PPLite: Zero-overhead encoding of NNC polyhedra

PPLite: Zero-Overhead Encoding of NNC Polyhedra

Anna Becchi[a,*], Enea Zaffanella[b,*]

[a] *Fondazione Bruno Kessler, Via Sommarive 18, Povo, 38123 Trento, Italy*
[b] *Università di Parma, Parco Area delle Scienze 53/a (Campus), 43124 Parma, Italy*

Abstract

We present an alternative Double Description representation for the domain of NNC (not necessarily closed) polyhedra, together with the corresponding Chernikova-like conversion procedure. The representation uses no slack variable at all and provides a solution to a few technical issues caused by the encoding of an NNC polyhedron as a closed polyhedron in a higher dimension space. We then reconstruct the abstract domain of NNC polyhedra, providing all the operators needed to interface it with commonly available static analysis tools: while doing this, we highlight the efficiency gains enabled by the new representation and we show how the canonicity of the new representation allows for the specification of proper, semantic widening operators. A thorough experimental evaluation shows that our new abstract domain achieves significant efficiency improvements with respect to classical implementations for NNC polyhedra.

Keywords: convex polyhedra, double description, strict inequalities, static analysis

1. Introduction

The Double Description (DD) method [47] allows for the representation and manipulation of convex polyhedra by using two different geometric representations: one based on a finite collection of *constraints*, the other based on a finite collection of *generators*. Starting from any one of these representations, the other can be derived by application of a conversion procedure [21, 22, 23], thereby obtaining a DD pair; the procedure allows for the identification and removal of redundant elements from both representations, yielding a DD pair in minimal form; moreover, it is incremental, allowing for capitalizing on the work already done when new constraints and/or generators need to be added to an input DD pair.

The DD method lies at the foundation of several software libraries and tools. The following is an incomplete list of available implementations:

*Corresponding author
Email addresses: abecchi@fbk.eu (Anna Becchi), enea.zaffanella@unipr.it (Enea Zaffanella)

- cdd (`www.inf.ethz.ch/personal/fukudak/cdd_home`);
- PolyLib (`http://icps.u-strasbg.fr/PolyLib`);
- NewPolka, part of Apron (`http://apron.cri.ensmp.fr/library`);
- Parma Polyhedra Library (`http://bugseng.com/products/ppl`);
- 4ti2 (`www.4ti2.de`);
- Skeleton (`www.uic.unn.ru/~zny/skeleton`);
- Addibit (`www.informatik.uni-bremen.de/agbs/bgenov/addibit`).

Despite the intrinsic exponential complexity of the conversion procedure, these implementations turn out to be surprisingly effective in many contexts. As a consequence, the range of applicability of the DD method keeps widening, also due to several incremental improvements in the efficiency of the most critical processing phases [36, 37, 45, 56]. Quoting from [36]:

> The double description method is a simple and useful algorithm [...] despite the fact that we can hardly state any interesting theorems on its time and space complexities.

Implementations of the DD method are actively used, either directly or indirectly, in several research fields, with applications as diverse as bioinformatics [53, 54], computational geometry [1, 3], analysis of analog and hybrid systems [19, 34, 40, 41], automatic parallelization [15, 50], scheduling [31], static analysis of software [12, 24, 30, 32, 38, 42].

In the classical setting, the DD method is meant to compute geometric representations for *topologically closed* polyhedra in an n-dimensional vector space: these polyhedra can be represented by finite constraint systems where all of the constraints are linear *non-strict* inequalities. There are applications that, at least in principle, also require the ability to deal with linear *strict* inequality constraints, leading to the definition of *not necessarily closed* (NNC) polyhedra. For example, this is the case for some of the analysis tools developed for the verification of hybrid systems [19, 34, 40, 41]; other examples of the use of NNC polyhedra include static analysis tools such as PAGAI [42], where strict inequalities are used to model the semantics of conditional tests acting on program variables of floating point type; the direct use of strict inequalities has also been proposed in [24] for the automatic discovery of ranking functions, so as to prove the termination of program fragments.

