The Parma Polyhedra Library: Toward a Complete Set of Numerical Abstractions for the Analysis and Verification of Hardware and Software Systems ⋆

Roberto Bagnara [a], Patricia M. Hill [b], Enea Zaffanella [a]

[a] *Department of Mathematics, University of Parma, Italy*

[b] *School of Computing, University of Leeds, UK*

Abstract

Since its inception as a student project in 2001, initially just for the handling (as the name implies) of convex polyhedra, the *Parma Polyhedra Library* has been continuously improved and extended by joining scrupulous research on the theoretical foundations of (possibly non-convex) numerical abstractions to a total adherence to the best available practices in software development. Even though it is still not fully mature and functionally complete, the Parma Polyhedra Library already offers a combination of functionality, reliability, usability and performance that is not matched by similar, freely available libraries. In this paper, we present the main features of the current version of the library, emphasizing those that distinguish it from other similar libraries and those that are important for applications in the field of analysis and verification of hardware and software systems.

Key words: Formal methods, static analysis, computer-aided verification, abstract interpretation, numerical properties.

⋆ This work has been partly supported by PRIN project "AIDA: Abstract Interpretation Design and Applications."

Email addresses: `bagnara@cs.unipr.it` (Roberto Bagnara), `hill@comp.leeds.ac.uk` (Patricia M. Hill), `zaffanella@cs.unipr.it` (Enea Zaffanella).

1 Introduction

The *Parma Polyhedra Library* (PPL) is a collaborative project started in January 2001 at the Department of Mathematics of the University of Parma, Italy. Since 2002, the library is actively being developed also at the School of Computing of the University of Leeds, United Kingdom. The PPL, that was initially limited —as the name implies— to the handling of (not necessarily topologically closed) convex polyhedra [17], is now aimed at becoming a truly professional, functionally complete library for the handling of numeric approximations targeted at abstract interpretation and computer-aided verification of hardware and software systems.

In this paper we briefly describe the main features of the PPL. Unless otherwise stated, the description refers to version 0.9 of the library, released on March 12, 2006. In other cases the described features have not yet been included into an official release, but are still available from the PPL's public CVS repository (since development of the library takes place on that repository, the PPL is always in a state of continuous release).

In the sequel we will compare the PPL with other libraries for the manipulation of convex polyhedra. These are: [1]

- *PolyLib* (version 5.22.1), originally designed by D. K. Wilde [85], based on an efficient C implementation of N. V. Chernikova's algorithm [26,27,28] written by H. Le Verge [65], and further developed and maintained by V. Loechner [66]; [2]
- *New Polka* (version 2.1.0a), by B. Jeannet [62]; [3]
- the polyhedra library that comes with the *HyTech* tool (version 1.04f) [59]; [4]
- the *Octagon Abstract Domain Library* (version 0.9.8), by A. Miné [68,69]. [5]

For reasons of space and opportunity, the paper concentrates on introducing the library to prospective users. As a consequence, we assume the reader has some familiarity with the applications of numerical abstractions in formal analysis and verification methods and that she is not immediately concerned by the theory that lies behind those abstractions. However, the paper provides a complete set of references that enable any interested reader to reconstruct every detail concerning the library, the applications and the relevant theory.

[1] We restrict ourselves to those libraries that are freely available and provide the services required by applications in static analysis and computer-aided verification.
[2] `http://icps.u-strasbg.fr/~loechner/polylib/`.
[3] `http://pop-art.inrialpes.fr/people/bjeannet/newpolka/index.html`.
[4] `http://embedded.eecs.berkeley.edu/research/hytech/`.
[5] `http://www.di.ens.fr/~mine/oct/`.

The paper is structured as follows: Section 2 introduces the numerical abstractions currently supported by the Parma Polyhedra Library; Section 3 describes, also by means of examples, some of the most important features of the library; Section 4 gives some indications concerning efficiency; Section 5 illustrates the current development plans for the library; Section 6 reviews some of the applications using the PPL and concludes.

2 Currently Supported Abstractions

The numerical abstractions currently supported by the Parma Polyhedra Library are the domains of polyhedra, bounded difference shapes, octagonal shapes and grids; powersets of these using the generic powerset construction; and linear programming problems. For each of the supported abstractions, the library features all the operators required by semantic constructions based on *abstract interpretation* [31,32]; these are summarized in the following paragraphs.

Construction The domain elements can be initialized by means of a variety of constructors; in particular, they can be defined to be a vector space for a specified number of dimensions —each dimension representing some concrete entity relevant for the particular analysis— indicating that at this stage nothing is known about the entities, or the empty abstraction (again for a given number of dimensions), describing an inconsistent (unreachable) state. An element can also be initialized by means of *constraints*, specifying conditions its points must satisfy; alternatively, it can be described by *generators*, that are meant to be parametrically combined so as to describe all of its points. New elements can also be constructed by copying an existing element in the same abstraction.

Refinement and Expansion Any domain element can be refined by the addition of further constraints its points must satisfy, thereby improving the precision of the description: a typical application of refinement is to model the effect of conditional guards in if-then-else and while-loop statements. The expansion operators allow users to add new generators so as to expand the set of points the element must contain; these may be used, for instance, to "forget" some information regarding a space dimension, so as to model arbitrary input by the user.

Upper and Lower Bounds It is often useful to compute an upper or lower bound of two domain elements so as to obtain a correct approximation of their union or intersection. For example, an analyzer could use an upper bound operator to combine approximations that have been computed along the alternative branches of an if-then-else statement while lower bound operators are needed, in combination with conversion operators (see below), when conjunctively merging different approximations computed along the same set of computation paths.

Affine Images and Preimages A common form of statement in an imperative language is the assignment of an affine expression to a program variable. Their semantics can be modeled by images (in a forward analysis) and/or preimages (in a backward analysis) of affine transformations on domain elements. All domains fully support the efficient (and possibly approximate) computation of affine images and preimages. Also available are generalizations of these operators, useful for approximating more complex assignment statements (e.g., to model constrained nondeterministic assignments or when a non-linear expression can be bounded from below and/or above by affine expressions).

Changing Dimensions An analyzer needs to be able to add, remove, and more generally reorganize the space dimensions that are associated with the values of the concrete entities it is approximating. The simple addition and removal of dimensions is often needed at the entry and exit points of various kinds of programming contexts, such as declaration blocks where concrete entities (modeled by some of the space dimensions) may be local to the block. More complex operators are useful to support the integration of the results computed using different abstractions, that typically provide information about different sets of concrete entities. In the simplest case, when the abstractions are of the same kind but provide information about disjoint sets of concrete entities, it is enough to *concatenate* the two abstract elements into a new one. In more complex cases, when the described sets of concrete entities have an overlap, the space dimensions of one of the abstract elements need to be reconciled with those of the other, allowing for a correct integration of the information. This can be obtained by efficiently *mapping* the space dimensions according to a (partial) injective function. The library also supports the *folding* (and unfolding) of space dimensions, which is needed to correctly and efficiently summarize the information regarding collections of concrete entities [48].

Conversion Non-trivial analyses are typically based on a combination of domains. It is therefore important to be able to safely and efficiently convert between different abstractions. These conversions enable, for instance, the combination of domain elements representing different kinds of information, implementing so-called *reduction* operators. Another important application is the dynamic control of the precision/efficiency ratio: in order to gain efficiency in the analysis of a specific context, an element of a relatively precise domain can temporarily be converted into an element of a weaker domain and then back to the stronger abstraction on exit from that context.

Comparison The analysis of a program fragment is implemented by computing an over-approximation of its semantics. Since the latter is usually modeled as the least fixpoint of a continuous operator, a safe analysis typically needs to iteratively compute an over-approximation of this fixpoint. Therefore, a suitable lattice-theoretic comparison operator is needed so as to check for the convergence of the analysis. Comparison operators are also useful in selected contexts so as to efficiently predict whether or not some precision improving techniques (e.g., reductions, widening variants and so on) are applicable. For all of the reasons above, each domain provides three different comparison operators checking, respectively, equality, containment and strict containment.

Widening Most of the domains supported by the library admit *infinite ascending chains*. These are infinite sequences of domain elements such that every element in the sequence is contained in the element that follows it. With these characteristics, the fixpoint computations upon which abstract interpretation analysis techniques are based could be non-terminating. For this reason, the domains can be so employed only in conjunction with appropriate mechanisms for enforcing and/or accelerating the convergence of fixpoint computations: widening operators [30,31,33,34] provide a simple and general characterization for such mechanisms. [6] The PPL offers also several variations of the available widenings: [7] widening "with tokens" (an improvement to the widening delay technique proposed in [29]); and widening "up to" [53,57] (a technique whereby constraints that are judged to be important by the application can be preserved from the action of the widening).

Other Operators The library offers many other operators for use in a variety of more specialized contexts. For instance, before performing a non-trivial domain combination, an analyzer may need information about a particular domain element (such as, checking whether it denotes the empty set or the whole vector space; the dimension of the vector space; the affine dimension of

[6] Operators that do not provide a strict convergence guarantee are more properly called *extrapolation* operators.

[7] Some of these variations are extrapolation operators, as the guarantee of convergence is conditional on the way they are used.

the element; its relation with respect to a given constraint or generator, and so on). Another operator supported by the library is the *difference* operator, which computes the smallest domain element that contains the set difference of two elements; this is exploited in the implementation of the finite powerset widening operator proposed in [16]. The library also provides the *time-elapse* operator used to model hybrid systems [57].

The following sections briefly describe each of the supported domains. The emphasis here is on the features that are unique to the PPL. The reader is referred to the cited literature and to the library's documentation [14,15] for all the details.

