Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 334 (2018) 31–44

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

# On the Efficiency of Convex Polyhedra

Enea Zaffanella[1]

*Department of Mathematical, Physical and Computer Sciences*
*University of Parma, Italy*

**Abstract**

The domain of convex polyhedra plays a special role in the collection of numerical domains considered for program analysis and verification. As far as precision is concerned, it would be the most natural choice in many contexts but, due to its worst case exponential complexity, it is sometimes considered an unaffordable option. This has led to a systematic quest for simpler domains that are capable of reasonable precision using less computational resources. There are anyway cases where the use of the domain of convex polyhedra turns out to be feasible, also due to recent progress in their implementation. After reviewing a few known approaches to decrease the amount of resources needed when computing on this domain, we will introduce a couple of novel techniques that can be used to further improve its efficiency, without incurring precision losses.

*Keywords:* convex polyhedra, domain wrappers, Double Description

## 1 Introduction

Among the collection of the numerical domains that are usually considered by the designers of program analysis and verification tools, the abstract domain of convex polyhedra [12] plays a special role.

As far as precision is concerned, the domain of convex polyhedra would be the most natural choice in many contexts. However, due to its worst case exponential complexity, it is often considered an unaffordable option from a practical point of view. This has led to a systematic quest for domains that are simpler than convex polyhedra, so as to be less demanding in terms of computational resources, and yet capable of reasonable precision. An incomplete list of so-called *weakly relational* abstract domains can be filled by a review of the relevant literature of the past years: bounded differences [26,33], bounded logahedra [21], octagons [27], octahedra [10], parallelotopes [1], pentagons [25], subpolyhedra [23], template polyhedra [32], two variables per inequality [35], weighted hexagons [16].

[1] Email: enea.zaffanella@unipr.it

https://doi.org/10.1016/j.entcs.2018.03.004

| dim | cons | gens | w/o test | with test | ratio |
|---|---|---|---|---|---|
| 13 | 26 | 8192 | 0.12 | 0.03 | 4.0 |
| 14 | 28 | 16384 | 0.45 | 0.05 | 9.0 |
| 15 | 30 | 32768 | 1.66 | 0.12 | 13.8 |
| 16 | 32 | 65536 | 7.00 | 0.26 | 26.9 |

Table 1
Impact of the quick adjacency test of [17] when computing hypercube vertices (time is in seconds).

An alternative approach, used less frequently, is to keep the abstract domain of convex polyhedra but, in order to recover efficiency, replace the most expensive operations with less precise, approximated versions. The usual target for such a replacement is the computation of the *convex polyhedral hull* of two (or more) polyhedra, corresponding to the join (also known as *strong join*) of the abstract domain. During a static analysis, joins model the merging of explicit control flow paths; perhaps less intuitively, they may also be needed when modeling the so-called *weak assignments*, e.g., assignments whose target variable is obtained by pointer dereferencing but for which the analysis is unable to deterministically identify the affected variable. In [31] a few alternatives to the strong join operator are proposed, ranging from the less precise *weak join* (called *envelope* in [6]) to the more precise family of *k-restricted joins*; choosing $k = 2$ results in the definition of the *inversion join*, which has been further studied in [34].

In other contexts, even the precision of convex polyhedra is still not enough for the considered goals, due to the limitations resulting from linearity and convexity. In these cases the designers of formal verification tools may adopt a domain based on *sets* of convex polyhedra [8,15], accepting the corresponding impact on efficiency, or domains that directly deal with non-linear constraints, such as the abstract domain of ellipsoids [7] and the arithmetic-geometric progression domain [13].

There are anyway cases where the use of the domain of convex polyhedra, either considered in isolation or subject to a powerset construction, turns out to be feasible. The reasons are twofold. On the one hand, state-of-the-art implementations of the domain of convex polyhedra are surprisingly effective (taking into account the exponential complexity bound); these include those based on the Double Description (DD) method, such as the Parma Polyhedra Library [4] (PPL) or the Apron library [22], as well as new ones using a constraint-only representation [14], typically based on the Fourier-Motzkin elimination procedure. Such an efficiency not only results from the careful coding of the corresponding core algorithms, but also from the application of several incremental improvements arising from some newer results from the literature. As an example, in the conversion algorithm of the PPL, the integration of the quick (incomplete) adjacency test recalled in [17, Corollary 4.3], complementing the well-known quick (incomplete) non-adjacency test of [24], resulted in a significant reduction of the computation times for some frequently occurring classes of polyhedra (see Table 1).

On the other hand, no matter how good the underlying implementation is, a *blind* adoption of the domain of convex polyhedra in a static analysis or verification tool is likely to result in unacceptable inefficiencies, due to the intrinsic exponential

complexity: special care has to be taken so as to avoid common *efficiency bugs*, i.e., issues that in principle do not directly affect the correctness or the precision of the analysis, but only its efficiency. [2]

The most common reason for an efficiency bug is the attempt to keep track of too many program variables at once, without resorting to well-known techniques such as program slicing or *variable packing*, either statically [7] or dynamically [18,36] computed. [3] Such an efficiency bug can be worsened by the systematic inlining of function calls and/or by analysis tools accepting partially compiled code as input: for instance, if used without precautions, the SSA form tends to artificially increase the number of program variables.

In the following we will consider new algorithms and implementation techniques that can be used to decrease the amount of resources needed when computing on the domain of convex polyhedra. In particular, we will focus on those approaches that have the potential of improving the efficiency of the analysis without incurring precision losses. The techniques presented, which are the result of ongoing collaborations, have all been implemented in the context of the Parma Polyhedra Library. Being work in progress, only preliminary experimental evaluations have been conducted so far. The initial results, besides confirming the absence of precision losses, are encouraging in terms of the efficiency gains that can be obtained.

