal{P})$ as a shorthand for $\mathcal{R} = \mathrm{con}\bigl(\mathrm{con\_repr}(\mathcal{C})\bigr)$, provided the constraint system $\mathcal{C}$ describing $\mathcal{P}$ is clear from context.

4.1 The Generators of NNC Polyhedra

One of the fundamental features of the DD method, and the very reason for its name, is the ability to represent a closed polyhedron using a system of constraints or a system of generators. While being equivalent, there are contexts where each of these descriptions is the most appropriate, so that a good library should provide the client application with both possibilities.

Any NNC polyhedron can be easily described by using constraint systems containing strict inequalities, but a similar generalization of the concept of generator system seems to be missing. This causes an asymmetry in the handling of NNC polyhedra using the DD method that is reflected in existing software libraries. For instance, the following sentence comes from the documentation of New Polka [24, Section 1.1.4, page 10] (where s denotes the ϵ coefficient):

> Don't ask me the intuitive meaning of $s \neq 0$ in rays and vertices !

The problem is discussed in some more detail in [19, Section 4.5, pp. 10–11]:

> While strict inequations handling is transparent for constraints (being displayed accurately), the extra dimension added to the variables space is apparent when it comes to generators : one extra coefficient, resp. extra vertices (as `epsilon` is bounded), materialize this dimension in every generator, resp. generators system.
> This makes more difficult to define polyhedra with the only help of generators : one should carefully study the extra vertices with non null `epsilon` coefficients added to constraints defined polyhedra, in the case of large inequations, and the case of strict inequations.

A clear understanding of the generator systems of NNC polyhedra is helpful, for instance, in order to generalize to the domain $\mathbb{P}_n$ those operators on closed polyhedra that are defined or implemented in terms of the generator system representation (such as the *time-elapse* operator of [20, 21] or the generators-based widening of [6]).

We will now show how, by decoupling the user interface from the details of the particular implementation, it is possible to provide an intuitive generalization of the concept of generator system, so that the geometric features of any NNC polyhedron can be accurately represented. The key step is the introduction of a new kind of generator.

Definition 4. (Closure point.) *A vector* $\boldsymbol{c} \in \mathbb{R}^n$ *is a* closure point *of* $S \subseteq \mathbb{R}^n$ *if and only if* $\boldsymbol{c} \in \mathbb{C}(S)$.

When considering NNC polyhedra, closure points can be characterized by a property which is similar to the one used when defining rays.

Proposition 1. *A vector* $\boldsymbol{c} \in \mathbb{R}^n$ *is a closure point of the NNC polyhedron* $\mathcal{P} \in \mathbb{P}_n$ *if and only if* $\mathcal{P} \neq \varnothing$ *and* $\lambda\boldsymbol{p} + (1-\lambda)\boldsymbol{c} \in \mathcal{P}$ *for every point* $\boldsymbol{p} \in \mathcal{P}$ *and* $\lambda \in \mathbb{R}$ *such that* $0 < \lambda < 1$.

We are now able to provide a parametric description for any NNC polyhedron.

Theorem 2. *The set* $\mathcal{P} \subseteq \mathbb{R}^n$ *is an NNC polyhedron if and only if there exist finite sets* $R, P, C \subseteq \mathbb{R}^n$ *of cardinality* k, ℓ *and* m, *respectively, such that* $\mathbf{0} \notin R$ *and, for any matrices* $K \in \mathbb{R}^{n\times k}$, $L \in \mathbb{R}^{n\times \ell}$, $M \in \mathbb{R}^{n\times m}$ *where* $K \in \operatorname{matrix}(R)$, $L \in \operatorname{matrix}(P)$ *and* $M \in \operatorname{matrix}(C)$,

$$\mathcal{P} = \left\{ K\boldsymbol{\mu} + L\boldsymbol{\nu} + M\boldsymbol{\eta} \in \mathbb{R}^n \;\middle|\; \begin{array}{l} \boldsymbol{\mu} \in \mathbb{R}^k_+, \boldsymbol{\nu} \in \mathbb{R}^\ell_+, \boldsymbol{\nu} \neq \mathbf{0}, \boldsymbol{\eta} \in \mathbb{R}^m_+, \\ \sum_{i=1}^{\ell} \nu_i + \sum_{i=1}^{m} \eta_i = 1 \end{array} \right\}.$$

When $\mathcal{P} \neq \varnothing$, we say that $\mathcal{P}$ is described by the *extended generator system* $\mathcal{G} = (R, P, C)$. As was the case for closed polyhedra, the vectors in R and P are rays and points of $\mathcal{P}$, respectively. The condition $\boldsymbol{\nu} \neq \mathbf{0}$ ensures that at least one of the points of P plays an active role in any convex combination of the vectors of P and C. It follows from Proposition 1 that the vectors of C are closure points of $\mathcal{P}$. Since rays and closure points need a supporting point, we have $\mathcal{P} = \varnothing$ if and only if $P = \varnothing$.