2.1 Closed and Not Necessarily Closed Polyhedra

The Parma Polyhedra Library supports computations on the abstract domain of convex polyhedra [36,52]. The PPL implements both the abstract domain of topologically *closed convex polyhedra* (briefly called *C polyhedra* and implemented by class `C_Polyhedron`) and the abstract domain of *not necessarily closed convex polyhedra* (*NNC polyhedra* for short, class `NNC_Polyhedron`). In both cases, polyhedra are represented and manipulated using the *Double Description* (DD) method of Motzkin et al. [70]. In this approach, a closed convex polyhedron can be specified in two ways, using a *constraint system* (class `Constraint_System`) or a *generator system* (class `Generator_System`): the constraint system is a finite set of linear equality or inequality constraints (class `Constraint`); the generator system is a finite set of different kinds of vectors, collectively called *generators*, which are rays and points of the polyhedron (class `Generator`). An example of double description is depicted in Figure 1: the polyhedron represented by the shaded region can be represented by the set of vectors satisfying the constraints or, equivalently, by the set

$$\{\, \pi_1 \mathbf{p}_1 + \pi_2 \mathbf{p}_2 + \rho_1 \mathbf{r}_1 + \rho_2 \mathbf{r}_2 \in \mathbb{R}^2 \mid \pi_1, \pi_2, \rho_1, \rho_2 \in \mathbb{R}_+, \pi_1 + \pi_2 = 1 \,\},$$

where $\mathbf{p}_1 = \binom{4}{1}$, $\mathbf{p}_2 = \binom{1}{4}$, $\mathbf{r}_1 = \binom{1}{2}$, $\mathbf{r}_2 = \binom{2}{1}$, and $\mathbb{R}_+$ is the set of non-negative real numbers. In words, each vector can be obtained by adding a non-negative combination of the rays and a convex combination of the points.

Implementation of convex polyhedra using the DD method offer some important advantages to analysis and verification applications. The first one is due to the "mix" of operations such applications require: some of them are more efficiently performed on the representation with constraints. This is the case for the addition of constraints and for the intersection, which is simply implemented as the union of constraint systems, and for deciding whether a generator is subsumed or not by a polyhedron (e.g., to decide whether a point is inside or outside). Some operations are instead more efficiently performed

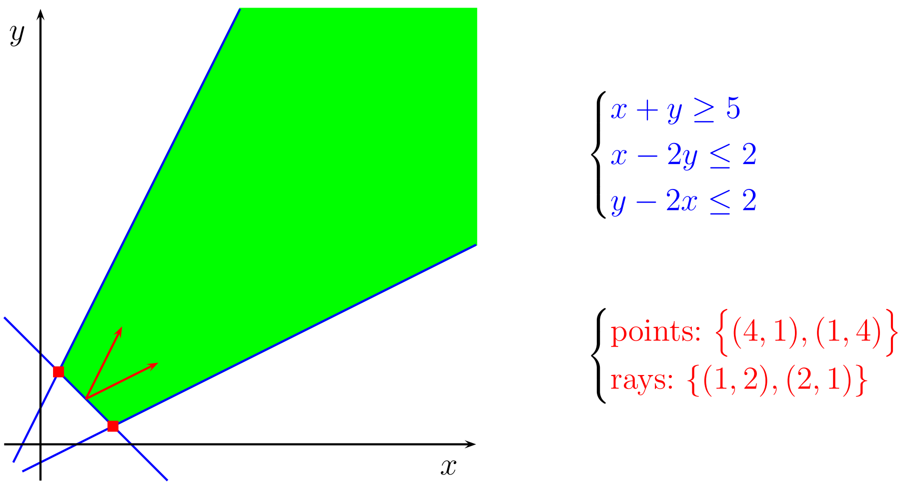


Figure 1. The double description method for polyhedra

on generators: computing the convex polyhedral hull (just by taking the union of generator systems), adding individual generators (e.g., the addition of the rays $\mathbf{r} = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$ and $-\mathbf{r}$ to the set of rays for the polyhedron in Figure 1 is the easiest way to "forget" all the information concerning the space dimension x), projection onto designated dimensions, deciding whether the space defined by a constraint is disjoint, intersects or includes a given polyhedron, finiteness/boundedness tests (a polyhedron is finite/bounded if and only if it has no rays among its generators), and the *time-elapse* operator of [56,57]. There are also important operations, such as the inclusion and equality tests and the widenings, that are more efficiently performed when *both* representations are available. Systems of constraints and generators enjoy a quite strong and useful duality property. Very roughly speaking, the constraints of a polyhedron are (almost) the generators of the *polar* [73,82] of the polyhedron, the generators of a polyhedron are (almost) the constraints of the polar of the polyhedron, and the polar of the polar of a polyhedron is the polyhedron itself. This implies that computing constraints from generators is the same problem as computing generators from constraints. The algorithm of N. V. Chernikova [26,27,28] (later improved by H. Le Verge [65] and by K. Fukuda and A. Prodon [47]) solves both problems yielding a minimized system and can be implemented so that the source system is also minimized in the process. This is basically the algorithm employed by PolyLib, New Polka and the Parma Polyhedra Library. It is worth noticing that it is not restrictive to express the coefficients of constraints and rays by integer numbers, as using rational numbers would not result in increased expressivity (but would have a negative impact on efficiency). For points, a common integer denominator suffices. [8]

[8] The correctness requirements of applications in the field of system's analysis and verification prevent the adoption of floating-point coefficients, since any rounding error on the wrong side can invalidate the overall computation. For domains as complicated as that of polyhedra, the correct, precise and reasonably efficient handling

Restricting the attention to convex polyhedra, two of the main innovations introduced by the PPL are the complete handling of NNC polyhedra and the introduction of a new widening operator. Apart from the PPL, the only libraries—among those that provide the services required by applications in static analysis and computer-aided verification—that support NNC polyhedra are the already mentioned New Polka and the library by N. Halbwachs, A. Kerbrat and Y.-E. Proy called, simply, *Polka* [54]. The Polka library, however, is not available in source format and binaries are distributed under rather restrictive conditions (until about the year 1996 they could be freely downloaded), so our knowledge of it is as given in [54], the programmer's manual in a package that includes the actual library. The support provided by Polka and New Polka for NNC polyhedra is incomplete, incurs avoidable inefficiencies and leaves the client application with the non-trivial task of a correct interpretation of the results. In particular, even though an NNC polyhedron can be described by using constraint systems that may contain strict inequalities, the Polka and New Polka libraries lack a corresponding extension for generator systems. In contrast, the PPL implements the proposal put forward in [13], whereby the introduction of *closure points* as a new kind of generator, allows a clean user interface, symmetric to the constraint system interface for NNC polyhedra, that is decoupled from the implementation. As explained in detail in [13], a user of the PPL is fully shielded from implementation details such as the extra ϵ dimension that users of the other libraries have to carefully take into account. Another feature that is unique to the PPL is the support for the minimization of the descriptions of an NNC polyhedron: we refer the interested reader to [13] for a precise account of the impact this new feature has on performance and usability.

The original widening operator proposed by Cousot and Halbwachs [36,52] is termed *standard widening* since, for 25 years, all analysis and verification tools that employed convex polyhedra also employed that operator. Nonetheless, there was an unfulfilled demand for more precise widening operators. The Parma Polyhedra Library, besides the standard widening, offers the new widening proposed in [12]: on a single application this is always more precise than the standard widening. As these widenings are not monotonic, increased precision on a single application does not imply increased precision on the final result of the analysis. In practice, however, an overall increase of precision is almost always achieved [12].

Both widenings can be improved, as said before, by applying the "widening with tokens" delay strategy or the "widening up-to" technique; moreover, "bounded" extrapolation operators are available that provide additional precision guarantees over the widenings upon which they are built.

---

of floating-point rounding errors is an open issue.

By restricting to particular subclasses of linear constraints, it is possible to obtain domains that are simpler and computationally more efficient than the one of convex polyhedra. One possibility, which has a long tradition in computer science [20], is to only consider *potential constraints*, also known as *bounded differences*: these are restricted to take the form $v_i - v_j \leq d$ or $\pm v_i \leq d$, where v_i and v_j are variables and d, the *inhomogeneous term*, belongs to some computable number family. Systems of bounded differences have been used by the artificial intelligence community as a way to reason about temporal quantities [1,37], as well as by the model checking community as an efficient yet precise way to model and propagate timing requirements during the verification of various kinds of concurrent systems [38,64]. In the abstract interpretation field, the idea of using an abstract domain of bounded differences was put forward in [6] and the first fully developed application of bounded differences in this field can be found in [80]. Possible representations for finite systems of bounded differences are matrix-like data structures called *difference-bound matrices* (DBM) [20] and weighted graphs called *constraint networks* [37]. These representations, however, have a "syntactic" nature: they encode sets of constraints rather than geometric shapes. As pointed out in [10] this nature has several drawbacks, the most important one being that natural extrapolation operators do not provide a convergence guarantee, that is, they are not widenings. This results into an extra burden on the client application, which has to take into account the implementation details and use the domain elements with care so as to avoid non-termination of the analysis.

In order to overcome the difficulties mentioned above and to continue pursuing a complete separation between interface (which must be natural and easy to use) and implementation (which must be efficient and robust), the Parma Polyhedra Library offers the "semantic" domain of *bounded difference shapes*. A bounded difference shape is nothing but a geometric shape, that is, a convex polyhedron: its internal representation needs not concern (and, in fact, is completely hidden from) the client application. The class template implementing this domain in the PPL is `BD_Shape<T>`, where the class template type parameter `T` defines the family of numbers that are used to (correctly) approximate the inhomogeneous terms of bounded differences. The value of `T` may be one of the following:

- a bounded precision native integer type (that is, from `signed char` to `long long` and from `int8_t` to `int64_t`);
- a bounded precision floating point type (`float`, `double` or `long double`);
- an unbounded integer or rational type, as provided by GMP (`mpz_class` or `mpq_class`).

Among other things, PPL's `BD_Shape<T>` offers the proper widening operator defined in [10] and a user interface that matches the interfaces of the general polyhedra classes `C_Polyhedron` and `NNC_Polyhedron`.

2.3 Octagonal Shapes

Another restricted class of linear constraints was introduced in [19]. These are of the form $av_i + bv_j \leq d$, where $a, b \in \{-1, 0, +1\}$ and d belongs to some computable number family. Systems of such constraints were called *simple sections* in [19] and have been given the structure of an abstract domain by A. Miné [68]. The resulting *octagon abstract domain* has, due to its syntactic nature, the same problems outlined in the previous section. This is why, as explained in detail in [10], the Parma Polyhedra Library offers a semantic domain of *octagonal shapes*, for which it provides a widening operator. This is implemented by the class template `Octagonal_Shape<T>`, where the class template type parameter `T` can be instantiated as for bounded difference shapes. [9] Another feature of this class is that its implementation uses the strong closure algorithm introduced in [10], which has lower complexity than the one used in the Octagon Abstract Domain Library [10] [68,69].