The paper is structured as follows. Section 2, after introducing some notation and terminology, provides an informal presentation for the DD method. In the next two sections we present some of the work in progress on the domain of convex polyhedra: Section 3 discusses the use of domain wrappers to improve the efficiency in specific contexts; Section 4 describes a new approach for the representation and manipulation of NNC polyhedra. We conclude in Section 5.

## 2 Preliminaries

Assuming the reader is familiar with the notions of static analysis based on Abstract Interpretation [11] and the use of convex polyhedra as an abstract domain [12], we recall some terminology and notation, mostly adapted from [3].

We write $\mathbb{R}^n$ to denote the Euclidean topological space of dimension $n > 0$ and $\mathbb{R}_+$ for the set of non-negative reals; for $S \subseteq \mathbb{R}^n$, $\mathrm{cl}(S)$ and $\mathrm{relint}(S)$ denote the topological closure and the relative interior of $S$, respectively. The scalar product of two vectors $\boldsymbol{a}_1, \boldsymbol{a}_2 \in \mathbb{R}^n$ is denoted by $\boldsymbol{a}_1^{\mathrm{T}}\boldsymbol{a}_2$. For each vector $\boldsymbol{a} \in \mathbb{R}^n$, where $\boldsymbol{a} \neq \boldsymbol{0}$, and scalar $b \in \mathbb{R}$, the linear non-strict inequality constraint $\beta = (\boldsymbol{a}^{\mathrm{T}}\boldsymbol{x} \geq b)$ defines a closed affine half-space of $\mathbb{R}^n$. A topologically closed convex polyhedron (for short, closed polyhedron) is defined as the set of solutions of a finite system $\mathcal{C}$ of linear non-strict inequality constraints.

A vector $\boldsymbol{r} \in \mathbb{R}^n$ such that $\boldsymbol{r} \neq \boldsymbol{0}$ is a *ray* of a non-empty polyhedron $\mathcal{P} \subseteq \mathbb{R}^n$ if,

[2] Efficiency bugs can become precision bugs, e.g., when a timeout causes a switch to a less precise abstract domain or operators.

[3] It may be worth recalling that, for the static analysis described in [7], which was tracking linear relations using octagons rather than convex polyhedra, the analyzed program had 10K global variables, while the average size of variable packs was 4.

for every point $\boldsymbol{p} \in \mathcal{P}$ and every non-negative scalar $\rho \in \mathbb{R}_+$, it holds $\boldsymbol{p} + \rho\boldsymbol{r} \in \mathcal{P}$. The empty polyhedron has no rays. If both $\boldsymbol{r}$ and $-\boldsymbol{r}$ are rays of $\mathcal{P}$, then we say that $\boldsymbol{r}$ is a *line* of $\mathcal{P}$. By Minkowski and Weyl theorems [37], the set $\mathcal{P} \subseteq \mathbb{R}^n$ is a closed polyhedron if and only if there exist finite sets $R, P \subseteq \mathbb{R}^n$ of cardinality $r$ and $p$, respectively, such that $\mathbf{0} \notin R$ and

$$\mathcal{P} = \mathrm{gen}\big(\langle R, P\rangle\big) \stackrel{\text{def}}{=} \left\{ R\boldsymbol{\rho} + P\boldsymbol{\pi} \in \mathbb{R}^n \;\middle|\; \boldsymbol{\rho} \in \mathbb{R}^r_+, \boldsymbol{\pi} \in \mathbb{R}^p_+, \sum_{i=1}^{p} \pi_i = 1 \right\}.$$

When $\mathcal{P} \neq \varnothing$, we say that $\mathcal{P}$ is described by the *generator system* $\mathcal{G} = \langle R, P\rangle$.

The Double Description method due to Motzkin et al. [29], by exploiting the duality principle, allows to combine the constraints and the generators of a polyhedron $\mathcal{P}$ into a DD pair $(\mathcal{C}, \mathcal{G})$: a *conversion* procedure is used to obtain each description starting from the other one, also removing the redundant elements. [4] For presentation purposes, we focus on the conversion from constraints to generators. We also omit the description of important concepts such as polyhedral cones, homogenization, the lattice of faces and adjacency; the interested reader is referred to [37] for the theory and to [9,24] for the details related to implementation.

The conversion procedure starts from a DD pair $(\mathcal{C}_{univ}, \mathcal{G}_{univ})$ representing the whole vector space and adds, one at a time, the elements of the input constraint system $\mathcal{C}_{in}$, keeping the DD pair up-to-date. At each iteration, when adding the constraint $\beta$ to polyhedron $\mathcal{P} = \mathrm{gen}(\mathcal{G})$, the generator system $\mathcal{G}$ is partitioned into the three components $\mathcal{G}^+$, $\mathcal{G}^0$, $\mathcal{G}^-$, based on the sign of the scalar products of the generators with $\beta$ (those in $\mathcal{G}^0$ are the *saturators* of $\beta$); the new generator system is computed as $\mathcal{G}' \stackrel{\text{def}}{=} \mathcal{G}^+ \cup \mathcal{G}^0 \cup \mathcal{G}^\star$, where

$$\mathcal{G}^\star \stackrel{\text{def}}{=} \big\{ \mathrm{combine}_\beta(g^+, g^-) \;\big|\; g^+ \in \mathcal{G}^+, g^- \in \mathcal{G}^-, \mathrm{adjacent}_{\mathcal{P}}(g^+, g^-) \big\};$$

function ‘$\mathrm{combine}_\beta$’ computes a linear combination of its arguments, yielding a generator that saturates the constraint $\beta$; predicate ‘$\mathrm{adjacent}_{\mathcal{P}}$’ is used to discard those pairs of generators that are not *adjacent* in $\mathcal{P}$ (since these would only produce redundant generators).