Any strict inequality constraint $\beta \equiv (\boldsymbol{a}^{\mathrm{T}}\boldsymbol{x} > b)$ can be transformed into a non-strict inequality constraint $\beta' \equiv (\boldsymbol{a}^{\mathrm{T}}\boldsymbol{x} - \epsilon \geq b)$ by adding a fresh *slack* variable ϵ satisfying the strict positivity constraint $\epsilon > 0$. Hence, from a theoretical point of view, strict inequalities are not really needed: the construction outlined in Section 3 will make clear that any application computing on the domain of NNC polyhedra can be equivalently based on a library computing on the domain of topologically closed polyhedra (in a higher dimension vector

space). The few DD method implementations providing support for NNC polyhedra are all based on such an *indirect* representation of the strict inequalities, implicitly using this supplementary space dimension. The advantage of such an approach is the possibility of reusing, almost unchanged, all of the well-studied algorithms and optimizations that have been developed for the classical case of closed polyhedra [9, 14, 40, 41]. The obvious disadvantage is that the addition of the slack variable carries with itself an overhead, as well as a few more technical issues.

In this paper, we pursue a different approach for the handling of NNC polyhedra in the DD method. Namely, we specify a *direct* representation, dispensing with the need of the slack variable and hence enabling corresponding gains in terms of *efficiency*. The main insight of this new approach is the separation of the (constraints or generators) geometric representation into two components, the skeleton and the non-skeleton of the representation, playing quite different roles: while keeping a geometric encoding for the skeleton component, we will adopt a combinatorial encoding for the non-skeleton one. For this new representation, we propose the corresponding variant of the Chernikova's conversion procedure, where both components are handled by respective processing phases, so as to take advantage of their peculiarities. In particular, we develop *ad hoc* functions and procedures for the combinatorial non-skeleton part.

The new representation and conversion procedure are then used to specify all of the operators that are needed for the definition of a classical static analysis based on Abstract Interpretation [30], leading to the implementation of PPLite, a new library for convex polyhedra derived from the PPL (Parma Polyhedra Library, [11, 12]). While describing the operators, we stress on the efficiency gains that are triggered by the new representation, contrasting it with the more classical approach based on the use of a slack variable. In particular, we propose a more appropriate, semantics-based specification for the widening operator on NNC polyhedra and then we further improve its precision by applying the framework of [5, 6].

We conclude by showing an extensive experimental evaluation where the PPLite library is integrated in a static analysis tool, so as to compare its performance with more classical implementations. This experimental evaluation, besides assessing the new implementation in terms of correctness and precision, provides evidence for the significant speedups that are obtained by the new representation on a wide range of benchmarks.

In applications such as static analysis, where the computed polyhedra are often over-approximated to ensure the efficiency and the termination of the abstract computation, the abstract domain of NNC polyhedra could be replaced from the very beginning by the abstract domain of topologically closed polyhedra, without incurring a significant precision penalty: in practice, each linear strict inequality $\beta \equiv (\boldsymbol{a}^{\mathrm{T}}\boldsymbol{x} > b)$ is *relaxed* to become a non-strict inequality $\beta' \equiv (\boldsymbol{a}^{\mathrm{T}}\boldsymbol{x} \geq b)$, without adding a slack variable. This is indeed a common approach, mainly motivated by the fear that the traditional implementations of the domain of NNC polyhedra will be affected by the corresponding overhead. Thanks to our new representation, this worry is no longer justified: our experi-

mental evaluation confirms that the adoption of the domain of NNC polyhedra incurs no intrinsic penalty. It should also be noted that there are contexts where the analysis and verification tools have to perform *exact* computations, with no over-approximation allowed:[1] for instance, this happens when considering piecewise constant hybrid systems and performing reachability analysis in order to *disprove* a candidate safety property [18, 35]. In this setting, any relaxation of a strict inequality constraint into the corresponding non-strict version could compromise the overall correctness of the result, so that the analysis tool needs to compute on the domain of NNC polyhedra. Notice that strict inequality constraints could be present in the initial specification of the analyzed system; as an example, the NAV_m_n hybrid systems used as benchmarks in the 2020 edition of the ARCH-COMP verification competition[2] are characterized by discrete transitions having the strict constraint $\beta \equiv (time > 0)$ in the guard component, so as to make sure that any two consecutive discrete transitions are separated by some strictly positive amount of time. Strict inequalities can also arise during the analysis, e.g., when the semantic construction needs to compute the exact, possibly disjunctive complement of a topologically closed polyhedron.