In Figure 1, we provide a few examples of the use of extended generator systems for the description of NNC polyhedra: (closure) points are represented by small (un-) filled circles, whereas rays are represented by vectors that, for notational convenience, are applied to points. The NNC polyhedron $\mathcal{P}_1$ is an open rectangle and is described by the closure points A, B, C, D and the point E; note that E could have been replaced by any other point of $\mathcal{P}_1$, whereas all the four closure points have to be included in any generator system for $\mathcal{P}_1$. The NNC polyhedron $\mathcal{P}_2$ is another rectangle that is neither closed nor open: since A' is a point, the open segments $]A', B'[$ and $]A', D'[$ are included in $\mathcal{P}_2$; similarly, the open segment $]B', C'[$ is included in $\mathcal{P}_2$ because E' is a point of the

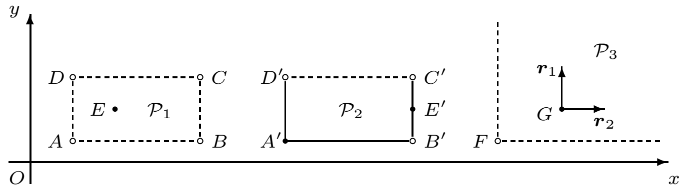

Fig. 1. Using closure points to define NNC polyhedra.

generator system (note that E' is needed, since both B' and C' are not in $\mathcal{P}_2$, but it could have been replaced by any other point lying on this open segment); in contrast, the closed segment $[C', D']$ is disjoint from $\mathcal{P}_2$, because neither C' nor D' are points of $\mathcal{P}_2$. Finally, the NNC polyhedron $\mathcal{P}_3$ can be regarded as the translation by F of the open positive quadrant. Thus the generator system includes the closure point F, the rays $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ and the point G; again, the latter could have been replaced by any other point of $\mathcal{P}_3$.

We will now show how the high level description of an NNC polyhedron provided by an extended generator system can be mapped into an implementation based on the ϵ dimension approach. Namely, if $\mathcal{G} = (R, P, C)$ is the extended generator system describing $\mathcal{P} \in \mathbb{P}_n$, the corresponding closed representation $\mathcal{R} \in \mathbb{CP}_{n+1}$ is described by the generator system $\mathrm{gen\_repr}(\mathcal{G}) \stackrel{\mathrm{def}}{=} (R', P')$ where

$$\begin{aligned} R' &= \bigl\{ (\boldsymbol{r}^{\mathrm{T}}, 0)^{\mathrm{T}} \bigm| \boldsymbol{r} \in R \bigr\}, \\ P' &= \bigl\{ (\boldsymbol{p}^{\mathrm{T}}, 1)^{\mathrm{T}} \bigm| \boldsymbol{p} \in P \bigr\} \cup \bigl\{ (\boldsymbol{x}^{\mathrm{T}}, 0)^{\mathrm{T}} \bigm| \boldsymbol{x} \in P \cup C \bigr\}. \end{aligned}$$

Even in this case, the value 1 for the coordinate of the ϵ dimension in the translation of points is almost arbitrary: any other positive value could be chosen. Different though equivalent extended generator systems $\mathcal{G}_j$ describing $\mathcal{P} \in \mathbb{P}_n$ may result in different representation polyhedra $\mathcal{R}_j = \mathrm{gen}\bigl(\mathrm{gen\_repr}(\mathcal{G}_j)\bigr)$. It is worth noting that the closure points of $\mathcal{P}$ are mapped to points of $\mathcal{R}$ having a zero ϵ coordinate. In contrast, the points of $\mathcal{P}$ are mapped to a pair of points of $\mathcal{R}$, having a zero and a strictly positive ϵ coordinate, respectively; by doing this, we explicitly enforce in the closed representation the key invariant saying that any point of $\mathcal{P}$ is also a closure point of $\mathcal{P}$.

Definition 5. (ϵ-representation.) *A polyhedron $\mathcal{R} \in \mathbb{CP}_{n+1}$ is said to be an* ϵ-representation *if and only if*

$$\exists \delta > 0 \,.\, \mathcal{R} \subseteq \mathrm{con}\bigl(\{0 \le \epsilon \le \delta\}\bigr); \tag{1}$$

$$\bigl(\exists \epsilon > 0 \,.\, (\boldsymbol{x}^{\mathrm{T}}, \epsilon)^{\mathrm{T}} \in \mathcal{R}\bigr) \implies (\boldsymbol{x}^{\mathrm{T}}, 0)^{\mathrm{T}} \in \mathcal{R}. \tag{2}$$

$\mathcal{R}$ is said to be an ϵ-representation for *$\mathcal{P} \in \mathbb{P}_n$, denoted $\mathcal{R} \Rrightarrow_\epsilon \mathcal{P}$, if $\mathcal{R}$ is an ϵ-representation and*

$$\mathcal{P} = [\![\mathcal{R}]\!] \stackrel{\mathrm{def}}{=} \bigl\{ \boldsymbol{x} \in \mathbb{R}^n \bigm| \exists \epsilon > 0 \,.\, (\boldsymbol{x}^{\mathrm{T}}, \epsilon)^{\mathrm{T}} \in \mathcal{R} \bigr\}. \tag{3}$$

Proposition 2. *Let* $(\mathcal{C}, \mathcal{G}) \equiv \mathcal{P} \in \mathbb{P}_n$*. Then we have* $\operatorname{con}\bigl(\operatorname{con\_repr}(\mathcal{C})\bigr) \Rrightarrow_\epsilon \mathcal{P}$ *and* $\operatorname{gen}\bigl(\operatorname{gen\_repr}(\mathcal{G})\bigr) \Rrightarrow_\epsilon \mathcal{P}$*.*

Operations such as the intersection of NNC polyhedra and the application of affine transformations can be safely performed on any of the ϵ-representations of the arguments; the same holds for the poly-hull operation, provided none of the arguments is an empty NNC polyhedron. Some care has to be taken when testing the emptiness of an NNC polyhedron or the inclusion of an NNC polyhedron into another one [20]. For instance, any ϵ-representation included in the hyperplane $\epsilon = 0$ actually encodes the empty NNC polyhedron.