If linear *strict* inequality constraints ($\boldsymbol{a}^{\mathrm{T}}\boldsymbol{x} > b$) are allowed, a not necessarily closed (NNC) polyhedron is obtained. The generator system is extended to become a triple $\mathcal{G} = \langle R, P, C\rangle$ where those in $C$ are *closure points* [3] for the polyhedron $\mathcal{P}$, i.e., they belong to its topological closure $\mathrm{cl}(\mathcal{P})$. Namely,

$$\mathcal{P} = \mathrm{gen}\big(\langle R, P, C\rangle\big) \stackrel{\text{def}}{=} \left\{ R\boldsymbol{\rho} + P\boldsymbol{\pi} + C\boldsymbol{\gamma} \in \mathbb{R}^n \;\middle|\; \begin{array}{l} \boldsymbol{\rho} \in \mathbb{R}^r_+, \boldsymbol{\pi} \in \mathbb{R}^p_+, \boldsymbol{\pi} \neq \mathbf{0}, \boldsymbol{\gamma} \in \mathbb{R}^c_+, \\ \sum_{i=1}^{p} \pi_i + \sum_{i=1}^{c} \gamma_i = 1 \end{array} \right\}.$$

Implementations of the DD method for closed polyhedra can be extended to also work on NNC polyhedra by adding a *slack* variable [3,19]: this satisfies the

[4] Implementations are based on a stronger minimality concept, taking into special account equality constraints and lines.

non-strict constraints $0 \leq \epsilon \leq 1$, but it is *interpreted* to allow only for strictly positive values. A strict inequality constraint is encoded as $(\boldsymbol{a}^{\mathrm{T}}\boldsymbol{x} - e\epsilon \geq b)$ and a point $\boldsymbol{p} \in \mathbb{R}^n$ is encoded as $(\boldsymbol{p}^{\mathrm{T}}, e)^{\mathrm{T}} \in \mathbb{R}^{n+1}$, where in both cases $e > 0$; all the other constraints/generators (in particular, the closure points) have a 0 coefficient for $\epsilon$. Hence, an NNC polyhedron $\mathcal{P} \subseteq \mathbb{R}^n$ is represented as a closed polyhedron $\mathcal{R} \in \mathbb{R}^{n+1}$, which is said to be an $\epsilon$-representation [3] for $\mathcal{P}$. Its semantic interpretation is

$$\mathcal{P} = [\![\mathcal{R}]\!] \stackrel{\text{def}}{=} \left\{ \boldsymbol{p} \in \mathbb{R}^n \;\middle|\; \exists e \in \mathbb{R} \,.\, \left(e > 0 \wedge (\boldsymbol{p}^{\mathrm{T}}, e)^{\mathrm{T}} \in \mathcal{R}\right) \right\}.$$

When completed with the addition of suitable strong minimization procedures [3], this technique allows to reuse, almost unchanged, all the algorithms and techniques developed for topologically closed polyhedra.

## 3 Wrappers for Convex Polyhedra

Designers of static analysis and verification tools try hard to implement their software so as to be parametric with respect to the abstract domain chosen. To this end, they often provide *wrappers* (i.e., thin layers of software) to interface their tool with the implementation of the abstract domain. In this section we similarly propose a few abstract domain wrappers, but with a different goal: these are meant to provide a workaround to a few commonly occurring efficiency bugs. For clarity, we will sometimes say "shell" to refer to the wrapper object (for a convex polyhedron), while calling "kernel" the wrapped one (i.e., the convex polyhedron itself).

As a running example, we consider PAGAI [20], a static analyzer for invariant generation built on top of the LLVM infrastructure and the Apron library. PAGAI analyzes the bitcode generated by LLVM, possibly combining classical Abstract Interpretation and SMT-solving techniques; we consider the "simple" static analysis, so as to factor out the computational cost of SMT decision procedures.

When using PAGAI with convex polyhedra, it is possible to choose between the native Apron domain (`pk`) or the Apron layer for the PPL domain (`ppl_poly`). In both cases, the domain has to support strict inequality constraints (hence, it is the domain of NNC polyhedra): these are used to model strict conditional tests involving floating point variables.[5] To experimentally evaluate the wrappers for NNC polyhedra presented below, PAGAI and Apron have been modified to make available a new `ppl_wpoly` abstract domain. Some information on (a selection of) the tests used for the experimental evaluation is reported in Table 2, which also summarizes data on the joins and abstract assignment operators computed during a run of the analyzer (these are independent from the specific convex polyhedra domain tested).

[5] If all the variables mentioned in the conditional test are integral, it is automatically tightened into a more precise non-strict test.

| | | | join | | | assign | | |
|---|---|---|---|---|---|---|---|---|
| test | sloc | `.bc` size | calls | avg dim | max dim | calls | unconstr | alias |
| decompress | 495 | 72312 | 9636 | 21.9 | 62 | 22357 | 22044 | 20395 |
| filter | 417 | 14968 | 32910 | 18.2 | 34 | 14949 | 12446 | 8412 |
| adpcm | 468 | 69028 | 17803 | 22.3 | 42 | 3697 | 3530 | 2474 |
| cover | 228 | 33588 | 15997 | 2.2 | 5 | 2746 | 2707 | 2707 |
| fft1 | 136 | 20256 | 6277 | 8.4 | 24 | 2012 | 1812 | 788 |

Table 2
Test programs for PAGAI: data on abstract operations for the convex polyhedra domains.