The paper is structured as follows. Section 2, after introducing the required notation and terminology, briefly describes the Double Description method for the representation of closed polyhedra, also sketching the Chernikova's conversion algorithm. Section 3 summarizes the encoding of NNC polyhedra into closed polyhedra based on the addition of a slack variable, highlighting a few technical issues. Section 4, after recalling a few properties of the face lattices of closed and NNC polyhedra, proposes the new representation for NNC polyhedra, which uses no slack variable and distinguishes between a geometric and a combinatorial component. Section 5 is devoted to the extension of the Chernikova's conversion algorithm to the case of NNC polyhedra adopting this new representation. Section 6 shows how, using the new representation and conversion algorithm, we can implement all the abstract domain operators needed for the development of a static analysis based on Abstract Interpretation. Section 7, after providing some details on the development of the PPLite library, summarizes the results of a thorough experimental evaluation where the newly implemented domain is compared with existing ones in order to asses its correctness, precision and efficiency. We conclude in Section 8. The proofs for the stated results can be found in Appendix A.

This paper is a combined, extended and improved version of [16], where we were originally proposing the new representation for NNC polyhedra and presenting the first part of the conversion algorithm, and [17], where we completed the specification of the conversion algorithm and we explained how the

[1] Exact computations can be obtained when considering specific subclasses of systems; also, the exactness requirement often implies that the analysis procedure may fail to terminate.

[2] See `https://cps-vo.org/group/ARCH/FriendlyCompetition`; these benchmarks, derived from the well-known *navigation benchmarks* [33], are part of the PCDB category (Piecewise Constant Dynamics and Bounded model checking); they are meant to test the capabilities of bounded model checking tools.

operators needed for Abstract Interpretation can be implemented. We propose here a more complete and uniform presentation, adopting consistent terminology and notation and providing, in Appendix A, the proofs of all the stated formal results. We also enriched the experimental evaluation to demonstrate that the new representation is orthogonal to (i.e., can benefit from) generic constructions aiming at improving the efficiency of the abstract domain, such as Cartesian factoring. The specification of the improved widening operator in Section 6.2 and the corresponding experimental evaluation in Section 7 are also new to this paper.

2. Preliminaries

We assume some familiarity with the basic notions of lattice theory [20].

For a lattice $\langle L, \sqsubseteq, \bot, \top, \sqcap, \sqcup \rangle$, an element $a \in L$ is an *atom* if $\bot \sqsubset a$ and there exists no element $b \in L$ such that $\bot \sqsubset b \sqsubset a$. The lattice L is said to be *atomistic* if every element of L can be obtained as the join of a set of atoms. For $S \subseteq L$, the *upward closure* of S is defined as $\uparrow S \doteq \{\, x \in L \mid \exists s \in S \,.\, s \sqsubseteq x \,\}$. The set $S \subseteq L$ is *upward closed* if $S = \uparrow S$; we denote by $\wp_{\uparrow}(L)$ the set of all the upward closed subsets of L. For $x \in L$, $\uparrow x$ is a shorthand for $\uparrow\{x\}$. The notation for *downward closure* is similar.

Given two posets $\langle L, \sqsubseteq \rangle$ and $\langle L^\sharp, \sqsubseteq^\sharp \rangle$ and two monotonic functions $\alpha\colon L \to L^\sharp$ and $\gamma\colon L^\sharp \to L$, the pair (α, γ) is a *Galois connection* [28] (between L and $L^\sharp$) if

$$\forall x \in L, x^\sharp \in L^\sharp : \alpha(x) \sqsubseteq^\sharp x^\sharp \Leftrightarrow x \sqsubseteq \gamma(x^\sharp).$$

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 $\beta \equiv (\boldsymbol{a}^{\mathrm{T}} \boldsymbol{x} \geq b)$ defines a closed affine half-space of $\mathbb{R}^n$; similarly, the linear equality constraint $\beta \equiv (\boldsymbol{a}^{\mathrm{T}} \boldsymbol{x} = b)$ defines an affine hyperplane of $\mathbb{R}^n$.