2.4 Grids

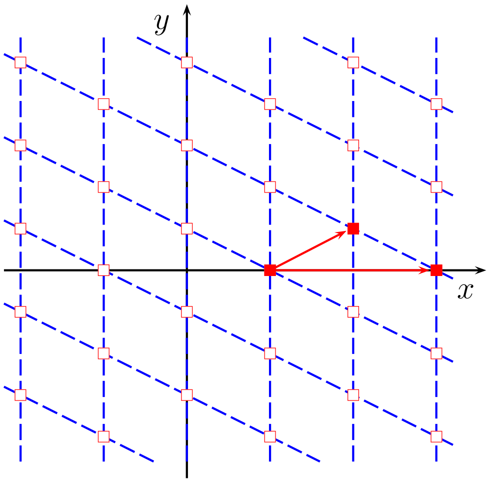
Given $a_1, \ldots, a_n, b, f \in \mathbb{Z}$, the *linear congruence relation* $a_1 v_1 + \cdots + a_n v_n \equiv_f b$ denotes the subset of $\mathbb{R}^n$ given by

$$\left\{ \langle q_1, \ldots, q_n \rangle \in \mathbb{R}^n \;\middle|\; \exists \mu \in \mathbb{Z} \,.\, \sum_{i=1}^{n} a_i q_i = b + \mu f \right\};$$

when $f \neq 0$, the relation is said to be *proper*; when $f = 0$, the relation is equivalent to (i.e., it denotes the same hyperplane as) $a_1 v_1 + \cdots + a_n v_n = b$. A *congruence system* is a finite set of congruence relations and a *grid* is any subset of $\mathbb{R}^n$ whose elements satisfy all the congruences of such a system. The *grid domain* is the set of all such grids.

An example of grid is given in Figure 2, where the solutions of the congruence relations are given by the dashed lines and the grid elements by the (filled and unfilled) squares. The figure shows also that there is an alternative way of describing the same grid: if we call the points marked by the filled squares

[9] Support for octagonal shapes has not yet been included into a formal release. It is however complete and available in the PPL's public CVS repository.
[10] Until at least version 0.9.8.



$$\begin{cases} x \equiv_2 0 \\ x + 2y \equiv_4 2 \end{cases}$$

$$\Big\{\text{points: } \big\{(2,0),(6,0),(4,1)\big\}$$

$$\begin{cases} \text{points: } \big\{(2,0)\big\} \\ \text{parameters: } \big\{(4,0),(2,1)\big\} \end{cases}$$

Figure 2. The double description method for grids

$\mathbf{p}_1 = \binom{2}{0}$, $\mathbf{p}_2 = \binom{6}{0}$ and $\mathbf{p}_3 = \binom{4}{1}$, we can see that the grid is given by

$$\{\, \pi_1\mathbf{p}_1 + \pi_2\mathbf{p}_2 + \pi_3\mathbf{p}_3 \in \mathbb{R}^2 \mid \pi_1, \pi_2, \pi_3 \in \mathbb{Z}, \pi_1 + \pi_2 + \pi_3 = 1 \,\}. \tag{1}$$

We say that the set of *points* $\{\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3\}$ *generates* the grid. Some of these generating points can be replaced by *parameters* that give the direction and spacing for the neighboring points. Specifically, by subtracting the point $\mathbf{p}_1$ from each of the other two generating points $\mathbf{p}_2$ and $\mathbf{p}_3$, we obtain the parameters $\mathbf{q}_2 = \binom{4}{0}$ and $\mathbf{q}_3 = \binom{2}{1}$ and can now express the grid as

$$\{\, \mathbf{p}_1 + \pi_2\mathbf{q}_2 + \pi_3\mathbf{q}_3 \in \mathbb{R}^2 \mid \pi_2, \pi_3 \in \mathbb{Z} \,\}.$$

Notice that, in the generator representation for grids, points and parameters can have rational, non integer coordinates.

The domain of grids, as briefly described above, has been introduced by P. Granger [50,51] and its design has been completed in [8,9] by including refined algorithms and new operators for affine images, preimages and their generalizations, grid-difference and widening. The Parma Polyhedra Library includes the first truly complete implementation of this abstract domain by means of the `Grid` class. Congruence relations and systems thereof are realized by the classes `Congruence` and `Congruence_System`, and likewise for generators with the classes `Grid_Generator`, and `Grid_Generator_System`.

A more restricted domain, the domain of *integral lattices* has been partly implemented in PolyLib [66] following the approach in [74,75]. An integral lattice in dimension n is the grid generated by the affine integral combination —as in (1)— of exactly $n+1$ affinely independent points that are additionally bound to have integer coordinates. These restrictions have the consequences that only the generator representation is supported and, as explained in [8,9], the reduced expressivity has an impact on the possibility to solve concrete analysis problems. The implementation of the domain of integral lattices in PolyLib is incomplete in the sense that it misses domain operators such as the join. Instead, it provides a union operator which, given two lattices returns a *set* of lattices representing the precise set union of the points of the given lattices. Similarly the difference operation on a pair of lattices returns a set of lattices whose union contains the exact set difference of their points. Moreover, although image and preimage operators are supported by PolyLib, as the integral lattice must be full dimensional, only invertible image operators are allowed.

2.5 Powersets

For applications requiring a high degree of precision, the Parma Polyhedra Library provides a generic implementation of the *finite powerset* construction [7,16]. This upgrades an abstract domain into a refined one where finite disjunctions of non redundant elements are precisely representable. The construction is implemented generically by the class template `Powerset<D>`, where the type parameter `D` must provide about ten methods and operators implementing, among other things, an entailment predicate and operators for obtaining an upper bound and the "meet" of two given elements. No other requirements are imposed and, in fact, the test suite of the PPL includes a program where `Powerset<D>` is instantiated with a non numerical domain.

The class template `Pointset_Powerset<PS>` provides a specialization of class `Powerset<D>` that is suitable for the instantiation with the "semantic" numerical domains of the PPL: (C or NNC) polyhedra, bounded difference shapes, octagonal shapes, grids and combinations thereof. A most notable and, at the time of writing, unique feature of this implementation is the provision —in addition to the extrapolation operator proposed in [25]— of provably correct, *certificate-based* widening operators that lift the widening operators defined on the underlying domain `PS` [16].

2.6 Linear Programming Problems

The library includes a Linear Programming (LP) solver based on the simplex algorithm and using exact arithmetic. Note that the absence of rounding errors is essential for most (if not all) of the intended applications. Since it is common practice to solve an LP problem by exploiting duality results, correctness may be lost even if *controlled rounding* is used; this is because a feasible but possibly non-optimal solution for the dual of an LP problem may actually correspond to an unfeasible solution for the original LP problem.

The LP solver interface allows for both satisfiability checks and optimization of linear objective functions. A limited form of incrementality allows for the efficient re-optimization of an LP problem after modification of the objective function. Ongoing implementation work is focusing on improving the efficiency of the solver as well as providing better support for incremental computations, so as to also allow for the efficient re-optimization of the LP problem after a modification of the feasible region caused by the addition of further constraints. [11]

3 Main Features

In this section we will briefly review the main features of the Parma Polyhedra Library. We will focus on usability, on the absence of arbitrary limits due to the use of fully dynamic data structures, on some of the measures that contribute to the library's robustness, on the support for resource-bounded computations, on the possibility to use machine integer coefficients without compromising correctness, on portability and on the availability of complete documentation.

3.1 Usability

By "usability" of the Parma Polyhedra Library we actually mean two things:

(1) that the library provides natural, easy to use interfaces that can be used, even by the non expert, to quickly prototype an application;
(2) that, nonetheless, the library and its interfaces provide all the functionalities that allow their use in the development of professional, reliable applications.

In other words, simplicity of the interfaces has not been obtained with simplistic solutions or by the omission of functionalities.

As mentioned before particular care has been taken (to the point of developing the necessary theoretical concepts) in the complete decoupling of the user interfaces from all implementation details. So, the internal representation of, say, constraints, congruences, generators and systems thereof need not concern the client application. All the user interfaces, whatever language they interface to, refer to high-level concepts and never to their possible implementation in terms of vectors, matrices or other data structures. For instance, unlike

[11] The incremental LP solver has not yet been included into a formal release. It is however complete and available in the PPL's public CVS repository.

PolyLib and New Polka, implementation devices (such as so-called *positivity constraints* [85] or *ϵ-representations* [13,56]) never surface at the user level and need not concern the client application. As another example, a user of the Octagon Abstract Domain Library must know that octagons are represented there by means of difference-bound matrices, that some of the operations on octagons do "close" these matrices, and that one argument to the widening is better closed for improved accuracy while the other should *not* be closed as this would prevent convergence.

The Parma Polyhedra Library currently offers, besides the C++ interface, a portable Prolog interface and a C interface. The Prolog interface is "portable" in that it supports (despite the lack of standardization of Prolog foreign language interfaces) six major Prolog systems: Ciao, GNU Prolog, SWI-Prolog, SICStus, XSB and YAP. The C interface is particularly important, as it allows to interface the PPL with almost anything else: for example, third parties have already built interfaces for Haskell, Java and Objective Caml. The design of the interfaces to directly supported languages has been focused on ensuring that programmers can use the library following the most natural programming style in *that* language. As a simple example, 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. This can be done because both languages allow to override (or freely interpret) operators and provide exceptions as a powerful method of reporting run-time errors. Here is how a NNC polyhedron in a space of dimension 3 can be created using the C++ interface:

```
#include <ppl.hh>

namespace PPL = Parma_Polyhedra_Library;

  ...
  PPL::Variable X(0);
  PPL::Variable Y(1);
  PPL::Variable Z(2);
  PPL::NNC_Polyhedron ph(3, PPL::UNIVERSE);
  ph.add_constraint(X + 2*Y + 5*Z >= 7);
  ph.add_constraint(X < 5*Z);
  ...
```

And here is how the same polyhedron can be created using the Prolog interface:

```
...
numbervars([X, Y, Z], 0, _),
ppl_new_NNC_Polyhedron_from_constraints(
  [X + 2*Y + 5*Z >= 7, X < 5*Z],
  PH
),
...
```

In standard C things are more complicate as the language syntax does not allow to represent, say, constraints as easily.[12] Thus, in order to build constraints the C application will have to build the linear expressions occurring in them and the memory used to hold these intermediate data structures will have to be explicitly managed, unless a conservative garbage collector is used. Moreover, lack of exceptions means that non-trivial error detection and handling will demand significantly more effort in C than in C++ or Prolog (*all* the functions of the C interface return an `int`: a negative value indicates an error has occurred). The best approach to development using the C interface is to begin by developing a layer of interface functions that are suited to the application at hand: the PPL's C interface provides all the required services.