### *3.1 On Demand NNC polyhedra*

Since in many programs conditional tests on floating point variables are very unlikely to occur (some programs have no floating point variable at all), the systematic use of NNC polyhedra, even where a domain of topologically closed (C) polyhedra would suffice, can be regarded as an efficiency bug: in critical contexts, NNC polyhedra can be significantly slower than C polyhedra (see the first three columns in Table 5).

The first wrapper we propose (denoted $\mathbb{D}$) is meant to *dynamically* switch between the domains of C and NNC polyhedra; in order to avoid the NNC overhead as much as possible, the switch from C to NNC is done only when needed, while the switch from NNC to C is done whenever possible (i.e., as soon as detecting that the polyhedron is topologically closed).

The implementation of this wrapper is quite simple:[6] the kernel object is either a C or an NNC polyhedron; the shell object only needs to perform the switches, delegating everything else to the kernel. Switches are rarely needed, since almost all of the operations on the convex polyhedra domain preserve topological closure: intersection, convex polyhedral hull, affine images and preimages, addition and removal of space dimensions, widening, addition of non-strict constraints, plus all of the read only operators; the one exception is the addition of strict inequality constraints (plus a few other operators, such as the convex polyhedral difference or the addition of closure points, which however are not used in PAGAI).

### *3.2 A Wrapper for Unconstrained Variables*

The approximation of assignments is one of the most frequently executed abstract operations. A closer look at the assignments computed during the runs of the static analyzer revealed that the variable assigned is very likely to be unconstrained (compare the 7th and 8th columns in Table 2; this is probably due to the fact that the analyzed bitcode is the result of the SSA transformation). In implementations of polyhedra based on the DD method, each unconstrained space dimension is represented as a *line* in the generator system. In all the constraints and the other generators the unconstrained dimension occurs with a zero coefficient (explicitly represented in most implementations based on the DD method).

The idea underlying our second wrapper (denoted $\mathbb{U}$) is to keep track of the set of

[6] The coding and testing of the `DNNC_Polyhedron` class were completed in a single day in October 2015.

*unconstrained* space dimensions, so that the kernel only knows about the constrained variables. In practice, we record a partial, injective map from the set $\{0, \ldots, n-1\}$ of the shell dimensions to the set $\{0, \ldots, m-1\}$ of the kernel dimensions, where $m \leq n$. The unmapped shell dimensions are known to be unconstrained and when a dimension becomes constrained (resp., unconstrained), it is added to (resp., removed from) the kernel polyhedron, keeping the map up-to-date.

The knowledge of the set of unconstrained dimensions can also be exploited to optimize a few abstract operators. In particular, when computing the convex polyhedral hull, all the unconstrained dimensions of the arguments will be unconstrained in the result too: hence, the kernel polyhedron of each argument can be preprocessed to remove those (constrained) dimensions that are unconstrained in the other argument, yielding a reduction of the number of dimensions when actually computing the join on the kernels.

Another possible optimization can be applied to the approximation of *invertible* affine assignments, i.e., those where the variable assigned also occurs in the right hand side affine expression (e.g., `x = x + 1`). If the variable is known to be unconstrained, then the affine map modeling the assignment is the identity function. However, the Apron interface layer to the PPL domains seems to prevent the use of domain operators computing affine images: an assignment such as `x = x + 1` is implemented by (a) adding the primed variable $x'$, (b) adding the constraint $x' = x + 1$, (c) projecting away the unprimed variable $x$ and finally (d) renaming $x'$ to $x$. Also, invertible assignments rarely occur when analyzing programs using PAGAI (we conjecture this is another side effect of the SSA transformation).

### *3.3 A Wrapper for Aliased Variables*

The last column in Table 2 shows that a good percentage of all the assignments computed are of the form `x = y`, i.e., the right hand side expression is a single variable (once again, this is likely due to the SSA form).

Following the same line of reasoning used above for the $\mathbb{U}$ wrapper, we propose a third wrapper (denoted $\mathbb{A}$) that keeps track of the set of *aliased* space dimensions. In practice, the wrapper keeps a partition of the shell dimensions and only one dimension for each partition block is made known to the kernel polyhedron. In this case, maintaining the partition information up-to-date (in particular, discovering new alias relations after modifying the kernel polyhedron) may require non-trivial computational work: the implementation may range between a fully eager approach, where all aliasing is extracted as soon as possible, and several lazy alternatives, with quite different effects on the efficiency of the wrapper.

Note that this wrapper could be generalized to another one ($\mathbb{E}$) keeping track of arbitrary affine equations (or some intermediate forms of equations, such as the octagonal ones). On its surface, the idea seems similar to the symbolic constant propagation domain proposed in [7,28]; a closer look shows several differences. The main one regards their motivations: the domain in [28] was proposed as a way to enhance the *precision* of numerical domains (in particular, the non-relational or weakly relational ones); in contrast, in our context we only care about *efficiency*, as

| | | | | | `ppl_wpoly` (composed) wrappers | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| test | box | oct | pk | ppl | $\mathbb{D}$ | $\mathbb{U}$ | $\mathbb{A}$ | $\mathbb{UD}$ | $\mathbb{AU}$ | $\mathbb{AUD}$ |
| decompress | 8.32 | 17.84 | 20.96 | 15.59 | 14.57 | 10.34 | 17.65 | 9.76 | 9.53 | 9.31 |
| filter | 2.76 | 7.79 | 82.47 | 89.72 | 49.45 | 43.23 | 64.26 | 21.35 | 36.69 | 18.75 |
| adpcm | 1.76 | 4.29 | 22.27 | 15.30 | 11.73 | 5.21 | 17.88 | 3.84 | 5.31 | 4.01 |
| cover | 1.38 | 1.44 | 2.83 | 3.56 | 3.01 | 2.68 | 3.58 | 2.45 | 2.69 | 2.53 |
| fft1 | 0.45 | 0.87 | 2.39 | 2.18 | 1.96 | 1.25 | 2.30 | 1.18 | 1.33 | 1.25 |