Proposition 3. *Let* $\mathcal{R} \Rrightarrow_\epsilon \mathcal{P}$*,* $\mathcal{R}_1 \Rrightarrow_\epsilon \mathcal{P}_1$ *and* $\mathcal{R}_2 \Rrightarrow_\epsilon \mathcal{P}_2$*. Then*

1. $\mathcal{P} = \varnothing$ *if and only if* $\mathcal{R} \subseteq \operatorname{con}\bigl(\{\epsilon \leq 0\}\bigr)$*;*
2. $\mathcal{R}_1 \cap \mathcal{R}_2 \Rrightarrow_\epsilon \mathcal{P}_1 \cap \mathcal{P}_2$*;*
3. $(\mathcal{P}_1 \neq \varnothing \wedge \mathcal{P}_2 \neq \varnothing) \implies (\mathcal{R}_1 \uplus \mathcal{R}_2 \Rrightarrow_\epsilon \mathcal{P}_1 \uplus \mathcal{P}_2)$*;*
4. *let* $f \stackrel{\text{def}}{=} \lambda \boldsymbol{x} \in \mathbb{R}^n \,.\, A\boldsymbol{x} + \boldsymbol{b}$ *be any affine transformation defined on* $\mathbb{P}_n$*; then* $g(\mathcal{R}) \Rrightarrow_\epsilon f(\mathcal{P})$*, where*

$$g \stackrel{\text{def}}{=} \lambda \begin{pmatrix} \boldsymbol{x} \\ \epsilon \end{pmatrix} \in \mathbb{R}^{n+1} \,.\, \begin{pmatrix} A & \mathbf{0} \\ \mathbf{0}^{\mathrm{T}} & 1 \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \epsilon \end{pmatrix} + \begin{pmatrix} \boldsymbol{b} \\ 0 \end{pmatrix}$$

is the corresponding affine transformation on $\mathbb{CP}_{n+1}$*.*

4.2 The Issue of Minimization

When adopting the ϵ dimension approach proposed in [20], the computed representation $\mathcal{R} \in \mathbb{CP}_{n+1}$ of an NNC polyhedron $\mathcal{P} \in \mathbb{P}_n$ will depend not only on the particular constraint system considered, but also on the sequence of operations (intersections, poly-hulls, affine transformations, etc.) performed on the polyhedron. If not properly handled, such an abundance of possible representations may cause problems when trying to provide a non-redundant description of $\mathcal{P}$.

The reason is that libraries such as New Polka compute the minimal forms of the closed representation $\mathcal{R}$. Very often, such an approach results in a redundant description of the represented NNC polyhedron: there may be a different ϵ-representation for $\mathcal{P}$ that is characterized by a smaller number of constraints (generators). The following example illustrates this point.

Consider the two NNC polyhedra $\mathcal{P}_1, \mathcal{P}_2 \in \mathbb{P}_1$ defined as

$$\mathcal{P}_1 \stackrel{\text{def}}{=} \operatorname{con}\bigl(\{0 < x < 2\}\bigr), \qquad \mathcal{P}_2 \stackrel{\text{def}}{=} \operatorname{con}\bigl(\{2 < x < 3\}\bigr).$$

These polyhedra are encoded by the closed polyhedra $\mathcal{R}_1, \mathcal{R}_2 \in \mathbb{CP}_2$ such that[5]

$$\mathcal{R}_1 \stackrel{\text{def}}{=} \operatorname{con\_repr}(\mathcal{P}_1) = \bigl\{\, (x, \epsilon)^{\mathrm{T}} \in \mathbb{R}^2 \bigm| \epsilon \geq 0,\ x - \epsilon \geq 0,\ -x - \epsilon \geq -2 \,\bigr\},$$
$$\mathcal{R}_2 \stackrel{\text{def}}{=} \operatorname{con\_repr}(\mathcal{P}_2) = \bigl\{\, (x, \epsilon)^{\mathrm{T}} \in \mathbb{R}^2 \bigm| \epsilon \geq 0,\ x - \epsilon \geq 2,\ -x - \epsilon \geq -3 \bigr\}.$$

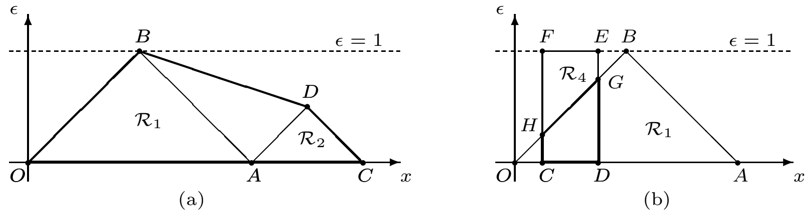

Fig. 2. (a) The poly-hull of $\mathcal{R}_1$ and $\mathcal{R}_2$; (b) The intersection of $\mathcal{R}_1$ and $\mathcal{R}_4$.

Suppose now that the user wants to compute the poly-hull of the two original polyhedra, therefore obtaining the NNC polyhedron $\mathcal{P}_3 \stackrel{\text{def}}{=} \operatorname{con}\bigl(\{0 < x < 3\}\bigr)$.