2.1. Topologically closed convex polyhedra

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 and linear equality constraints; namely, $\mathcal{P} = \mathrm{con}(\mathcal{C})$ where

$$\mathrm{con}(\mathcal{C}) \doteq \left\{\, \boldsymbol{p} \in \mathbb{R}^n \;\middle|\; \forall \beta \equiv (\boldsymbol{a}^{\mathrm{T}} \boldsymbol{x} \bowtie b) \in \mathcal{C}, \bowtie \in \{=, \geq\} \,.\, \boldsymbol{a}^{\mathrm{T}} \boldsymbol{p} \bowtie b \,\right\}.$$

The set $\mathbb{CP}_n$ of all closed polyhedra on the vector space $\mathbb{R}^n$, partially ordered by set inclusion, is a lattice $\langle\, \mathbb{CP}_n, \subseteq, \emptyset, \mathbb{R}^n, \cap, \uplus \,\rangle$, where the empty set and $\mathbb{R}^n$ are the bottom and top elements, the binary meet operator is set intersection and the binary join operator '$\uplus$' is the convex polyhedral hull.

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, for every point $\boldsymbol{p} \in \mathcal{P}$ and every non-negative scalar $\rho \in \mathbb{R}_+$, it holds

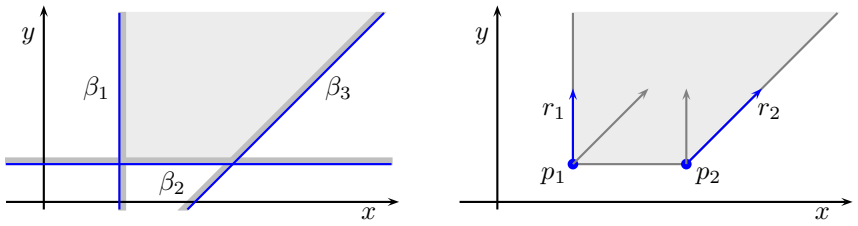

Figure 1: Constraint and generator representations of a closed polyhedron.

$\boldsymbol{p} + \rho\boldsymbol{r} \in \mathcal{P}$. 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 [52], the set $\mathcal{P} \subseteq \mathbb{R}^n$ is a closed polyhedron if and only if there exist finite sets $L, R, P \subseteq \mathbb{R}^n$ of cardinality ℓ, r and p, respectively, such that $\mathbf{0} \notin (L \cup R)$ and $\mathcal{P} = \mathrm{gen}\big(\langle L, R, P\rangle\big)$, where

$$\mathrm{gen}\big(\langle L, R, P\rangle\big) \doteq \big\{\, L\boldsymbol{\lambda} + R\boldsymbol{\rho} + P\boldsymbol{\pi} \in \mathbb{R}^n \;\big|\; \boldsymbol{\lambda} \in \mathbb{R}^\ell, \boldsymbol{\rho} \in \mathbb{R}^r_+, \boldsymbol{\pi} \in \mathbb{R}^p_+, \textstyle\sum_{i=1}^p \pi_i = 1 \,\big\}.$$

When $\mathcal{P} \neq \emptyset$, we say that $\mathcal{P}$ is described by the *generator system* $\mathcal{G} = \langle L, R, P\rangle$. In the following, we will abuse notation by adopting the usual set operator and relation symbols to denote the corresponding component-wise extensions on generator systems. For instance, for $\mathcal{G} = \langle L, R, P\rangle$ and $\mathcal{G}' = \langle L', R', P'\rangle$, we will write $\mathcal{G} \subseteq \mathcal{G}'$ to mean $L \subseteq L'$, $R \subseteq R'$ and $P \subseteq P'$; similarly, we may write $\wp(\mathcal{G})$ to denote the set of all generator systems $\mathcal{G}'$ such that $\mathcal{G}' \subseteq \mathcal{G}$.