Still on the subject of ease of use, the numeric abstractions provided by the library provide similar interfaces. A high degree of integration is obtained through the adoption of common data types and the availability of methods specifically designed for interoperability. As a consequence, the specification and implementation of new abstractions and tools can be easily obtained by composition of the available services. As a simple example, the code in Listing 1 is exploiting the LP solver capabilities to efficiently compute (and print) all the upper bounds for the variables corresponding to the dimensions of a closed polyhedron.

Note that the code in Listing 1 is taking advantage of a limited form of incrementality provided by the LP solver: the check for satisfiability (corresponding to the first phase of the simplex algorithm) is executed only once and it is not repeated when optimizing the different objective functions (namely, only the second phase of the simplex algorithm is executed in the for-loop). Code similar to the one above is actually used in the library itself to precisely approximate a polyhedron by means of a bounded difference or octagonal shape without incurring the potentially high cost of converting from the constraint to the generator representation of the polyhedron.

[12] Unless one represents them with strings. While a string-based interface is certainly possible, we do not believe it would make things simpler: strings would have to be parsed at run-time and parsing errors would have to be properly handled.

Listing 1. Printing upper bounds for all variables using the LP solver

```
#include <ppl.hh>
#include <iostream>

using namespace Parma_Polyhedra_Library;
using namespace Parma_Polyhedra_Library::IO_Operators;

void print_upper_bounds(const C_Polyhedron& ph) {
  LP_Problem lp(ph.constraints());
  // Check the satisfiability of the problem.
  if (!lp.is_satisfiable()) {
    std::cout << "unsatisfiable" << std::endl;
    return;
  }
  // Print the upper bound of each variable.
  lp.set_optimization_mode(MAXIMIZATION);
  Generator g(point());
  Coefficient num, den;
  const dimension_type dim = ph.space_dimension();
  for (dimension_type i = 0; i < dim; ++i) {
    Variable x(i);
    lp.set_objective_function(x);
    LP_Problem_Status status = lp.solve();
    if (status == UNBOUNDED_LP_PROBLEM)
      std::cout << x << " < +infty" << std::endl;
    else {
      assert(status == OPTIMIZED_LP_PROBLEM);
      g = lp.optimizing_point();
      lp.evaluate_objective_function(g, num, den);
      std::cout << x << " <= " << num << "/" << den
                << std::endl;
    }
  }
}
```

3.2 Absence of Arbitrary Limits

The only real restrictions imposed by the library on the client application are those caused by limitations of the available virtual memory. All data structures are fully dynamic and automatically expand (in amortized constant time) and shrink in a way that is completely transparent to the user, ensuring the best use of available memory.

In contrast, in the PolyLib, New Polka and HyTech libraries, 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.

3.3 Robustness

The clean separation between interface and implementation, among other important advantages, allows for the adoption of incremental and lazy computation techniques. The increased efficiency due to these techniques amply repays the cost of the interface checks that contribute to the library's robustness, which, as it will be explained in the sequel, is one of its most important features.

First, the library systematically checks all the interface invariants and throws an exception if any one of them is violated. This makes it very difficult to inadvertently create invalid objects and greatly simplifies the debugging of client applications. Secondly, the library is exception-safe, that is, it never leaks resources or leaves invalid object fragments around, even in the presence of exceptions. In particular, if an exception is raised, then all the memory allocated by the failed computation is discarded. These features allows applications using the PPL to use timeouts or to sensibly deal with run-time errors such as arithmetic overflows and out-of-memory conditions, so as to continue the computation in a reliable state (see below for more on this subject). It is important to stress that, while error handling and recovery are somewhat optional features (there may be no interest in continuing the computation after an error has occurred), error detection should be considered a mandatory feature in the field of system's analysis and verification, since failure to detect an error can easily result into undefined (i.e., completely unpredictable) behavior, therefore compromising any statement about correctness.

For comparison, PolyLib detects only some errors, sometimes setting a flag and sometimes printing a message and aborting, whereas New Polka and the HyTech libraries detect some errors, print an error message and abort. The Octagon Abstract Domain Library makes no attempt at detecting errors. None of these libraries perform a systematic check of interface invariants.

The library, thanks to its exception-safety characteristics, naturally supports resource-bounded computations, that is, computations that are limited in the amount of CPU time or virtual memory or both. The client application can take advantage of this feature by attempting a computation that is potentially very expensive imposing a maximum limit on the resources to be used. Should this bound be exceeded, an exception is raised and the client application can resort to a simplified computation (possibly using a simpler numerical abstraction) trusting that the PPL will release all the resources allocated by the interrupted computation. With these facilities at hand, users of the library can quite easily code resource-bounded or, at least, resource-conscious numerical abstractions. An example is shown in Listing 2. This uses the *Parma Watchdog Library* (PWL), a library that virtualizes the interval timers for any XSI-conforming implementation of UNIX.[13] The PWL, which is currently distributed with the PPL, gives an application the ability to work with an unbounded number of independent "watchdog" timers.

The example class `Up_Appr_Polyhedron` is meant to provide convex polyhedra with "upward approximated", resource-conscious operations. As the main representation it uses closed convex polyhedra; the user can set a timeout for the operations and externally impose a limit on the virtual memory available to the process. When a resource limit is reached, the class temporarily switches to a simpler family of polyhedra: bounded difference shapes. Listing 3 shows a simple implementation for the intersection operation. If no timeout has been requested, then the operation is performed without any overhead. Otherwise the polyhedra are copied, a watchdog timer, `w`, is set and the operation is attempted. When `w` expires, the PPL is asked to abandon all expensive computations and to throw `Timeout_object`. Alternatively, if the process exceeds the virtual memory it has been allotted, then the `bad_alloc` standard exception will be thrown. If none of those happen, then control will be returned to the caller. Otherwise, bounded difference shapes approximating the argument polyhedra will be computed using a polynomial complexity method,[14] these shapes will be intersected and the intersection will be used to construct the resulting convex polyhedron.

The technique illustrated in a simplified way by Listings 2 and 3 is quite powerful and allows to deal with the complexity-precision trade-off in a very flexible way. In particular, it is possible, thanks to the PWL, to work with

[13] XSI it is the core application programming interface for C and shell programming for systems conforming to the *Single UNIX Specification*.

[14] As mentioned in Section 2.6, it is also possible to compute these approximations using a "simplex complexity" algorithm (i.e., theoretically exponential but very efficient in practice).

Listing 2. Declaration of a class for resource-conscious computations

```
#include <ppl.hh>
#include <pwl.hh>
#include <stdexcept>

namespace PPL = Parma_Polyhedra_Library;
namespace PWL = Parma_Watchdog_Library;

class Up_Appr_Polyhedron {
private:
  // The type of the main polyhedral abstraction.
  typedef PPL::C_Polyhedron PH;

  // The type of a simpler polyhedral abstraction.
  typedef PPL::BD_Shape<mpq_class> SPH;

  PH ph;

  // Timeout in hundredth of a second; 0 for no timeout.
  static unsigned long timeout_hs;

  class Timeout : public PPL::Throwable {
    ...
  };

  static Timeout Timeout_object;

public:
  static void set_timeout(unsigned long n);
  ...
  void intersection_assign(const Up_Appr_Polyhedron& y);
  ...
};
```

multiple timers: while individual polyhedra operations can be guarded by a timer, other timers can monitor operations of greater granularity, such as entire analysis phases. When an analysis phase is taking too much, then the timeouts used for the individual operations can be shortened or the analyzer can switch to a totally different, less complex analysis technique. It is worth observing that, notwithstanding the friendliness of the PPL's user interfaces, professional applications in the field of system's analysis and verification are not expected to be directly based on the abstractions provided by the library.

Listing 3. Definition of a resource-conscious intersection method

```
void Up_Appr_Polyhedron
::intersection_assign(const Up_Appr_Polyhedron& y) {
  if (timeout_hs == 0) {
    // No timeout: do the operation directly.
    ph.intersection_assign(y.ph);
    return;
  }

  // Save copies of 'ph' and 'y.ph': they may be needed
  // to recover from a resources exhaustion condition.
  PH xph_copy = ph;
  PH yph_copy = y.ph;

  try {
    PWL::Watchdog w(timeout_hs,
                    PPL::abandon_expensive_computations,
                    Timeout_object);
    ph.intersection_assign(y.ph);
    PPL::abandon_expensive_computations = 0;
    return;
  }
  catch (const Timeout&) {
    // Timeout expired.
  }
  catch (const std::bad_alloc&) {
    // Out of memory.
  }

  // Resources exhausted: use simpler polyhedra.
  PPL::abandon_expensive_computations = 0;

  SPH xph_simple(xph_copy, PPL::POLYNOMIAL_COMPLEXITY);
  SPH yph_simple(yph_copy, PPL::POLYNOMIAL_COMPLEXITY);
  xph_simple.intersection_assign(yph_simple);
  ph = PH(xph_simple.minimized_constraints());

  // Restore 'y.ph'.
  const_cast<PH&>(y.ph) = yph_copy;
}
```

Rather, the PPL abstractions have been designed so as to serve as building blocks for the actual analysis domains: in this field the complexity-precision trade-off is often so serious that the right way to face it is, by necessity, application-dependent.

3.5 Unbounded or Native Integer Coefficients

For the representation of general convex polyhedra, with the default configuration, the Parma Polyhedra Library uses unbounded precision integers. On the other hand, if speed is important and the numerical coefficients involved are likely to be small, applications may use native integers (8, 16, 32 and 64 bit integers are supported by the PPL). This is a safe strategy since, when using native integers, the library also performs systematic (yet efficient) overflow detection. It is thus possible to adopt an approach whereby computations are first attempted with native integers. If a computation runs to completion, the user can be certain that no overflow occurred. Otherwise an exception is raised (as in the case seen before for resource-bounded computations), so that the client application can be restarted with bigger native integers or with unbounded integers. This is another application of the library's exception-safety, as one can rather simply code the above approach as follows:

```
try {
  // Analyze with 64-bit coefficients.
  ...
}
catch (const std::overflow_error&) {
  // Analyze with unbounded coefficients.
  ...
}
...
```

Again, the client application does not need to be concerned about the resources allocated by the PPL during the computation of the `try` block: everything will be deallocated automatically.

Concerning other libraries, PolyLib and New Polka can use unbounded integers as coefficients, whereas the library of HyTech does not support them. Differently from the PPL, these libraries use finite integral types without any mechanism for overflow detection. Technically speaking and according to the C standard (the language in which they are written), this means that the effects of an overflow are completely undefined, i.e., client applications cannot make any assumption about what can happen should an overflow occur. In addition, PolyLib (and, according to [21], some versions of HyTech) can use floating point values, in which case underflows and rounding errors, in addition to overflows, can affect the results.