Table 3
Testing wrappers for convex polyhedra in PAGAI (box = intervals; oct = octagons; pk = Apron's NNC; ppl = PPL's NNC; $\mathbb{D}/\mathbb{U}/\mathbb{A}$= wrappers on PPL's NNC; time is in seconds).

there should be no precision gain/loss. This different point of view becomes evident, for instance, when modeling the join operator: in the symbolic domain of [28] the join is typically very imprecise; in our wrapper, those equations that can not be preserved at the shell level are communicated down to the kernel level, thereby preserving the precision of the join.

It is also worth noting that, apart from $\mathbb{D}$, the wrappers proposed could be applied to other numerical domains, such as the weakly relational ones. For instance, the alias domain $\mathbb{A}$ (or even its generalization $\mathbb{E}$) could use the domain of parallelotopes [1] as its kernel.

### *3.4 Experimental Evaluation*

In Table 3 we summarize the results of the experimental evaluation on the tests of Table 2.[7] After the name of the test, in the first three columns we report, for reference, the times obtained when using the Apron's native domains (boxes, octagons and polyhedra). The 4th column reports the times obtained by the (unwrapped) `ppl_poly` domain. The next three columns report the times for `ppl_wpoly`, using the wrappers $\mathbb{D}$, $\mathbb{U}$ and $\mathbb{A}$ discussed above. The wrappers $\mathbb{U}$ and $\mathbb{A}$ have been implemented in `C++` as class templates and some care has been explicitly taken to make them composable. Hence, in the last three columns of Table 3, we also report on the use of the composed wrappers $\mathbb{UD}$, $\mathbb{AU}$ and $\mathbb{AUD}$; a name such as $\mathbb{UD}$ can be read as $\mathbb{U} \circ \mathbb{D}$, i.e., the $\mathbb{U}$ wrapper having $\mathbb{D}$ as the kernel.

Each time reported is the *total* time for running the `pagai` executable on the bitcode for the test program, including the input and output phases: the time actually spent in the abstract domains is just a portion of it. As an example, when using the `ppl_poly` domain on the test `decompress`, 50% of the total time is spent outside of the code giving access to the abstract domain operations; another 10% is spent in the Apron layer for the PPL domain; the remaining 40% (6.15 secs) is spent in the PPL code itself. This should be taken into proper account when reasoning on the speedups obtained by using the wrappers. Also note that the composition of the wrappers may incur some avoidable overhead.

According to the times shown in Table 3, the wrappers $\mathbb{D}$ and $\mathbb{U}$ provide signifi-

[7] All tests have been performed on a laptop with an Intel Core i7-3632QM CPU, 16 GB of RAM and running GNU/Linux.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| $\mathcal{G}^+$ | R | R | R | C | C | C | P | P | P |
| $\mathcal{G}^-$ | R | C | P | R | C | P | R | C | P |
| $\geq$ | R | C | P | C | C | P | P | P | P |
| $>$ | R | C | C+P | C | C | C+P | C | C | C |

| | 10 | 11 | 12 |
|---|---|---|---|
| $\mathcal{G}^+$ | R | C | P |
| $\mathcal{G}^0$ | P | P | P |
| $\geq$ | | | |
| $>$ | C+P | C+P | C |

Table 4
Case analyses for the linear combination of generators (R = ray, C = closure point, P = point) when adding non-strict ($\geq$) and strict ($>$) inequality constraints.

cant efficiency improvements; moreover, further efficiency gains are obtained when using their composition $\mathbb{UD}$, somehow confirming that the two wrappers are actually solving orthogonal efficiency bugs.

The alias wrapper $\mathbb{A}$ only improves efficiency on `filter`, whereas on the other tests the improvement (if any) is hidden by the corresponding computational overhead. The results are marginally better when using $\mathbb{AU}$ and $\mathbb{AUD}$, in particular for the tests requiring more time, making the full composition $\mathbb{AUD}$ the fastest of all considered domains when analysing `decompress` and `filter`.

## 4 An Improved Representation for NNC Polyhedra

As recalled in Section 2, implementations of NNC polyhedra based on the DD method require the addition of a supplementary space dimension (the $\epsilon$-dimension). While being completely adequate from the theoretical point of view and behaving reasonably well from a practical one, this approach has a couple of drawbacks.

First, the use of the $\epsilon$-dimension may result in a significant increase in the number of generators used in the low-level representation of NNC polyhedra, for instance doubling the number of vertices in the case of a polytope that happens to be topologically closed. Second, operations on NNC polyhedra tend to generate a very high number of $\epsilon$-redundant constraints and/or generators; the strong minimization procedures defined in [3], while effective, are not incremental: their use in the middle of a sequence of operations on NNC polyhedra may trigger a complete recomputation of the DD pair. In principle, incremental strong minimization procedures can be obtained by exploiting the work in [2], where an algorithm for the *removal* of constraints/generators from a DD pair is defined. However, the algorithm turns out to be effective mainly when removing a small number of elements from a DD pair, which is seldom the case in the considered context.