Example 1. In Figure 1 we show the constraint and generator representations of a polyhedron $\mathcal{P} \subseteq \mathbb{R}^2$. On the left-hand side of the figure the polyhedron is represented by constraint system $\mathcal{C} = \{\beta_1, \beta_2, \beta_3\}$, so that $\mathcal{P} = \mathrm{con}(\mathcal{C})$ is obtained by intersecting the three closed affine half-spaces defined by constraints $\beta_1 \equiv (x \geq 2)$, $\beta_2 \equiv (y \geq 1)$ and $\beta_3 \equiv (y - x \geq -4)$. On the right-hand side of the figure, the same polyhedron is represented by generator system $\mathcal{G} = \langle L, R, P\rangle$, where $L = \emptyset$, $R = \{r_1, r_2\}$, $P = \{p_1, p_2\}$, $r_1 = (0, 1)$, $r_2 = (1, 1)$, $p_1 = (2, 1)$ and $p_2 = (5, 1)$; hence each point in $\mathcal{P} = \mathrm{gen}(\mathcal{G})$ is obtained by adding a convex combination of the points in P and a non-negative combination of the rays in R. Note that, for intuition and readability, in the figure we "apply" scaled versions of rays r_1 and r_2 to points p_1 and p_2. Also note that polyhedron $\mathcal{P}$ is unbounded, but topologically closed.

2.2. The conversion procedure for closed polyhedra

A Double Description pair is formed by a constraint system $\mathcal{C}$ and a generator system $\mathcal{G}$ describing the same polyhedron: we will write $\mathcal{P} \equiv (\mathcal{C}, \mathcal{G})$ meaning that $\mathcal{P} = \mathrm{con}(\mathcal{C}) = \mathrm{gen}(\mathcal{G})$. The Double Description method due to Motzkin et al. [47], 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

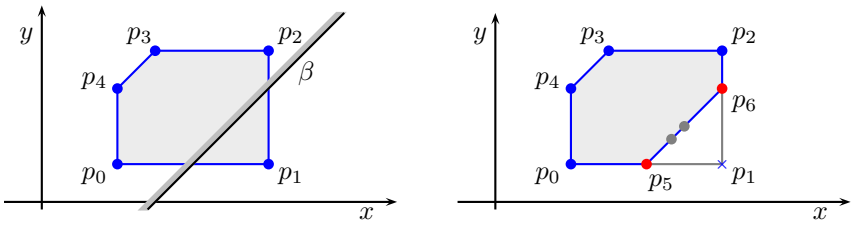

Figure 2: Incremental addition of a constraint.

redundant elements. For presentation purposes, we focus on the conversion from constraints to generators; the conversion from generators to constraints works in the same way, using duality to switch the roles of constraints and generators.

The conversion procedure starts from a DD pair $(\mathcal{C}_0, \mathcal{G}_0)$ representing the whole vector space and adds, one at a time, the elements of the input constraint system $\mathcal{C} = \{\beta_0, \ldots, \beta_m\}$, producing a sequence of DD pairs $\{(\mathcal{C}_k, \mathcal{G}_k)\}_{0 \leq k \leq m+1}$ representing the polyhedra

$$\mathbb{R}^n = \mathcal{P}_0 \xrightarrow{\beta_0} \ldots \xrightarrow{\beta_{k-1}} \mathcal{P}_k \xrightarrow{\beta_k} \mathcal{P}_{k+1} \xrightarrow{\beta_{k+1}} \ldots \xrightarrow{\beta_m} \mathcal{P}_{m+1} = \mathcal{P}.$$

At each iteration, when adding the constraint β_k to polyhedron $\mathcal{P}_k = \mathrm{gen}(\mathcal{G}_k)$, the generator system $\mathcal{G}_k$ is partitioned into the three components $\mathcal{G}_k^+$, $\mathcal{G}_k^0$, $\mathcal{G}_k^-$, according to the sign of the scalar products of the generators with β_k (those in $\mathcal{G}_k^0$ are the *saturators* of β_k); the new generator system for polyhedron $\mathcal{P}_{k+1}$ is computed as $\mathcal{G}_{k+1} \doteq \mathcal{G}_k^+ \cup \mathcal{G}_k^0 \cup \mathcal{G}_k^\star$, where

$$\begin{aligned}\mathcal{G}_k^\star &= \mathrm{comb\_adj}_{\beta_k}(\mathcal{G}_k^+, \mathcal{G}_k^-)\\ &\doteq \left\{ \mathrm{comb}_{\beta_k}(g^+, g^-) \;\middle|\; g^+ \in \mathcal{G}_k^+, g^- \in \mathcal{G}_k^-, \mathrm{adj}_{\mathcal{P}_k}(g^+, g^-) \right\}.\end{aligned}$$