3.6 Portability and Documentation

Great care has been taken to ensure the portability of the PPL. The library is written in standard C++, it follows all the available applicable standards and uses sophisticated automatic configuration mechanisms. It is known to run on all major UNIX-like operating systems, on Mac OS X (whether Intel- or PowerPC-based) and on Windows (via Cygwin or MinGW).

A big investment has also been made on documentation and at several levels. First, the theoretical underpinnings have been thoroughly investigated, integrated when necessary and written down: an extensive bibliography is available on the PPL web site. Secondly, during the entire development of the library, the quality, accessibility and completeness of the documentation has always been given a particular emphasis: while some parts of the library need more work in this respect, the vast majority of the code is thoroughly documented and some parts of it approach the ideal of "literate programming."

The library has been documented using the Doxygen tool.[15] Doxygen is a documentation system for C++, C, Java, and other languages that allows to generate high-quality documentation from a collection of documented source files. The source files can be documented by means of ordinary comments, that can be placed near the program elements being documented: just above the declaration or definition of a member, class or namespace, for instance. This makes it much easier to keep the documentation consistent with the actual source code. Moreover, Doxygen allows the typesetting of mathematical formulas within comments by means of the relevant subset of LaTeX, which is an important feature for a project like the PPL. It is also able to automatically extract the code structure and use this information to generate include dependency graphs, inheritance diagrams, and collaboration diagrams. Doxygen can generate documentation in various formats, such as HTML, PostScript and PDF. The HTML and PDF output are fully hyperlinked, a feature that greatly facilitates "navigation" in the available documentation.

The Parma Polyhedra Library is equipped with two manuals generated with the help of Doxygen: a user's manual, containing all and only the information needed by people wishing to use the library [15]; and a developer's reference manual that contains, in addition, all the details concerning the library implementation [14]. All manuals are available, in various formats, from the PPL web site and the user's manual is also included in each source distribution.

[15] `http://www.doxygen.org`.

4 Efficiency

One natural question is how does the efficiency of the Parma Polyhedra Library compare with that of other polyhedra libraries. Of course, such a question does not have a definite answer. Apart from clarifying whether CPU or memory efficiency or both are the intended measures of interest, the answer will depend on the targeted applications: with different applications the results can vary wildly. Moreover, even within the same application, big variations may be observed for different inputs. For these reasons, it must be admitted that the only way to meaningfully assess the performance of the library is with respect to a particular application, a particular set of problem instances, and a particular definition of 'performance'.

For the same reasons, it is nonetheless instructive to compare the performance of various polyhedra libraries on a well-defined problem with a large set of freely available inputs. One such problem, called *vertex/facet enumeration*, is particularly relevant for implementations based on the Double Description method such as the Parma Polyhedra Library, New Polka and PolyLib, as this problem has to be solved whenever one description has to be converted into the other one. The vertex/facet enumeration problem is a well-studied one and several systems have been expressly developed to solve it. We have thus compared the above mentioned libraries with the following (in parentheses, the versions we have tested):

- cddlib (version 0.94b), a C implementation of the Double Description method, by K. Fukuda [47]; [16]
- lrslib (version 0.42b), a C implementation of the reverse search algorithm for vertex enumeration/convex hull problems, by D. Avis [3,4]; [17]
- pd (version 1.7), a C program implementing a primal-dual algorithm using rational arithmetic, by A. Marzetta and maintained by D. Bremner [24]. [18]

Both cddlib and lrslib come with driver programs that support a polyhedra input format that was introduced by K. Fukuda and extended by D. Avis; this input format is also supported by the pd program. The distributions of cddlib and lrslib provide more than 100 different inputs of varying complexity for these programs. Driver programs that can read the same input format and use the PPL, New Polka and PolyLib are part of the PPL distribution since version 0.7.

The tests have been performed on a PC equipped with an AMD Athlon 2800+ with 1 GB of RAM and running GNU/Linux and GMP version 4.2. All the

[16] `http://www.cs.mcgill.ca/~fukuda/soft/cdd_home/cdd.html`.
[17] `http://cgm.cs.mcgill.ca/~avis/C/lrs.html`.
[18] `http://www.cs.unb.ca/profs/bremner/pd/`.

software has been compiled with GCC 4.0.3 at the optimization level that is the default for each package (i.e., the PPL was compiled with '`-O2`', its default; PolyLib, cddlib, and pd with `-O2`; New Polka and lrslib with '`-O3`'). The obtained running times, in seconds, are reported in Tables 1 and 2.[19] The entries marked with 'n.a.' in the pd's column indicate the problems that cannot be solved by pd, which can only handle polyhedra that contain the origin. Entries marked with 'ovfl' indicate the problems on which pd runs into an arithmetic overflow. It should be noted that strictly speaking, lrslib solves a slightly different, easier problem than the one solved by the other systems: while the latter guarantee the result is minimized, the output of lrslib may contain duplicate rays.

Table 1: Efficiency on vertex enumeration

input	PPL	New Polka	PolyLib	cddlib	lrslib	pd
`ccc4.e`	0.00	0.11	0.03	0.00	0.00	n.a.
`ccc5.e`	0.00	0.10	0.04	0.02	0.00	n.a.
`ccc6.e`	0.03	0.14	0.26	0.61	3.12	n.a.
`ccp4.e`	0.00	0.10	0.02	0.00	0.00	0.01
`ccp5.e`	0.00	0.10	0.04	0.02	0.00	5.36
`ccp6.e`	0.05	0.15	0.31	0.90	3.95	$> 1h$
`cp4.e`	0.00	0.08	0.03	0.00	0.00	0.01
`cp5.e`	0.00	0.12	0.05	0.02	0.00	5.29
`cp6.e`	0.05	0.18	0.30	0.88	3.86	$> 1h$
`cube.e`	0.00	0.05	0.02	0.00	0.00	0.00
`cut16_11.e`	0.00	0.12	0.04	0.02	0.00	3.86
`cut32_16.e`	0.05	0.17	0.28	0.91	4.32	$> 1h$
`cyclic10-4.e`	0.00	0.07	0.02	0.00	0.00	0.00
`cyclic12-6.e`	0.00	0.08	0.03	0.01	0.00	0.44
`cyclic14-8.e`	0.00	0.09	0.04	0.06	0.01	172.10
`cyclic16-10.e`	0.02	0.14	0.07	0.24	0.04	$> 1h$
`dcube10.e`	0.02	0.13	0.10	0.23	0.02	$> 1h$
`dcube12.e`	0.17	0.22	2.66	1.29	0.10	$> 1h$
`dcube3.e`	0.00	0.08	0.02	0.00	0.00	0.00
`dcube6.e`	0.00	0.07	0.03	0.00	0.00	0.35
`dcube8.e`	0.00	0.12	0.03	0.04	0.00	119.53
`irbox20-4.e`	0.00	0.08	0.01	0.01	0.00	0.01
`irbox200-4.e`	0.02	0.07	0.02	0.98	0.05	0.18
`mp5.e`	0.00	0.11	0.05	0.59	0.84	3.93
`prodst62.e`	34.78	161.09	123.28	$> 1h$	$> 1h$	ovfl
`redcheck.e`	0.00	0.06	0.01	0.00	0.00	0.00
`reg24-5.e`	0.00	0.08	0.03	0.01	0.00	0.01
`reg600-5_m.e`	5.05	19.12	12.37	135.49	33.63	n.a.
`samplev1.e`	0.00	0.09	0.02	0.00	0.00	n.a.
`samplev2.e`	0.00	0.06	0.02	0.00	0.00	n.a.
`samplev3.e`	0.00	0.07	0.02	0.00	0.00	n.a.

[19] Filenames have been shortened to fit the table on the page: in particular the `.ext` and `.ine` extensions have been shortened to `.e` and `.i`, respectively; moreover, the file called `integralpoints.ine` has been renamed `integpoints.i`.

Table 1: Efficiency on vertex enumeration (continued)

input	PPL	New Polka	PolyLib	cddlib	lrslib	pd
tsp5.e	0.00	0.12	0.04	0.00	0.00	n.a.
allzero.i	0.00	0.06	0.01	0.00	0.00	n.a.
cp4.i	0.00	0.09	0.02	0.00	0.00	n.a.
cp5.i	0.01	0.12	0.05	0.14	5.91	n.a.
cross10.i	0.08	0.22	0.14	10.50	$> 1h$	1.38
cross12.i	0.84	3.04	2.86	166.15	$> 1h$	15.24
cross4.i	0.00	0.07	0.02	0.00	0.00	0.00
cross6.i	0.00	0.06	0.03	0.04	0.09	0.01
cross8.i	0.01	0.07	0.03	0.54	28.90	0.14
cube.i	0.00	0.09	0.02	0.00	0.00	0.00
cube10.i	0.02	0.15	0.16	0.24	0.03	$> 1h$
cube12.i	0.19	0.32	3.70	1.35	0.18	$> 1h$
cube3.i	0.00	0.06	0.02	0.00	0.00	0.00
cube6.i	0.00	0.10	0.03	0.01	0.00	0.34
cube8.i	0.00	0.10	0.05	0.06	0.00	123.23
cubetop.i	0.00	0.08	0.02	0.00	0.00	n.a.
cubocta.i	0.00	0.10	0.02	0.00	0.00	0.00
cyc.i	0.00	0.08	0.01	0.00	0.00	0.00
cyclic17_8.i	0.04	0.15	0.14	0.32	0.08	$> 1h$
diamond.i	0.00	0.08	0.01	0.00	0.00	0.00
dodeca_m.i	0.00	0.07	0.02	0.00	0.00	n.a.
ex1.i	0.00	0.07	0.02	0.00	0.00	n.a.
grcubocta.i	0.00	0.06	0.02	0.01	0.00	0.01
hexocta.i	0.00	0.05	0.01	0.02	0.00	0.01
icododeca_m.i	0.00	0.07	0.02	0.05	0.00	n.a.
in0.i	0.00	0.06	0.02	0.00	0.00	n.a.
in1.i	0.00	0.08	0.02	0.01	0.00	n.a.
in2.i	0.00	0.08	0.03	0.00	0.00	n.a.
in3.i	0.00	0.05	0.02	0.00	0.00	n.a.
in4.i	0.00	0.10	0.03	0.01	0.00	n.a.
in5.i	0.00	0.07	0.04	0.03	0.00	n.a.
in6.i	0.02	0.14	0.09	0.42	0.03	n.a.
in7.i	0.18	0.40	0.44	0.96	0.08	n.a.
infeas.i	0.00	0.09	0.02	0.00	0.00	n.a.
integpoints.i	0.00	0.10	0.04	0.05	0.00	n.a.
kkd18_4.i	0.00	0.10	0.02	0.02	0.00	n.a.
kkd27_5.i	0.05	0.12	0.07	0.08	0.01	n.a.
kkd38_6.i	2.95	5.26	1.52	0.32	0.05	n.a.
kq20_11_m.i	0.19	0.41	0.45	0.99	0.08	n.a.
metric40_11.i	0.00	0.11	0.04	0.07	0.57	n.a.
metric80_16.i	0.13	0.24	0.07	0.54	32.09	n.a.
mit31-20.i	23.34	27.21	115.00	103.19	25.53	$> 1h$
mp5.i	0.00	0.08	0.04	0.06	0.58	n.a.
mp5a.i	0.00	0.10	0.03	0.06	0.57	n.a.
mp6.i	0.33	0.42	0.75	4.66	1177.96	n.a.
nonfull.i	0.00	0.06	0.02	0.00	0.00	n.a.
origin.i	0.00	0.08	0.02	0.00	0.00	n.a.
project1_m.i	0.00	0.10	0.04	0.03	0.00	1.10
project1res.i	0.00	0.08	0.02	0.00	0.00	0.00