At the representation level, the situation will be the one described in Figure 2(a): $\mathcal{P}_3$ is represented by the closed polyhedron generated by the four vertices O, C, D, and B, whereas point A is identified as redundant. Formally,

$$\mathcal{R}_3 \stackrel{\text{def}}{=} \bigl\{\, (x, \epsilon)^{\mathrm{T}} \in \mathbb{R}^2 \bigm| \epsilon \geq 0,\ x - \epsilon \geq 0,\ -x - \epsilon \geq -3,\ -x - 3\epsilon \geq -4 \,\bigr\}.$$

The last non-strict inequality, which corresponds to the segment $[B, D]$ in Figure 2(a), is not redundant as far as the ϵ-representation $\mathcal{R}_3$ is concerned. However, this non-strict inequality stands for the strict inequality $x < 4$, which is clearly redundant when considering the represented polyhedron $\mathcal{P}_3$.

The problem outlined above is even more critical when dealing with higher dimension vector spaces: it is straightforward to devise examples where more than half of the constraints in the "minimized" representation happen to be redundant. Even when disregarding these pathological cases, redundancy could have a serious negative impact on the efficiency of some of the operations computed on the polyhedron, e.g., those characterized by a worst case complexity that is exponential in the size of the description.

Besides efficiency issues, the presence of redundant constraints may also cause headaches to the users of the library. For instance, suppose one wants to know if a given NNC polyhedron is not topologically closed. Ordinary users (i.e., all the users but the experts) may be tempted to implement such a test by checking whether the constraint system in minimal form contains any strict inequality. Unfortunately, such an approach would be unsound, as can be easily observed by considering the scenario proposed in Figure 2(b). Here, the NNC polyhedron $\mathcal{P}_1$ is intersected with the NNC polyhedron $\mathcal{P}_4 \stackrel{\text{def}}{=} \operatorname{con}\bigl(\{1 \leq 4x \leq 3\}\bigr)$, whose representation $\mathcal{R}_4 = \operatorname{con\_repr}(\mathcal{P}_4)$ is the rectangle having vertices C, D, E and F. The resulting trapezium is another ϵ-representation for the NNC polyhedron $\mathcal{P}_4$, which is topologically closed. However, any constraint system describing the trapezium will also encode the strict inequality $x > 0$, corresponding to the closed segment $[G, H]$.

[5] In both cases, we do not explicitly include the ϵ-upper-bound constraint $\epsilon \leq 1$, which happens to be redundant.

It is therefore meaningful to address the problem of providing a minimization procedure that is able to remove all of these redundancies. To this end, the introduction of some notation will be helpful.

We say that a vector $\boldsymbol{v}$ *saturates* the constraint $\langle \boldsymbol{a}, \boldsymbol{x} \rangle \bowtie b$ if and only if $\langle \boldsymbol{a}, \boldsymbol{v} \rangle = b$. For any constraint system $\mathcal{C}$ and generator system $\mathcal{G} = (R, P)$, we define

$$\begin{aligned}
\mathrm{sat_con}(\boldsymbol{v}, \mathcal{C}) &\stackrel{\mathrm{def}}{=} \{\, c \in \mathcal{C} \mid \boldsymbol{v} \text{ saturates } c \,\}; \\
\mathrm{sat_gen}(c, \mathcal{G}) &\stackrel{\mathrm{def}}{=} \{\, \boldsymbol{v} \in R \cup P \mid \boldsymbol{v} \text{ saturates } c \,\}.
\end{aligned}$$

Let $(\mathcal{C}, \mathcal{G}) \equiv \mathcal{R} \in \mathbb{CP}_{n+1}$ be such that $\mathcal{R} \Rrightarrow_\epsilon \mathcal{P}$. The set of *strict* and *non-strict inequality encodings* $\mathcal{C}_>$ and $\mathcal{C}_\geq$ of constraint system $\mathcal{C}$ are defined as

$$\begin{aligned}
\mathcal{C}_> &\stackrel{\mathrm{def}}{=} \Bigl\{\, \bigl(\langle \boldsymbol{a}, \boldsymbol{x} \rangle + s \cdot \epsilon \geq b\bigr) \in \mathcal{C} \Bigm| \boldsymbol{a} \neq \boldsymbol{0}, s < 0 \,\Bigr\}; \\
\mathcal{C}_\geq &\stackrel{\mathrm{def}}{=} \Bigl\{\, \bigl(\langle \boldsymbol{a}, \boldsymbol{x} \rangle + s \cdot \epsilon \geq b\bigr) \in \mathcal{C} \Bigm| \boldsymbol{a} \neq \boldsymbol{0}, s = 0 \,\Bigr\};
\end{aligned}$$

we also define the set of *ϵ-upper-bounds* as

$$\mathcal{C}_\epsilon \stackrel{\mathrm{def}}{=} \Bigl\{\, \bigl(\langle \boldsymbol{a}, \boldsymbol{x} \rangle + s \cdot \epsilon \geq b\bigr) \in \mathcal{C} \Bigm| \boldsymbol{a} = \boldsymbol{0}, s < 0 \,\Bigr\}.$$