Both issues above simply disappear if a new, more direct encoding for NNC polyhedra, with no additional $\epsilon$-dimension, is used. A first attempt along this line of reasoning was performed in [30], where a variant of Chernikova's conversion algorithm was defined for the NNC case. The new algorithm works like the classical one, incrementally adding one (strict or non-strict) inequality constraint at each iteration and partitioning the generators into the sets $\mathcal{G}^+$, $\mathcal{G}^0$, $\mathcal{G}^-$. However, when linearly combining the generators, it performs a systematic case analysis on the generator kind, as described in Table 4, so as to identify the kind of generator resulting from each combination.

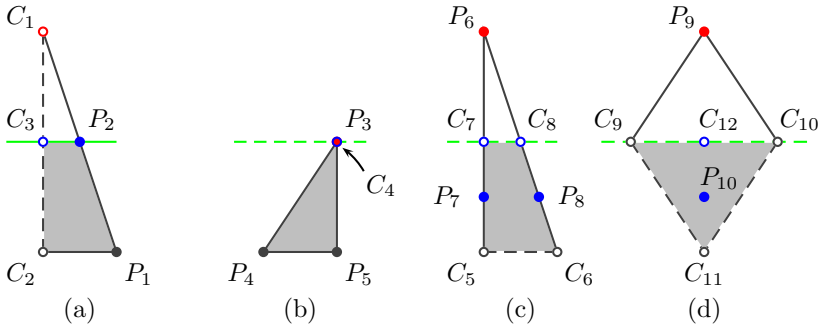


Fig. 1. Examples of linear combinations for generators: (unfilled) circles represent (closure) points; (dashed) lines represent (strict) inequality constraints.

Consider for example polyhedron (a) in Figure 1, defined by closure points $C_1$, $C_2$ and point $P_1$. The addition of the non-strict inequality (in green in the figure) makes $C_1$ an element of $\mathcal{G}^-$, while $C_2$ and $P_1$ are in $\mathcal{G}^+$; therefore, $C_2$ is combined with $C_1$ yielding closure point $C_3$ (see column 5 in Table 4); similarly, $P_1$ is combined with $C_1$ yielding point $P_2$ (column 8); hence, we obtain the polyhedron having points $P_1$, $P_2$ and closure points $C_2$, $C_3$.

Consider now polyhedron (b) in Figure 1, where a strict inequality constraint is being added. In this case, the component $\mathcal{G}^-$ is empty; however, the right hand side portion of Table 4 indicates that, when processing a strict inequality, the generators in $\mathcal{G}^+$ also need to be combined with the points in $\mathcal{G}^0$: in our example, when combining $P_4$ or $P_5$ with $P_3$, the latter is *transformed* into the closure point $C_4$ (column 12), since a point can not saturate a strict inequality.

The polyhedron (c) in Figure 1 shows a special case occurring when processing a strict constraint: the closure point $C_5$ in $\mathcal{G}^+$ has to be combined with $P_6$ in $\mathcal{G}^-$, yielding closure point $C_7$ (column 6); however, in order to include the open segment $(C_5, C_7)$ in the resulting polyhedron, we also need to generate a point on the open segment itself ($P_7$ in our example). The entries in Table 4 for this special case are thus marked as 'C+P'. The same reasoning applies when combining $C_6$ and $P_6$, yielding $C_8$ and $P_8$.

The polyhedron (d) in Figure 1 is a minor variation of (c) that is meant to highlight the main limitation of the proposed algorithm. When adding the strict inequality constraint, closure point $C_{11}$ in $\mathcal{G}^+$ is combined with point $P_9$ in $\mathcal{G}^-$ according to column 6 of Table 4, obtaining closure point $C_{12}$ (which is later detected as redundant and removed) and point $P_{10}$ (which is not redundant). This is going to compute the correct result, but the attentive reader may have noted that the generators $C_{11}$ and $P_9$ are *not* adjacent in the input polyhedron. Thus, in order to be correct, we can not exploit the adjacency tests when combining the generators using Table 4, potentially incurring into high inefficiencies.

### 4.1 The Adjacency Problem: Towards a Solution

In order to recover from the efficiency loss we have pointed out, we have to further refine the case analysis of Table 4 so as to identify those cases (or subcases) where the optimizations based on the adjacency tests can be restored without compromising the correctness of the algorithm. A decisive step towards such a goal is being made in [5]. For space reasons, we are not going to provide a full description of the new algorithm, which is work in progress; we will informally describe the main idea underlying it and show the initial experimental results obtained.

The key observation in [5] is that the generators of an NNC polyhedron $\mathcal{P}$ can be split into two components: the first one, named the *skeleton* [8] of $\mathcal{P}$, is made by those generators that "contribute" to the topological closure of $\mathcal{P}$. More precisely, for a non-redundant generator system $\mathcal{G} = \langle R, P, C \rangle$ describing $\mathcal{P}$, we define its set of *skeleton points* $SP \subseteq P$ as those points that are not redundant in $\mathrm{cl}(\mathcal{P})$; then, the skeleton of $\mathcal{G}$ is defined as $\mathrm{skel}\big(\langle R, P, C \rangle\big) \stackrel{\text{def}}{=} \langle R, SP, C \rangle$.