Table 1: Efficiency on vertex enumeration (continued)

input	PPL	New Polka	PolyLib	cddlib	lrslib	pd
`project2_m.i`	0.02	0.09	0.05	0.46	0.13	n.a.
`project2res.i`	0.00	0.08	0.02	0.08	0.01	n.a.
`rcubocta.i`	0.00	0.08	0.01	0.01	0.00	0.01
`reg24-5.i`	0.00	0.08	0.02	0.01	0.00	0.01
`rhomtria_m.i`	0.00	0.09	0.02	0.04	0.00	n.a.
`sample.i`	0.00	0.08	0.01	0.00	0.00	0.00
`sampleh1.i`	0.00	0.05	0.02	0.00	0.00	n.a.
`sampleh2.i`	0.00	0.06	0.01	0.00	0.00	n.a.
`sampleh3.i`	0.00	0.04	0.02	0.00	0.00	n.a.
`sampleh4.i`	0.00	0.07	0.01	0.00	0.00	n.a.
`sampleh5.i`	0.00	0.05	0.01	0.00	0.00	n.a.
`sampleh6.i`	0.00	0.06	0.01	0.00	0.00	n.a.
`sampleh7.i`	0.00	0.09	0.01	0.00	0.00	n.a.
`sampleh8.i`	52.87	73.64	78.76	$> 1h$	4.59	n.a.
`trunc10.i`	8.81	9.06	0.08	1.66	9.15	737.37
`trunc7.i`	0.02	0.13	0.04	0.13	0.16	17.51
`tsp5.i`	0.00	0.10	0.04	0.02	0.00	n.a.
total	130.34	308.12	345.70			

In Table 2 we have collected the data concerning the hardest of these problems. Here we have imposed a memory limit of 768 MB: entries marked with 'mem' indicate the problems on which this limit was exceeded. The entries marked with 'tab' in the New Polka's column indicate the problems where New Polka ran "out of table space" in the conversion algorithm.

Table 2: Efficiency on vertex enumeration: hard problems

input	PPL	New Polka	PolyLib	cddlib	lrslib	pd
`cp7.e`	$> 1h$	tab	$> 1h$	$> 1h$	$> 1h$	$> 1h$
`cyclic25_13.e`	89.94	tab	354.83	221.98	12.10	n.a.
`cp6.i`	$> 1h$	tab	$> 1h$	$> 1h$	$> 1h$	n.a.
`mit.i`	$> 1h$	tab	$> 1h$	950.00	2024.16	n.a.
`mit288-281.i`	mem	tab	mem	mem	$> 1h$	ovfl
`mit41-16.i`	137.14	tab	320.37	325.65	33.89	$> 1h$
`mit708-9.i`	$> 1h$	tab	$> 1h$	1016.23	1927.18	n.a.
`mit71-61.i`	$> 1h$	tab	mem	$> 1h$	$> 1h$	n.a.
`mit90-86.i`	mem	tab	mem	$> 1h$	$> 1h$	ovfl

Another possibility of evaluating the performance of the Parma Polyhedra Library on a standard problem with standard data is offered by linear programming, which is the paradigm upon which several approaches to analysis and verification (such as, e.g., [76,78]) rest upon. This requires either a version of the simplex based on exact arithmetic, or, in case a classical floating-point implementation is used, it forces to validate the obtained result with some alternative methods. The *MathSAT* [20] decision procedure [23], which is applicable to the formal verification of infinite state systems (such as timed and hybrid systems), is based on a version of the *Cassowary Constraint Solving Toolkit* [5], modified so as to use exact arithmetic instead of floating-point numbers. Moreover, the algorithm employed by MathSAT requires incremental satisfiability checks: a set of constraints is added and satisfiability is checked, more constraints are added and satisfiability is re-checked, and so forth. We have thus measured the efficiency of the PPL's incremental constraint solver by comparison with the version of Cassowary used in MathSAT and with the *Wallaroo Linear Constraint Solving Library*, [21] another descendant of Cassowary. The benchmarks we used are quite standard in the linear programming community: they come from the '`lp`' directory of *NetLib*. [22] The solution times, in seconds, obtained for the problem of adding one constraint at a time, checking for satisfiability at each step, are given in Table 3.

Table 3: Efficiency of the simplex solver on incremental satisfiability checking

input	PPL	Wallaroo	Cassowary/MathSAT
`adlittle.mps`	0.33	1.46	1.51
`afiro.mps`	0.02	0.05	0.07
`blend.mps`	13.45	5.40	8.23
`boeing1.mps`	47.28	87.80	75.48
`boeing2.mps`	2.32	10.58	14.67
`kb2.mps`	0.11	0.30	0.46
`sc105.mps`	0.48	10.95	7.23
`sc50a.mps`	0.05	0.64	0.56
`sc50b.mps`	0.06	0.70	0.94
total	64.10	117.88	109.15

5 Development Plans

In this section we briefly review the short- and mid-term development plans we have for the library. We deliberately omit all long-term projects: for all those we mention here, code —whether in the form of a prototype or as a proof-of-concept exercise— has already been developed that proves the feasibility of the proposal.

[20] `http://mathsat.itc.it/`.
[21] `http://sourceforge.net/projects/wallaroo/`.
[22] `http://www.netlib.org/lp/index.html`.

Intervals and Bounding Boxes An important numerical domain is the domain of *bounding boxes*: these are representable by means of finite set of *intervals* or be seen as finite conjunctions of constraints of the form $\pm v_i \leq d$ or $\pm v_i < d$. Despite the fact that bounding boxes have been one of the first abstract domains ever proposed [30] and that they have been implemented and reimplemented dozens of times, no freely available implementation is really suitable for the purposes of abstract interpretation. In fact, the available interval libraries either lack support for non-closed intervals (so that they are unable to represent constraints of the form $\pm v_i < d$), or they do not provide the right support for approximation in the sense of partial correctness (e.g., division by an interval containing zero gives rise to a run-time error instead of giving an interval containing the result under the assumption that the concrete division being approximated was not a division by zero), or they disregard rounding errors and are therefore unsafe. We are thus working at a complete implementation of bounding boxes based on intervals. Such intervals are parametric on a number of features: they support open as well as closed boundaries; boundaries can be chosen within one of the number families mentioned in Section 2.2 (when boundaries are floating point numbers, rounding is of course controlled to maintain soundness); independently from the type of the boundaries, both plain intervals of real numbers and intervals subject to generic *restrictions* are supported. This notion of restriction can be instantiated to obtain intervals of integer numbers, *modulo intervals* [71,72], and generalizations of the latter providing more precise information. [23]

Grid-Polyhedra An interesting line of development consists in the combination of the grids domain with the several polyhedral domains provided by the PPL: not only the $\mathbb{Z}$-polyhedra domain [2], but also many variations such as grid-polyhedra, grid-octagon, grid-bounded-difference, grid-interval domains (not to mention their powersets).

Polynomial Equalities and Inequalities The work in [18] proved the feasibility of representing systems of polynomial inequalities of bounded degree by encoding them into convex polyhedra. The prototype implementation used for the experimental evaluation presented in [18] is being turned into a complete abstract domain and will be incorporated into the PPL.

[23] An implementation of these interval families is already available in the PPL's public CVS repository.

5.2 More Language Interfaces

The current version of the PPL only offers C and Prolog interfaces for (C and NNC) polyhedra and LP problems. It would not be difficult to add, along the same lines, interfaces for all the other abstractions. It can be done, rather quickly, mostly as a "copy and paste" exercise. Instead of following that route (which would imply substantial code duplication and an unaffordable maintenance burden), we are working at an automatic way of obtaining these interfaces out of a few "templates." As part of this ongoing effort, we are extending the set of available *direct* interfaces [24] with Java and Objective Caml. [25] Finally there are plans to develop an interface that allows to use the PPL's numerical abstractions within *Mathematica*. [26]

5.3 Other Features

Other features on the horizon of the Parma Polyhedra Library include the inclusion of: bidirectional serialization functions for all the supported abstractions; the ask-and-tell generic construction of [6]; [27] and the extrapolation operators defined in [58] and [60]; We also plan to add support for more complexity-throttling techniques such as: breaking down the set of the variables of interests into "packs" of manageable size [22,35,84]; support for Cartesian factoring as defined in [55]; and the limitation of the number of constraints and generators and/or the size of coefficients in the representation of polyhedra [43].

6 Discussion

In this paper, we have presented the Parma Polyhedra Library, a library of numerical abstractions especially targeted to applications in the field of analysis and verification of software and hardware systems. We have illustrated the general philosophy that is behind the design of the library, its main features, examples highlighting the advantages offered to client applications, and the avenue we have prepared for future developments.

[24] As opposed to the *indirect* interfaces that can be obtained by passing through the C interface.
[25] Initial versions of these interfaces are already available in the PPL's public CVS repository.
[26] `http://www.wolfram.com/`.
[27] Preliminary support for both these features is already available in the PPL's public CVS repository.