For ease of notation, a constraint $c \in \mathcal{C}_\epsilon$ will be usually denoted as $\epsilon \leq \delta$, where $\delta \stackrel{\mathrm{def}}{=} b/s$. Note that, by Proposition 3, we have $\delta > 0$ whenever $\mathcal{P} \neq \varnothing$. Similarly, the set of *closure point encodings* $\mathcal{G}_C$, the set of *point encodings* $\mathcal{G}_P$, and the set of *unmatched point encodings* $\mathcal{G}_U \subseteq \mathcal{G}_P$ of the generator system $\mathcal{G} = (R, P)$ are defined as follows:

$$\begin{aligned}
\mathcal{G}_C &\stackrel{\mathrm{def}}{=} \bigl\{\, (\boldsymbol{p}^\mathrm{T}, e)^\mathrm{T} \in P \bigm| e = 0 \,\bigr\}; \\
\mathcal{G}_P &\stackrel{\mathrm{def}}{=} \bigl\{\, (\boldsymbol{p}^\mathrm{T}, e)^\mathrm{T} \in P \bigm| e > 0 \,\bigr\}; \\
\mathcal{G}_U &\stackrel{\mathrm{def}}{=} \bigl\{\, (\boldsymbol{p}^\mathrm{T}, e)^\mathrm{T} \in P \bigm| e > 0, (\boldsymbol{p}^\mathrm{T}, 0)^\mathrm{T} \notin P \,\bigr\}.
\end{aligned}$$

Definition 6. (Strong minimal form.) *Let $\mathcal{R} \Rrightarrow_\epsilon \mathcal{P}$, where $(\mathcal{C}, \mathcal{G}) \equiv \mathcal{R}$ is a DD pair in minimal form. Then, we say that*

- *$\mathcal{C}$ is in* strong minimal form *if there does not exist a constraint system $\mathcal{C}'$ in minimal form such that $(\mathcal{C}'_> \cup \mathcal{C}'_\geq) \subset (\mathcal{C}_> \cup \mathcal{C}_\geq)$ and $\mathrm{con}(\mathcal{C}') \Rrightarrow_\epsilon \mathcal{P}$;*
- *$\mathcal{G} = (R, P)$ is in* strong minimal form *if there does not exist a generator system $\mathcal{G}' = (R', P')$ such that $\mathcal{G}' \neq \mathcal{G}$, $R' \subseteq R$, $P' \subseteq P$ and $\mathrm{gen}(\mathcal{G}') \Rrightarrow_\epsilon \mathcal{P}$.*

For the computation of strong minimal forms (smf's, for short), the key step is the identification of ϵ-redundant constraints and generators.

Definition 7. (ϵ-redundancy.) *Let $(\mathcal{C}, \mathcal{G}) \equiv \mathcal{R} \in \mathbb{CP}_{n+1}$. A constraint c is* ϵ-redundant *in $\mathcal{C}$ if $c \in \mathcal{C}_>$ and at least one of the following conditions holds:*

$$\begin{gathered}
\mathrm{sat_gen}(c, \mathcal{G}) \cap \mathcal{G}_C = \varnothing; \\
\exists c' \in \mathcal{C}_> \setminus \{c\} \,.\, \mathrm{sat_gen}(c, \mathcal{G}) \setminus \mathcal{G}_P \subseteq \mathrm{sat_gen}(c', \mathcal{G}).
\end{gathered}$$

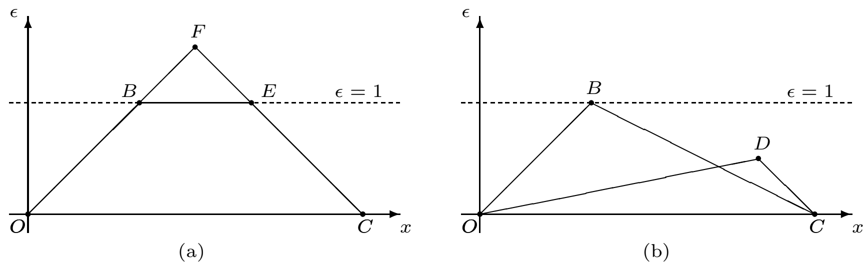

Fig. 3. Four different ϵ-representations for $\mathcal{P}_3$, obtained by applying the strong minimization process to (a) the constraint system; or (b) the generator system.

A generator $\boldsymbol{p}$ *is* ϵ-redundant *in* $\mathcal{G}$ *if* $\boldsymbol{p} \in \mathcal{G}_U$ *and*

$$\exists \boldsymbol{p}' \in \mathcal{G}_P \setminus \{\boldsymbol{p}\} \,.\, \mathrm{sat\_con}(\boldsymbol{p}, \mathcal{C}) \cap \mathcal{C}_{\geq} \subseteq \mathrm{sat\_con}(\boldsymbol{p}', \mathcal{C}).$$

Proposition 4. *Let* $\mathcal{R} \Rrightarrow_\epsilon \mathcal{P} \neq \varnothing$, *where* $(\mathcal{C}, \mathcal{G}) \equiv \mathcal{R}$ *is a DD pair in minimal form. Then, the following hold:*

1. *If* c *is* ϵ*-redundant in* $\mathcal{C}$, *then* $\mathrm{con}\bigl(\mathcal{C} \setminus \{c\} \cup \{\epsilon \leq 1\}\bigr) \Rrightarrow_\epsilon \mathcal{P}$;
2. *If* $\boldsymbol{p}$ *is* ϵ*-redundant in* $\mathcal{G} = (R, P)$, *then* $\mathrm{gen}\Bigl(\bigl(R, P \setminus \{\boldsymbol{p}\}\bigr)\Bigr) \Rrightarrow_\epsilon \mathcal{P}$;
3. *If* $\mathcal{C}$ *contains no* ϵ*-redundant constraint, then it is in* smf;
4. *If* $\mathcal{G}$ *contains no* ϵ*-redundant generator, then it is in* smf.