The *non-skeleton* component only contains the points in $P \setminus SP$. Examples of non-skeleton points are $P_7$ and $P_8$ in polyhedron (c) and $P_{10}$ in polyhedron (d) of Figure 1. When representing a non-skeleton point $\boldsymbol{p}$, its precise geometric position is an overkill: $\boldsymbol{p}$ can be abstracted by representing instead the *minimal face* $\mathcal{F}_{\boldsymbol{p}}$ such that $\boldsymbol{p} \in \mathrm{relint}(\mathcal{F}_{\boldsymbol{p}})$. The face $\mathcal{F}_{\boldsymbol{p}} \subseteq \mathcal{P}$ is itself an NNC polyhedron, so that we can compute its skeleton $\mathrm{skel}(\mathcal{F}_{\boldsymbol{p}})$, which is component-wise included in $\mathrm{skel}(\mathcal{G})$; by the minimality assumption above, $\mathrm{skel}(\mathcal{F}_{\boldsymbol{p}})$ contains only rays and closure points; hence, $\mathcal{F}_{\boldsymbol{p}}$ is described by its skeleton plus *any* single non-skeleton point taken from $\mathrm{relint}(\mathcal{F}_{\boldsymbol{p}})$. We say that $\mathrm{skel}(\mathcal{F}_{\boldsymbol{p}})$ is the *support* for the non-skeleton point $\boldsymbol{p}$. In our examples from Figure 1, the support for $P_7$ is $\langle \varnothing, \varnothing, \{C_5, C_7\} \rangle$ and the support for $P_{10}$ is $\langle \varnothing, \varnothing, \{C_9, C_{10}, C_{11}\} \rangle$ (where $C_{12}$ is omitted as it is redundant).

To summarize, the skeleton component of an NNC polyhedron is represented geometrically, while the non-skeleton component is only provided with a *combinatorial* representation. The new conversion algorithm in [5], by keeping a clear distinction of the two components, is able to optimize several computation steps:

- scalar products and saturation matrices are only computed for the skeleton;
- Table 4 is simplified by considering only the skeleton generators, since the non-skeleton points are subject to *ad hoc* treatment;
- the adjacency tests can be restored when combining the skeleton generators to produce other skeleton generators;
- the removal of redundancies from the non-skeleton component is made easier by the separate combinatorial representation.

A preliminary experimental evaluation, passing all of the regression tests of the PPL, shows important efficiency improvements on many benchmarks. Besides the expected gains on the NNC tests, we also observed a few unexpected efficiency gains on the topologically closed tests, i.e., when comparing the new algorithm with the

[8] This term is unrelated to the concept of $p$-skeleton used in algebraic topology.

| | C | $\epsilon$-based NNC | | new NNC | |
|---|---|---|---|---|---|
| test | time | time | ratio | time | ratio |
| `sampleh8` | 6.95 | 26.91 | 3.87 | 4.89 | 0.70 |
| `trunc10` | 1.64 | 6.40 | 3.90 | 1.02 | 0.62 |
| `mit31-20` | 1.35 | 4.72 | 3.50 | 0.77 | 0.57 |
| `kkd38_6` | 0.66 | 2.49 | 3.77 | 0.28 | 0.42 |

Table 5
Measuring the overhead of the conversion procedure for NNC polyhedra; the topologically closed polyhedra used as tests are part of the `ppl_lcdd` test suite (time is in seconds).

`C_Polyhedron` implementation of the PPL (see Table 5).

The new algorithm has been specified, implemented and tested for the conversion from the constraint to the generator representation; the undergoing generalization to the conversion from generators to constraints is based on the usual duality results.

## 5 Conclusions

Despite the intrinsic limitations implied by its theoretical complexity bounds, the domain of convex polyhedra turns out to be a feasible option in several contexts [36]. This is due not only to the recent progress on their implementation, but also to the understanding of the common efficiency bugs that a static analysis tool using this domain should definitely avoid. In this paper we have proposed a few domain wrappers that are able to provide an automatic workaround to some of these efficiency bugs. We have also sketched a new idea for the representation and manipulation of NNC polyhedra that seems to have a great potential in terms of efficiency with respect to the currently available implementations.

## Acknowledgment

Most of the new material presented in this paper is the result of joint work still in progress. The author would like to acknowledge Gianluca Amato and Francesca Scozzari for the contents presented in Sections 3.2 and 3.3; Anna Becchi and Simone Perri for the contents of Section 4.

## References

[1] Amato, G. and F. Scozzari, *The abstract domain of parallelotopes*, Electr. Notes Theor. Comput. Sci. **287** (2012), pp. 17–28.

[2] Amato, G., F. Scozzari and E. Zaffanella, *Efficient constraint/generator removal from double description of polyhedra*, Electr. Notes Theor. Comput. Sci. **307** (2014), pp. 3–15.

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

[4] Bagnara, R., P. M. Hill and E. Zaffanella, *The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems*, Science of Computer Programming **72** (2008), pp. 3–21.

[5] Becchi, A., "Poliedri NNC: una nuova rappresentazione e algoritmo di conversione (*NNC Polyhedra: a New Representation and Conversion Algorithm*)," Undergraduate thesis, Department of Mathematical, Physical and Computer Sciences, University of Parma, Italy (2017), in preparation, in Italian.

[6] Bemporad, A., K. Fukuda and F. D. Torrisi, *Convexity recognition of the union of polyhedra*, Computational Geometry: Theory and Applications **18** (2001), pp. 141–154.

[7] Blanchet, B., 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)* (2003), pp. 196–207.

[8] Bultan, T., 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** (1999), pp. 747–789.

[9] Chernikova, N. V., *Algorithm for discovering the set of all solutions of a linear programming problem*, U.S.S.R. Computational Mathematics and Mathematical Physics **8** (1968), pp. 282–293.

[10] Clarisó, R. and J. Cortadella, *The octahedron abstract domain*, Sci. Comput. Program. **64** (2007), pp. 115–139.

[11] Cousot, P. and R. Cousot, *Static determination of dynamic properties of programs*, in: *Proceedings of the Second International Symposium on Programming* (1976), pp. 106–130.