The Parma Polyhedra Library is being used on several applications in the field of verification of hardware and software systems. It has been used for the verification of properties of oscillator circuits [45,46]; to verify the soundness of *batch workflow networks* (a kind of Petri nets used in workflow management) [83]; in the field of safety analysis of continuous and hybrid systems to overapproximate the systems of linear differential equations expressing the dynamics of hybrid automata [40,42,44,79] and, in particular, the PPL is used in PHAVer, an innovative tool for the verification of such systems [43]. The PPL is also used: in a version of *TVLA* (3-Valued Logic Analysis Engine, `http://www.cs.tau.ac.il/~tvla/`), a system for the verification of properties of arrays and heap-allocated data [49]; in *iCSSV* (interprocedural C String Static Verifier), a tool for verifying the safety of string operations in C programs [41]; and in a static analyzer for *gated data dependence graphs*, an intermediate representation for optimizing compilation [61]. This analyzer employs, in particular, the precise widening operator and the widening with tokens technique introduced in [11,12]. In [76] the PPL is used to derive invariant linear equalities and inequalities for a subset of the C language; it is used in *StInG* [77] and *LPInv* [78], two systems for the analysis of transition systems; it is used for the model-checking of reconfigurable hybrid systems [81]; it is used in a static analysis tool for x86 binaries that automatically identifies instructions that can be used to redirect control flow, thus constituting vulnerabilities that can be exploited in order to bypass intrusion detection systems [63]; it is also used to represent and validate real-time systems' constraints and behaviors [39] and to automatically derive the *argument size relations* that are needed for termination analysis of Prolog programs [67].

In conclusion, even though the library is still not mature and functionally complete, it already offers a combination of functionality, reliability, usability and performance that is not matched by similar, freely available libraries. Moreover, since the PPL is free software and distributed under the terms of the GNU General Public License (GPL), and due to the presence of extensive documentation, the library can already be regarded as an important contribution secured to the community.

For the most up-to-date information, documentation and downloads and to follow the development work, the reader is referred to the Parma Polyhedra Library site at `http://www.cs.unipr.it/ppl/`.

Acknowledgments. We would like to express our gratitude and appreciation to all the present and past developers of the Parma Polyhedra Library: Irene Bacchi, Abramo Bagnara, Danilo Bonardi, Sara Bonini, Andrea Cimino, Katy Dobson, Giordano Fracasso, Maximiliano Marchesi, Elena Mazzi, David Merchat, Matthew Mundell, Andrea Pescetti, Barbara Quartieri, Elisa Ricci, Enric Rodríguez-Carbonell, Angela Stazzone, Fabio Trabucchi, Claudio

Trento, Alessandro Zaccagnini, Tatiana Zolo. Thanks also to Aaron Bradley, for contributing to the project his *Mathematica* interface, to Goran Frehse, for contributing his code to limit the complexity of polyhedra, and to all the users of the library that provided us with helpful feedback.

References

[1] J. F. Allen and H. A. Kautz. A model of naive temporal reasoning. In J. R. Hobbs and R. Moore, editors, *Formal Theories of the Commonsense World*, pages 251–268. Ablex, Norwood, NJ, 1985.

[2] C. Ancourt. *Génération automatique de codes de transfert pour multiprocesseurs à mémoires locales*. PhD thesis, Université de Paris VI, Paris, France, March 1991.

[3] D. Avis. Computational experience with the reverse search vertex enumeration algorithm. *Optimization Methods and Software*, 10:107–124, 1998.

[4] D. Avis. lrs: A revised implementation of the reverse search vertex enumeration algorithm. In G. Kalai and G. M. Ziegler, editors, *Polytopes — Combinatorics and Computation*, volume 29 of *Oberwolfach Seminars*, pages 177–198. Birkhäuser-Verlag, 2000.

[5] G. J. Badros, A. Borning, and P. J. Stuckey. The Cassowary linear arithmetic constraint solving algorithm. *ACM Transactions on Computer-Human Interaction*, 8(4):267–306, 2001.

[6] R. Bagnara. *Data-Flow Analysis for Constraint Logic-Based Languages*. PhD thesis, Dipartimento di Informatica, Università di Pisa, Pisa, Italy, March 1997. Printed as Report TD-1/97.

[7] R. Bagnara. A hierarchy of constraint systems for data-flow analysis of constraint logic-based languages. *Science of Computer Programming*, 30(1–2):119–155, 1998.

[8] R. Bagnara, K. Dobson, P. M. Hill, M. Mundell, and E. Zaffanella. Grids: A domain for analyzing the distribution of numerical values, 2006. Accepted for publication and presented at the International Symposium on Logic-based Program Synthesis and Transformation.

[9] R. Bagnara, K. Dobson, P. M. Hill, M. Mundell, and E. Zaffanella. A practical tool for analyzing the distribution of numerical values, 2006. Submitted for publication. Available at `http://www.comp.leeds.ac.uk/hill/Papers/papers.html`.

[10] R. Bagnara, P. M. Hill, E. Mazzi, and E. Zaffanella. Widening operators for weakly-relational numeric abstractions. In C. Hankin and I. Siveroni, editors, *Static Analysis: Proceedings of the 12th International Symposium*, volume 3672 of *Lecture Notes in Computer Science*, pages 3–18, London, UK, 2005. Springer-Verlag, Berlin.

[11] R. Bagnara, P. M. Hill, E. Ricci, and E. Zaffanella. Precise widening operators for convex polyhedra. In R. Cousot, editor, *Static Analysis: Proceedings of the 10th International Symposium*, volume 2694 of *Lecture Notes in Computer Science*, pages 337–354, San Diego, California, USA, 2003. Springer-Verlag, Berlin.

[12] R. Bagnara, P. M. Hill, E. Ricci, and E. Zaffanella. Precise widening operators for convex polyhedra. *Science of Computer Programming*, 58(1–2):28–56, 2005.

[13] R. Bagnara, P. M. Hill, and E. Zaffanella. Not necessarily closed convex polyhedra and the double description method. *Formal Aspects of Computing*, 17(2):222–257, 2005.

[14] R. Bagnara, P. M. Hill, and E. Zaffanella. *The Parma Polyhedra Library Developer's Manual*. Department of Mathematics, University of Parma, Parma, Italy, release 0.9 edition, March 2006. Available at `http://www.cs.unipr.it/ppl/`.

[15] R. Bagnara, P. M. Hill, and E. Zaffanella. *The Parma Polyhedra Library User's Manual*. Department of Mathematics, University of Parma, Parma, Italy, release 0.9 edition, March 2006. Available at `http://www.cs.unipr.it/ppl/`.

[16] R. Bagnara, P. M. Hill, and E. Zaffanella. Widening operators for powerset domains. *Software Tools for Technology Transfer*, 2006. To appear.

[17] R. Bagnara, E. Ricci, E. Zaffanella, and P. M. Hill. Possibly not closed convex polyhedra and the Parma Polyhedra Library. In M. V. Hermenegildo and G. Puebla, editors, *Static Analysis: Proceedings of the 9th International Symposium*, volume 2477 of *Lecture Notes in Computer Science*, pages 213–229, Madrid, Spain, 2002. Springer-Verlag, Berlin.

[18] R. Bagnara, E. Rodríguez-Carbonell, and E. Zaffanella. Generation of basic semi-algebraic invariants using convex polyhedra. In C. Hankin and I. Siveroni, editors, *Static Analysis: Proceedings of the 12th International Symposium*, volume 3672 of *Lecture Notes in Computer Science*, pages 19–34, London, UK, 2005. Springer-Verlag, Berlin.

[19] V. Balasundaram and K. Kennedy. A technique for summarizing data access and its use in parallelism enhancing transformations. In B. Knobe, editor, *Proceedings of the ACM SIGPLAN'89 Conference on Programming Language Design and Implementation (PLDI)*, volume 24(7) of *ACM SIGPLAN Notices*, pages 41–53, Portland, Oregon, USA, 1989. ACM Press.

[20] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[21] B. Bérard and L. Fribourg:. Reachability analysis of (timed) Petri nets using real arithmetic. In J. C. M. Baeten and S. Mauw, editors, *CONCUR'99: Concurrency Theory, Proceedings of the 10th International Conference*, volume 1664 of *Lecture Notes in Computer Science*, pages 178–193, Eindhoven, The Netherlands, 1999. Springer-Verlag, Berlin.

[22] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. A static analyzer for large safety-critical software. In *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation (PLDI'03)*, pages 196–207, San Diego, California, USA, 2003. ACM Press.

[23] M. Bozzano, R. Bruttomesso, A. Cimatti, T. A. Junttila, P. van Rossum, S. Schulz, and R. Sebastiani. The MathSAT 3 system. In R. Nieuwenhuis, editor, *Automated Deduction: Proceedings of the 20th International Conference*, volume 3632 of *Lecture Notes in Computer Science*, pages 315–321, Tallinn, Estonia, 2005. Springer-Verlag, Berlin.

[24] D. Bremner, K. Fukuda, and A. Marzetta. Primal-dual methods for vertex and facet enumeration. *Discrete and Computational Geometry*, 20(3):333–357, 1998.

[25] T. Bultan, R. Gerber, and W. Pugh. Model-checking concurrent systems with unbounded integer variables: Symbolic representations, approximations, and experimental results. *ACM Transactions on Programming Languages and Systems*, 21(4):747–789, 1999.

[26] 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.

[27] 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.

[28] 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.

[29] P. Cousot. Semantic foundations of program analysis. In S. S. Muchnick and N. D. Jones, editors, *Program Flow Analysis: Theory and Applications*, chapter 10, pages 303–342. Prentice Hall, Inc., Englewood Cliffs, New Jersey, 1981.

[30] P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In B. Robinet, editor, *Proceedings of the Second International Symposium on Programming*, pages 106–130, Paris, France, 1976. Dunod, Paris, France.

[31] 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, New York, 1977. ACM Press.

[32] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proceedings of the Sixth Annual ACM Symposium on Principles of Programming Languages*, pages 269–282, New York, 1979. ACM Press.

[33] P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4):511–547, 1992.

[34] 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.

[35] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. The ASTRÉE analyzer. In M. Sagiv, editor, *Programming Languages and Systems, Proceedings of the 14th European Symposium on Programming*, volume 3444 of *Lecture Notes in Computer Science*, pages 21–30, Edinburgh, UK, 2005. Springer-Verlag, Berlin.

[36] 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.

[37] E. Davis. Constraint propagation with interval labels. *Artificial Intelligence*, 32(3):281–331, 1987.

[38] D. L. Dill. Timing assumptions and verification of finite-state concurrent systems. In J. Sifakis, editor, *Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, pages 197–212, Grenoble, France, 1989. Springer-Verlag, Berlin.