It is worth stressing that, even if $(\mathcal{C}, \mathcal{G})$ is a DD pair for $\mathcal{R} \Rrightarrow_\epsilon \mathcal{P}$, after removing all ϵ-redundant constraints and generators the resulting descriptions $\mathcal{C}'$ and $\mathcal{G}'$ may be such that $\mathrm{con}(\mathcal{C}') \neq \mathrm{gen}(\mathcal{G}')$, so that $(\mathcal{C}', \mathcal{G}')$ is not a DD pair. Moreover, having one of the systems in smf does not imply that the corresponding dual description will also be in smf.

As an example, we now compute smf's for the polyhedron $\mathcal{R}_3$ represented in Figure 2(a). Let us first consider the constraint system. The two strict inequality encodings $x - \epsilon \geq 0$ and $-x - \epsilon \geq -3$, which correspond to segments $[O, B]$ and $[C, D]$, are not ϵ-redundant, because they are saturated by the closure point encodings O and C, respectively. In contrast, the constraint $x - 3\epsilon \geq 4$, corresponding to segment $[B, D]$, is identified as ϵ-redundant (no closure point encoding saturates it) and can be removed provided we add the constraint $\epsilon \leq 1$. The resulting constraint system, which is in smf, defines the trapezium of vertices O, C, E, and B represented in Figure 3(a). Note that the generator system for this trapezium is not in smf; this additional property can be enforced by removing the ϵ-upper-bound constraint $\epsilon \leq 1$, therefore obtaining the triangle of vertices O, C, and F.

Starting again from polyhedron $\mathcal{R}_3$, let us now consider the strong minimization of its generator system, which is made up of the four points O, C, D, and B. Each one of the two unmatched point encodings B and D is made ϵ-redundant by the other one (they both saturate the empty set of non-strict inequality encodings); as a consequence, one of them can be removed, obtaining either one of the triangles OCB and OCD represented in Figure 3(b), which are both in smf.

5 The Parma Polyhedra Library

The Parma Polyhedra Library (PPL, `http://www.cs.unipr.it/ppl/`) is a collaborative project started in January 2001 at the Department of Mathematics of the University of Parma. It aims at becoming a truly professional library for the handling of approximations based on (not necessarily closed) convex polyhedra targeted at abstract interpretation and computer-aided verification. In this section we briefly review some of the key features of the library. Before continuing, it is perhaps worth stressing that the theoretical treatment of previous sections applies to polyhedra in $\mathbb{R}^n$. Not surprisingly, the implementation only deals with rational polyhedra, that is, polyhedra that can be defined by constraints with rational coefficients or, dually, generators with rational coordinates. It is easy to prove that the property of being rational is preserved by all the operations of interest for the applications the library aims at.

The library is written in standard C++ and this ensures, among other things, maximum portability across different computing platforms. Using C++ for the development made it easier to adopt a number of programming techniques that are the key to the library's robustness, generality, efficiency and usability. However, for the sake of maximal code reuse and utility, care has been taken not to require the client application to be written in C++. The library includes a complete C interface and thus can be interfaced to all programming languages' implementations (there are many) that provide a C interface. There is also a Prolog interface supporting several Prolog systems.

One of the key features of the library is robustness. In particular, this means that failure is avoided whenever possible and, in all other cases, failure is recognized and handled properly. The PPL uses arbitrary precision integer arithmetic to implement coefficients and coordinates, and is thus immune from both rounding and overflow problems. In addition, all the data structures used in the implementation are fully dynamic and can expand automatically (in amortized constant time) to any dimension allowed by the available virtual memory. Two other aspects of failure avoidance are hiding and systematic checking of the interface invariants. In contrast to other libraries, the PPL hides the implementation details almost completely. For instance, the internal representation of constraints, generators and systems thereof need not concern the client application. Similarly, implementation devices such as the *positivity constraint* [32] and all the matters regarding the ϵ-representation encoding of NNC polyhedra are completely invisible from outside. The client application is provided with more natural interfaces, allowing the direct manipulation of higher level concepts, such as inequalities, lines and closure points. For instance, in the appropriate contexts, '`X < 5*Z`' and '`X + 2*Y + 5*Z >= 7`' is valid syntax expressing a strict and a non-strict inequality, both in the C++ and the Prolog interfaces. Even the C interface, which is at a considerably lower level of abstraction, refers to concepts like linear expression, constraint and constraint system and not to their possible implementations such as vectors and matrices. As usual, this is important for error prevention and to allow maximum latitude for the implementation.

```
typedef Parma_Polyhedra_Library::NNC_Polyhedron PH;
void complex_function(const PH& ph1, const PH& ph2, PH& result) {
  try {
    start_timer(max_time_for_complex_function);
    complex_function_on_polyhedra(ph1, ph2, result ...);
    stop_timer();
  }
  catch (Exception& e) { // Memory exhausted or timeout or other error.
    ...
    BoundingBox bb1, bb2, bb_result;
    ph1.shrink_bounding_box(bb1);  ph2.shrink_bounding_box(bb2);
    complex_function_on_bounding_boxes(bb1, bb2, bb_result ...);
    result = Polyhedron(bb_result);
  }
}
```

Fig. 4. Falling back to bounding boxes when the analysis with polyhedra *is* too costly.