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

The conversion procedure is usually followed by a *simplification* step, where the DD pair is modified, without affecting the represented polyhedron, so as to achieve some form of minimality. In the following, we say that $\mathcal{C}$ (resp., $\mathcal{G}$) is in *minimal form* if it contains a maximal set of equalities (resp., lines) and no redundancies: constraint $\beta \in \mathcal{C}$ is *redundant* in $\mathcal{C}$ if $\mathrm{con}(\mathcal{C}) = \mathrm{con}(\mathcal{C} \setminus \{\beta\})$; similarly for generators. We will not provide a formalization of these details, assuming anyway that these simplifications are implicitly taken into proper account when needed.

Example 2. In Figure 2 we show an example of the incremental addition of a constraint to a polyhedron, focusing on the effect of the conversion procedure on the generator representation. The polyhedron $\mathcal{P} \subseteq \mathbb{R}^2$ on the left-hand side of the figure is described by generator system $\mathcal{G} = \langle L, R, P \rangle$, where $L = R = \emptyset$

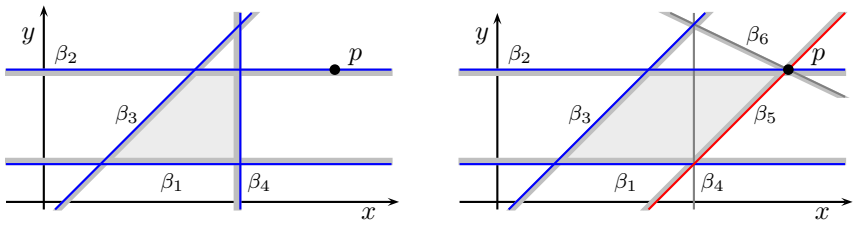

Figure 3: Incremental addition of a point generator.

and $P = \{p_0, p_1, p_2, p_3, p_4\}$. Having no lines or rays, $\mathcal{P}$ is a polytope; hence, to simplify exposition and with no risk of ambiguity, in the following we will systematically confuse the generator system $\mathcal{G}$ with its point component P. When adding the new constraint β, the generator system is first partitioned in $\mathcal{G}^+ = \{p_0, p_2, p_3, p_4\}$, $\mathcal{G}^0 = \emptyset$ and $\mathcal{G}^- = \{p_1\}$. Then, we compute

$$\mathcal{G}^\star = \text{comb\_adj}_\beta(\mathcal{G}^+, \mathcal{G}^-) = \{p_5, p_6\},$$

where $p_5 = \text{comb}_\beta(p_0, p_1)$ and $p_6 = \text{comb}_\beta(p_2, p_1)$ are shown in red on the right-hand side of the figure. Note that there is no need to linearly combine p_3 with p_1 or p_4 with p_1, because these pairs of generators are not adjacent: their combination would result in redundant points (shown in gray on the right-hand side of the figure), which can be obtained as convex combinations of p_5 and p_6. The new generator system is $\mathcal{G}' = \mathcal{G}^+ \cup \mathcal{G}^0 \cup \mathcal{G}^\star = \{p_0, p_2, p_3, p_4, p_5, p_6\}$.

The previous can be seen as the computation of the set intersection of $\mathcal{P}$ with the polyhedron $\text{con}(\{\beta\})$ (the closed half-space): as a matter of fact, the implementation of the meet $\mathcal{P}_1 \cap \mathcal{P}_2$ on the lattice $\mathbb{CP}_n$ is obtained by incrementally adding to the DD representation of $\mathcal{P}_1$ the constraints defining $\mathcal{P}_2$; the implementation of the join $\mathcal{P}_1 \uplus \mathcal{P}_2$ (the convex polyhedral hull) is obtained similarly, exploiting duality, by incrementally adding to the DD representation of $\mathcal{P}_1$ the generators defining $\mathcal{P}_2$.