[39] D. Doose and Z. Mammeri. Polyhedra-based approach for incremental validation of real-time systems. In L. T. Yang, M. Amamiya, Z. Liu, M. Guo, and F. J. Rammig, editors, *Proceedings of the International Conference on Embedded and Ubiquitous Computing (EUC 2005)*, volume 3824 of *Lecture Notes in Computer Science*, pages 184–193, Nagasaki, Japan, 2005. Springer-Verlag, Berlin.

[40] L. Doyen, T. A. Henzinger, and J.-F. Raskin. Automatic rectangular refinement of affine hybrid systems. Technical Report 2005.47, Centre Fédéré en Vérification, Université Libre de Bruxelles, Belgium, 2005.

[41] R. Ellenbogen. Fully automatic verification of absence of errors via interprocedural integer analysis. Master's thesis, School of Computer Science, Tel-Aviv University, Tel-Aviv, Israel, December 2004.

[42] G. Frehse. Compositional verification of hybrid systems with discrete interaction using simulation relations. In *Proceedings of the IEEE Conference on Computer Aided Control Systems Design (CACSD 2004)*, Taipei, Taiwan, 2004.

[43] G. Frehse. PHAVer: Algorithmic verification of hybrid systems past hytech. In M. Morari and L. Thiele, editors, *Hybrid Systems: Computation and Control:*

Proceedings of the 8th International Workshop (HSCC 2005), volume 3414 of *Lecture Notes in Computer Science*, pages 258–273, Zürich, Switzerland, 2005. Springer-Verlag, Berlin.

[44] G. Frehse, Z. Han, and B. Krogh. Assume-guarantee reasoning for hybrid I/O-automata by over-approximation of continuous interaction. In *Proceedings of the 43rd IEEE Conference on Decision and Control (CDC 2004)*, Atlantis, Paradise Island, Bahamas, 2004.

[45] G. Frehse, B. H. Krogh, and R. A. Rutenbar. Verifying analog oscillator circuits using forward/backward refinement. In *Proceedings of the 9th Conference on Design, Automation and Test in Europe (DATE 06)*, Munich, Germany, 2006. ACM SIGDA. CD-ROM publication.

[46] G. Frehse, B. H. Krogh, R. A. Rutenbar, and O. Maler. Time domain verification of oscillator circuit properties. Available at `http://www-verimag.imag.fr/~maler/Papers/oscil.pdf`, 2005. Presented at the 2005 Workshop on Formal Verification of Analog Circuits (a satellite event of ETAPS 2005).

[47] 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.

[48] D. Gopan, F. DiMaio, N. Dor, T. Reps, and M. Sagiv. Numeric domains with summarized dimensions. In K. Jensen and A. Podelski, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 10th International Conference, TACAS 2004*, volume 2988 of *Lecture Notes in Computer Science*, pages 512–529, Barcelona, Spain, 2004. Springer-Verlag, Berlin.

[49] D. Gopan, T. Reps, and M. Sagiv. A framework for numeric analysis of array operations. In *Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 338–350, Long Beach, California, USA, 2005.

[50] P. Granger. Static analysis of linear congruence equalities among variables of a program. In S. Abramsky and T. S. E. Maibaum, editors, *TAPSOFT'91: Proceedings of the International Joint Conference on Theory and Practice of Software Development, Volume 1: Colloquium on Trees in Algebra and Programming (CAAP'91)*, volume 493 of *Lecture Notes in Computer Science*, pages 169–192, Brighton, UK, 1991. Springer-Verlag, Berlin.

[51] P. Granger. Static analyses of congruence properties on rational numbers (extended abstract). In P. Van Hentenryck, editor, *Static Analysis: Proceedings of the 4th International Symposium*, volume 1302 of *Lecture Notes in Computer Science*, pages 278–292, Paris, France, 1997. Springer-Verlag, Berlin.

[52] N. Halbwachs. *Détermination Automatique de Relations Linéaires Vérifiées par les Variables d'un Programme*. Thèse de 3$^{\text{ème}}$ cycle d'informatique, Université scientifique et médicale de Grenoble, Grenoble, France, March 1979.

[53] 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.

[54] 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.

[55] N. Halbwachs, D. Merchat, and C. Parent-Vigouroux. Cartesian factoring of polyhedra in linear relation analysis. In R. Cousot, editor, *Static Analysis: Proceedings of the 10th International Symposium*, volume 2694 of *Lecture Notes in Computer Science*, pages 355–365, San Diego, California, USA, 2003. Springer-Verlag, Berlin.

[56] 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.

[57] 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.

[58] T. A. Henzinger and P.-H. Ho. A note on abstract interpretation strategies for hybrid automata. In P. J. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, editors, *Hybrid Systems II*, volume 999 of *Lecture Notes in Computer Science*, pages 252–264. Springer-Verlag, Berlin, 1995.

[59] 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.

[60] 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.

[61] C. Hymans and E. Upton. Static analysis of gated data dependence graphs. In R. Giacobazzi, editor, *Static Analysis: Proceedings of the 11th International Symposium*, volume 3148 of *Lecture Notes in Computer Science*, pages 197–211, Verona, Italy, 2004. Springer-Verlag, Berlin.

[62] 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`.

[63] C. Kruegel, E. Kirda, D. Mutz, W. Robertson, and G. Vigna. Automating mimicry attacks using static binary analysis. In *Proceedings of Security '05, the 14th USENIX Security Symposium*, pages 161–176, Baltimore, MD, USA, 2005.

[64] K. Larsen, F. Larsson, P. Pettersson, and W. Yi. Efficient verification of real-time systems: Compact data structure and state-space reduction. In *Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS'97)*, pages 14–24, San Francisco, CA, 1997. IEEE Computer Society Press.

[65] H. Le Verge. A note on Chernikova's algorithm. *Publication interne* 635, IRISA, Campus de Beaulieu, Rennes, France, 1992.

[66] 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 [85].

[67] F. Mesnard and R. Bagnara. cTI: A constraint-based termination inference tool for ISO-Prolog. *Theory and Practice of Logic Programming*, 5(1&2):243–257, 2005.

[68] A. Miné. The octagon abstract domain. In *Proceedings of the Eighth Working Conference on Reverse Engineering (WCRE'01)*, pages 310–319, Stuttgart, Germany, 2001. IEEE Computer Society Press.

[69] A. Miné. *Weakly Relational Numerical Abstract Domains*. PhD thesis, École Polytechnique, Paris, France, March 2005.

[70] 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.

[71] T. Nakanishi and A. Fukuda. Modulo interval arithmetic and its application to program analysis. *Transactions of Information Processing Society of Japan*, 42(4):829–837, 2001.

[72] T. Nakanishi, K. Joe, C. D. Polychronopoulos, and A. Fukuda. The modulo interval: A simple and practical representation for program analysis. In *Proceedings of the 1999 International Conference on Parallel Architectures and Compilation Techniques*, pages 91–96, Newport Beach, California, USA, 1999. IEEE Computer Society.

[73] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, 1988.

[74] P. Quinton, S. Rajopadhye, and T. Risset. On manipulating Z-polyhedra. Technical Report 1016, IRISA, Campus Universitaire de Bealieu, Rennes, France, July 1996.

[75] P. Quinton, S. Rajopadhye, and T. Risset. On manipulating Z-polyhedra using a canonic representation. *Parallel Processing Letters*, 7(2):181–194, 1997.

[76] S. Sankaranarayanan, M. Colón, H. B. Sipma, and Z. Manna. Efficient strongly relational polyhedral analysis. In E. A. Emerson and K. S. Namjoshi, editors, *Verification, Model Checking and Abstract Interpretation: Proceedings of the*

7th International Conference (VMCAI 2006), volume 3855 of *Lecture Notes in Computer Science*, pages 111–125, Charleston, SC, USA, 2006. Springer-Verlag, Berlin.

[77] S. Sankaranarayanan, H. B. Sipma, and Z. Manna. Constraint-based linear-relations analysis. In R. Giacobazzi, editor, *Static Analysis: Proceedings of the 11th International Symposium*, volume 3148 of *Lecture Notes in Computer Science*, pages 53–68, Verona, Italy, 2004. Springer-Verlag, Berlin.

[78] S. Sankaranarayanan, H. B. Sipma, and Z. Manna. Scalable analysis of linear systems using mathematical programming. In R. Cousot, editor, *Verification, Model Checking and Abstract Interpretation: Proceedings of the 6th International Conference (VMCAI 2005)*, volume 3385 of *Lecture Notes in Computer Science*, pages 25–41, Paris, France, 2005. Springer-Verlag, Berlin.

[79] S. Sankaranarayanan, H. B. Sipma, and Z. Manna. Fixed point iteration for computing the time elapse operator. In J. Hespanha and A. Tiwari, editors, *Hybrid Systems: Computation and Control: Proceedings of the 9th International Workshop (HSCC 2006)*, volume 3927 of *Lecture Notes in Computer Science*, pages 537–551, Santa Barbara, CA, USA, 2006. Springer-Verlag, Berlin.

[80] R. Shaham, E. K. Kolodner, and S. Sagiv. Automatic removal of array memory leaks in Java. In D. A. Watt, editor, *Proceedings of the 9th International Conference on Compiler Construction (CC 2000)*, volume 1781 of *Lecture Notes in Computer Science*, pages 50–66, Berlin, Germany, 2000. Springer-Verlag, Berlin.

[81] H. Song, K. Compton, and W. Rounds. SPHIN: a model checker for reconfigurable hybrid systems based on SPIN. In R. Lazic and R. Nagarajan, editors, *Proceedings of the 5rd Workshop on Automated Verification of Critical Systems*, University of Warwick, UK, 2005.

[82] J. Stoer and C. Witzgall. *Convexity and Optimization in Finite Dimensions I.* Springer-Verlag, Berlin, 1970.

[83] K. van Hee, O. Oanea, N. Sidorova, and M. Voorhoeve. Verifying generalized soundness for workflow nets. In I. Virbitskaite and A. Voronkov, editors, *Perspectives of System Informatics: Proceedings of the Sixth International Andrei Ershov Memorial Conference*, Lecture Notes in Computer Science, Akademgorodok, Novosibirsk, Russia, 2006. Springer-Verlag, Berlin. To appear.

[84] A. Venet and G. Brat. Precise and efficient static array bound checking for large embedded C programs. In *Proceedings of the ACM SIGPLAN 2004 Conference on Programming Language Design and Implementation (PLDI'04)*, pages 231–242, Washington, DC, USA, 2004. ACM Press.

[85] 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.