Forbidding access to the internal structures manipulated by the library and systematic checking of the interface invariants could impede the overall system performance. However, the resulting overhead can be completely repaid (to the point of giving rise to a speedup) if the implementation exploits the freedom it has from the user interface.

In the design of the library, particular care has been taken concerning issues of scalability since any user of any polyhedra library must face the possibility of excessive CPU time consumption and/or memory usage [23]. A far from satisfactory but possible solution is to hope for the best but eventually "kill" those processes requiring more than the available resources [7]. In contrast, what we aim at is support for the dynamic and automatic composition of the trade-off between expressivity and efficiency. In this scenario, the static analysis or verification procedures should, by default, use the more descriptive (and costly) domains, such as the convex polyhedra. When the computation of an operation on these descriptions requires too much time, or memory space, the system should detect this and perform a change of representation, for instance by approximating the input polyhedra with enclosing *bounding boxes* (rectangular regions with sides parallel to the axes); after the execution of the requested operation on this less precise (but much less computationally expensive) domain, the result is converted back into a convex polyhedron. Such a scenario is feasible only if the polyhedra library is implemented so that it can consistently react, without any loss of system resources, to events such as a timeout or an exception thrown by the memory allocation routines. As far as we know, the Parma Polyhedra Library is the first one to provide this facility. The idea is exemplified in Figure 4, where a robust C++ implementation of `complex_function` is sketched. The objective is to compute the polyhedron `result` starting from the polyhedra `ph1` and `ph2`. The operation is first tried on the polyhedra themselves and, in case of failure, the computation is done on the bounding box approximations of `ph1` and `ph2`. In the latter case the result of the bounding box computation, `bb_result` is converted into a convex polyhedron object and assigned to the output polyhedron `result`. The trade-off between efficiency and precision can also be controlled by means of the widening operations provided by the PPL, which are based on those originally proposed in [15] and improved in [14].

Concerning the efficiency of the PPL, at present no application can use both the PPL and another convex polyhedra library, so comparisons are not possible. However, the PPL has been integrated with the China analyzer [1] for the purpose of detecting linear argument size relations [5]. The performance of the combined system has been compared, on the same task, with the performance of the cTI analyzer [28], which uses an implementation of convex polyhedra based on the SICStus CLP(Q) package. The combined system China+PPL outperformed that version of cTI in a significant way, exhibiting termination instead of thrashing and speedups of one or more orders of magnitude.

The Parma Polyhedra Library is free software released under the GPL: code and documentation can be downloaded and its development can be followed at `http://www.cs.unipr.it/ppl/`.

6 Conclusion

Convex polyhedra provide the basis for several abstractions used in static analysis and computer-aided verification of complex and sometimes mission critical systems. For such purposes an implementation of convex polyhedra must be firmly based on a clear theoretical framework and written in accordance with sound software engineering principles. In this paper we have presented some of the most important ideas that are behind the Parma Polyhedra Library. In particular, we have provided a novel theoretical framework for the representation and manipulation of not necessarily closed convex polyhedra. We have also briefly sketched some important features of the design and implementation of the library. We believe that all this work constitutes a significant improvement in the state of the art and an encouragement to the wider adoption of abstract interpretation techniques in static analysis and computer-aided verification.

Acknowledgments. We would like to thank Costantino Medori for the many useful discussions we had with him on the subject of this paper, Fred Mesnard for letting us play with cTI's analyzer, and the anonymous reviewers for their helpful comments and observations.

References

1. R. Bagnara. *Data-Flow Analysis for Constraint Logic-Based Languages.* PhD thesis, Dipartimento di Informatica, Università di Pisa, Pisa, Italy, 1997. Printed as Report TD-1/97.
2. R. Bagnara, P. M. Hill, E. Ricci, and E. Zaffanella. *The Parma Polyhedra Library User's Manual.* Department of Mathematics, University of Parma, Parma, Italy, release 0.4 edition, July 2002. Available at `http://www.cs.unipr.it/ppl/`.