[12] Cousot, P. 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* (1978), pp. 84–96.

[13] Feret, J., *The arithmetic-geometric progression abstract domain*, in: *Verification, Model Checking, and Abstract Interpretation, 6th International Conference, VMCAI 2005, Paris, France, Proceedings*, 2005, pp. 42–58.

[14] Fouilhé, A., "Revisiting the abstract domain of polyhedra : constraints-only representation and formal proof," Ph.D. thesis, Grenoble Alpes University, France (2015).

[15] Frehse, G., *PHAVer: Algorithmic verification of hybrid systems past HyTech*, in: *Hybrid Systems: Computation and Control: Proceedings of the 8th International Workshop (HSCC 2005)*, Lecture Notes in Computer Science **3414** (2005), pp. 258–273.

[16] Fulara, J., K. Durnoga, K. Jakubczyk and A. Schubert, *Relational abstract domain of weighted hexagons*, Electr. Notes Theor. Comput. Sci. **267** (2010), pp. 59–72.

[17] Genov, B., "The Convex Hull Problem in Practice: Improving the Running Time of the Double Description Method," Ph.D. thesis, University of Bremen, Germany (2014).

[18] Halbwachs, N., D. Merchat and C. Parent-Vigouroux, *Cartesian factoring of polyhedra in linear relation analysis*, in: *Static Analysis: Proceedings of the 10th International Symposium*, Lecture Notes in Computer Science **2694** (2003), pp. 355–365.

[19] Halbwachs, N., Y.-E. Proy and P. Raymond, *Verification of linear hybrid systems by means of convex approximations*, in: *Static Analysis: Proceedings of the 1st International Symposium*, Lecture Notes in Computer Science **864** (1994), pp. 223–237.

[20] Henry, J., D. Monniaux and M. Moy, *PAGAI: A path sensitive static analyser*, Electr. Notes Theor. Comput. Sci. **289** (2012), pp. 15–25.

[21] Howe, J. M. and A. King, *Logahedra: A new weakly relational domain*, in: *Automated Technology for Verification and Analysis, 7th International Symposium, ATVA 2009, Macao, China. Proceedings*, 2009, pp. 306–320.

[22] Jeannet, B. and A. Miné, *Apron: A library of numerical abstract domains for static analysis*, in: *Computer Aided Verification, Proceedings of the 21st International Conference (CAV 2009)*, Lecture Notes in Computer Science **5643** (2009), pp. 661–667.

[23] Laviron, V. and F. Logozzo, *Subpolyhedra: A (more) scalable approach to infer linear inequalities*, in: *Verification, Model Checking, and Abstract Interpretation: Proceedings of the 10th International Conference (VMCAI 2009)*, Lecture Notes in Computer Science **5403** (2009), pp. 229–244.

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

[25] Logozzo, F. and M. Fähndrich, *Pentagons: A weakly relational abstract domain for the efficient validation of array accesses*, in: *Proceedings of the 2008 ACM Symposium on Applied Computing (SAC 2008)* (2008), pp. 184–188.

[26] Miné, A., *A new numerical abstract domain based on difference-bound matrices*, in: *Proceedings of the 2nd Symposium on Programs as Data Objects (PADO 2001)*, Lecture Notes in Computer Science **2053** (2001), pp. 155–172.

[27] Miné, A., *The octagon abstract domain*, in: *Proceedings of the Eighth Working Conference on Reverse Engineering (WCRE'01)* (2001), pp. 310–319.

[28] Miné, A., "Weakly Relational Numerical Abstract Domains," Ph.D. thesis, École Polytechnique, Paris, France (2005).

[29] Motzkin, T. S., H. Raiffa, G. L. Thompson and R. M. Thrall, *The double description method*, in: *Contributions to the Theory of Games – Volume II*, number 28 in Annals of Mathematics Studies, Princeton University Press, Princeton, New Jersey, 1953 pp. 51–73.

[30] Perri, S., "Un algoritmo stile Chernikova per poliedri NNC (*A Chernikova-style Algorithm for NNC Polyhedra*)," Undergraduate thesis, Department of Mathematics and Computer Science, University of Parma, Italy (2012), in Italian.

[31] Sankaranarayanan, S., M. Colón, H. B. Sipma and Z. Manna, *Efficient strongly relational polyhedral analysis*, in: *Verification, Model Checking and Abstract Interpretation: Proceedings of the 7th International Conference (VMCAI 2006)*, Lecture Notes in Computer Science **3855** (2006), pp. 111–125.

[32] Sankaranarayanan, S., H. B. Sipma and Z. Manna, *Scalable analysis of linear systems using mathematical programming*, in: *Verification, Model Checking and Abstract Interpretation: Proceedings of the 6th International Conference (VMCAI 2005)*, Lecture Notes in Computer Science **3385** (2005), pp. 25–41.

[33] Shaham, R., E. K. Kolodner and S. Sagiv, *Heap profiling for space-efficient java*, in: *Proceedings of the 2001 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), Snowbird, Utah, USA*, 2001, pp. 104–113.

[34] Simon, A., *A note on the inversion join for polyhedral analysis*, Electr. Notes Theor. Comput. Sci. **267** (2010), pp. 115–126.

[35] Simon, A., A. King and J. M. Howe, *Two variables per linear inequality as an abstract domain*, in: *Logic Based Program Synthesis and Transformation, 12th International Workshop*, Lecture Notes in Computer Science **2664** (2002), pp. 71–89.

[36] Singh, G., M. Püschel and M. T. Vechev, *Fast polyhedra abstract domain*, in: *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France*, 2017, pp. 46–59.

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