Example 3. In Figure 3 we show an example of incremental addition of a point to a polyhedron, focusing on the effect of the conversion procedure on the constraint representation. The polyhedron $\mathcal{P} \subseteq \mathbb{R}^2$ on the left-hand side of the figure is obtained by intersecting the four closed affine half-spaces induced by the constraint in $\mathcal{C} = \{\beta_1, \beta_2, \beta_3, \beta_4\}$. When adding a new point p, the constraint system $\mathcal{C}$ is partitioned into $\mathcal{C}^+ = \{\beta_1, \beta_3\}$, $\mathcal{C}^0 = \{\beta_2\}$ and $\mathcal{C}^- = \{\beta_4\}$ according to the sign of the scalar product of each constraint with p. A new set of constraints $\mathcal{C}^\star$ is obtained by combining pairs of adjacent constraints taken from $\mathcal{C}^+$ and $\mathcal{C}^-$: namely, we obtain constraint $\beta_5 = \text{comb}_p(\beta_1, \beta_4)$, shown in red on the right-hand side of the figure. Note that there is no need to linearly combine the non-adjacent pair β_3 and β_4, since they would produce the redundant constraint $\beta_6 = \text{comb}_p(\beta_3, \beta_4)$ (shown in gray on the right-hand side of the figure). Therefore, the new constraint system is $\mathcal{C} = \{\beta_1, \beta_2, \beta_3, \beta_5\}$.

As hinted by the previous examples, a key ingredient of any efficient implementation of the conversion procedure is the test for adjacency. The adjacency relation can be efficiently computed by keeping track of saturation information. For a constraint β and a generator system $\mathcal{G}$, we write $\mathrm{sat}(\beta, \mathcal{G})$ to denote the generator system composed by those elements of $\mathcal{G}$ *saturating* β (i.e., satisfying the corresponding equality constraint). For a constraint system $\mathcal{C}$, we define $\mathrm{sat}(\mathcal{C}, \mathcal{G}) = \bigcap \{ \mathrm{sat}(\beta, \mathcal{G}) \mid \beta \in \mathcal{C} \}$. We define $\mathrm{sat}(g, \mathcal{C})$ and $\mathrm{sat}(\mathcal{G}, \mathcal{C})$ similarly.

It is also worth noting that the one sketched above is just a high level description of a Chernikova-like conversion procedure. More technical observations are required at the mere implementation level, where each closed polyhedron $\mathcal{P} \subseteq \mathbb{R}^n$ is mapped, by *homogenization*, into a (topologically closed) convex polyhedral cone $\mathbb{C} \subseteq \mathbb{R}^{n+1}$. This process associates a new space dimension, usually denoted as ξ, to the inhomogeneous term of constraints; the new space dimension is constrained to only assume non-negative values, i.e., the *positivity constraint* $\xi \geq 0$ is added to the constraint representation of the polyhedral cone. When reinterpreted in the n dimensional vector space, this constraint can be read as the tautology $1 \geq 0$. The inverse map from a convex polyhedral cone $\mathbb{C}$ to the represented convex polyhedron $\mathcal{P}$ is obtained by only considering the points of the cone having a strictly positive coordinate for the ξ dimension:

$$\mathcal{P} = [\![\mathbb{C}]\!]_\xi \doteq \{ \boldsymbol{x} \in \mathbb{R}^n \mid (\boldsymbol{x}^\mathrm{T}, \xi)^\mathrm{T} \in \mathbb{C}, \xi > 0 \}.$$