3. R. Bagnara, E. Ricci, E. Zaffanella, and P. M. Hill. Possibly not closed convex polyhedra and the Parma Polyhedra Library. Quaderno 286, Dipartimento di Matematica, Università di Parma, Italy, 2002. See also [4]. Available at `http://www.cs.unipr.it/Publications/`.
4. R. Bagnara, E. Ricci, E. Zaffanella, and P. M. Hill. Errata for technical report "Quaderno 286". Available at `http://www.cs.unipr.it/Publications/`, 2002.
5. F. Benoy and A. King. Inferring argument size relationships with CLP(R). In J. P. Gallagher, editor, *Logic Programming Synthesis and Transformation: Proceedings of the 6th International Workshop*, volume 1207 of *Lecture Notes in Computer Science*, pages 204–223, Stockholm, Sweden, 1997. Springer-Verlag, Berlin.
6. F. Besson, T. P. Jensen, and J.-P. Talpin. Polyhedral analysis for synchronous languages. In A. Cortesi and G. Filé, editors, *Static Analysis: Proceedings of the 6th International Symposium*, volume 1694 of *Lecture Notes in Computer Science*, pages 51–68, Venice, Italy, 1999. Springer-Verlag, Berlin.
7. N. S. Bjørner, A. Browne, M. Colón, B. Finkbeiner, Z. Manna, M. Pichora, H. B. Sipma, and T. E. Uribe. *STeP: The Stanford Temporal Prover (Educational Release) User's Manual.* Computer Science Department, Stanford University, Stanford, California, version 1.4-α edition, July 1998. Available at `http://www-step.stanford.edu/`.
8. N. V. Chernikova. Algorithm for finding a general formula for the non-negative solutions of system of linear equations. *U.S.S.R. Computational Mathematics and Mathematical Physics*, 4(4):151–158, 1964.
9. N. V. Chernikova. Algorithm for finding a general formula for the non-negative solutions of system of linear inequalities. *U.S.S.R. Computational Mathematics and Mathematical Physics*, 5(2):228–233, 1965.
10. N. V. Chernikova. Algorithm for discovering the set of all solutions of a linear programming problem. *U.S.S.R. Computational Mathematics and Mathematical Physics*, 8(6):282–293, 1968.
11. M. A. Colón and H. B. Sipma. Synthesis of linear ranking functions. In T. Margaria and W. Yi, editors, *Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2001)*, volume 2031 of *Lecture Notes in Computer Science*, pages 67–81, Genova, Italy, 2001. Springer-Verlag, Berlin.
12. P. Cousot, editor. *Static Analysis: 8th International Symposium, SAS 2001*, volume 2126 of *Lecture Notes in Computer Science*, Paris, 2001. Springer-Verlag, Berlin.
13. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the Fourth Annual ACM Symposium on Principles of Programming Languages*, pages 238–252, 1977.
14. P. Cousot and R. Cousot. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation. In M. Bruynooghe and M. Wirsing, editors, *Proceedings of the 4th International Symposium on Programming Language Implementation and Logic Programming*, volume 631 of *Lecture Notes in Computer Science*, pages 269–295, Leuven, Belgium, 1992. Springer-Verlag, Berlin.
15. P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages*, pages 84–96, Tucson, Arizona, 1978. ACM Press.
16. N. Dor, M. Rodeh, and S. Sagiv. Cleanness checking of string manipulations in C programs via integer analysis. In Cousot [12], pages 194–212.

17. K. Fukuda and A. Prodon. Double description method revisited. In M. Deza, R. Euler, and Y. Manoussakis, editors, *Combinatorics and Computer Science, 8th Franco-Japanese and 4th Franco-Chinese Conference, Brest, France, July 3-5, 1995, Selected Papers*, volume 1120 of *Lecture Notes in Computer Science*, pages 91–111. Springer-Verlag, Berlin, 1996.
18. N. Halbwachs. Delay analysis in synchronous programs. In C. Courcoubetis, editor, *Computer Aided Verification: Proceedings of the 5th International Conference*, volume 697 of *Lecture Notes in Computer Science*, pages 333–346, Elounda, Greece, 1993. Springer-Verlag, Berlin.
19. N. Halbwachs, A. Kerbrat, and Y.-E. Proy. *POLyhedra INtegrated Environment*. Verimag, France, version 1.0 of POLINE edition, September 1995. Documentation taken from source code.
20. N. Halbwachs, Y.-E. Proy, and P. Raymond. Verification of linear hybrid systems by means of convex approximations. In B. Le Charlier, editor, *Static Analysis: Proceedings of the 1st International Symposium*, volume 864 of *Lecture Notes in Computer Science*, pages 223–237, Namur, Belgium, 1994. Springer-Verlag, Berlin.
21. N. Halbwachs, Y.-E. Proy, and P. Roumanoff. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design*, 11(2):157–185, 1997.
22. T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1(1+2):110–122, 1997.
23. T. A. Henzinger, J. Preussig, and H. Wong-Toi. Some lessons from the hytech experience. In *Proceedings of the 40th Annual Conference on Decision and Control*, pages 2887–2892. IEEE Computer Society Press, 2001.
24. B. Jeannet. *Convex Polyhedra Library*, release 1.1.3c edition, March 2002. Documentation of the “New Polka” library available at `http://www.irisa.fr/prive/Bertrand.Jeannet/newpolka.html`.
25. H. Le Verge. A note on Chernikova's algorithm. *Publication interne* 635, IRISA, Campus de Beaulieu, Rennes, France, 1992.
26. V. Loechner. *PolyLib*: A library for manipulating parameterized polyhedra. Available at `http://icps.u-strasbg.fr/~loechner/polylib/`, March 1999. Declares itself to be a continuation of [32].
27. Z. Manna, N. S. Bjørner, A. Browne, M. Colón, B. Finkbeiner, M. Pichora, H. B. Sipma, and T. E. Uribe. An update on STeP: Deductive-algorithmic verification of reactive systems. In R. Berghammer and Y. Lakhnech, editors, *Tool Support for System Specification, Development and Verification*, Advances in Computing Sciences. Springer-Verlag, Berlin, 1999.
28. F. Mesnard and U. Neumerkel. Applying static analysis techniques for inferring termination conditions of logic programs. In Cousot [12], pages 93–110.
29. T. S. Motzkin, H. Raiffa, G. L. Thompson, and R. M. Thrall. The double description method. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games – Volume II*, number 28 in Annals of Mathematics Studies, pages 51–73. Princeton University Press, Princeton, New Jersey, 1953.
30. W. Pugh. A practical algorithm for exact array dependence analysis. *Communications of the ACM*, 35(8):102–114, 1992.
31. J. Stoer and C. Witzgall. *Convexity and Optimization in Finite Dimensions I*. Springer-Verlag, Berlin, 1970.
32. D. K. Wilde. A library for doing polyhedral operations. Master's thesis, Oregon State University, Corvallis, Oregon, December 1993. Also published as IRISA *Publication interne* 785, Rennes, France, 1993.