2.3. Not necessarily closed convex polyhedra

A linear *strict* inequality constraint $\beta \equiv (\boldsymbol{a}^\mathrm{T}\boldsymbol{x} > 0)$ defines an open affine half-space of $\mathbb{R}^n$. When the constraint system $\mathcal{C}$ is extended to also allow for strict inequalities, the convex polyhedron $\mathcal{P} = \mathrm{con}(\mathcal{C})$, where

$$\mathrm{con}(\mathcal{C}) \doteq \{ \boldsymbol{p} \in \mathbb{R}^n \mid \forall \beta \equiv (\boldsymbol{a}^\mathrm{T}\boldsymbol{x} \bowtie b) \in \mathcal{C}, \bowtie \in \{=, \geq, >\} \,.\, \boldsymbol{a}^\mathrm{T}\boldsymbol{p} \bowtie b \},$$

is not necessarily (topologically) closed. The set $\mathbb{P}_n$ of all NNC polyhedra on the vector space $\mathbb{R}^n$ is a lattice $\langle \mathbb{P}_n, \subseteq, \emptyset, \mathbb{R}^n, \cap, \uplus \rangle$ and $\mathbb{CP}_n$ is a sublattice of $\mathbb{P}_n$.

As shown in [9, 14], a description of an NNC polyhedron $\mathcal{P} \in \mathbb{P}_n$ in terms of generators can be obtained by also taking into account its *closure points*, i.e., points that belong to the topological closure of the polyhedron, but are not necessarily included in the polyhedron itself. Namely, the results by Minkowski and Weyl can be generalized to the case of NNC polyhedra [9, Theorem 4.4]: we can extend the generator system with a finite set C of closure points, obtaining $\mathcal{G} = \langle L, R, C, P \rangle$ and $\mathcal{P} = \mathrm{gen}(\mathcal{G})$, where

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

When needed for notational convenience, we will split a constraint system into three components $\mathcal{C} = \langle C^=, C^\geq, C^> \rangle$; even in this case, as done for the generators, we will abuse the notation for set operator and relation symbols.

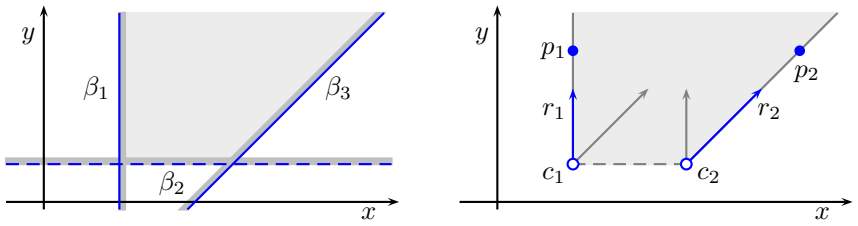

Figure 4: Constraint and generator representations of an NNC polyhedron.

Example 4. In Figure 4 we show the constraint and generator representations of an NNC polyhedron $\mathcal{P} \subseteq \mathbb{R}^2$ which is not topologically closed. On the left-hand side of the figure we show the constraint representation, which is a simple variant of the constraints used in Example 1, where the second constraint $\beta_2 \equiv (y > 1)$ is a strict inequality defining an open affine half-space (shown by a dashed line boundary; solid boundaries represent non-strict inequalities as before). On the right-hand side of the figure, the same polyhedron is represented by generator system $\mathcal{G} = \langle L, R, C, P \rangle$, where $L = \emptyset$, $R = \{r_1, r_2\}$, $C = \{c_1, c_2\}$, $P = \{p_1, p_2\}$, $r_1 = (0, 1)$, $r_2 = (1, 1)$, $c_1 = (2, 1)$, $c_2 = (5, 1)$, $p_1 = (2, 4)$ and $p_2 = (8, 4)$. As before, for intuition and readability, in the figure we "apply" scaled versions of rays r_1 and r_2 to closure points c_1 and c_2. Note that closure points are represented as unfilled circles, whereas filled circles represent proper points as before. It is worth observing that, due to the presence of point $p_1 \in P$ (resp., $p_2 \in P$), all of the points lying on the boundary of β_1 (resp., β_3) which are above the closure point $c_1 \in C$ (resp., $c_2 \in C$) are included in the polyhedron $\mathcal{P} = \mathrm{gen}(\mathcal{G})$; in contrast, none of the points of the segment $[c_1, c_2]$ are included in $\mathcal{P}$: the only way to obtain these points from the generators in $\mathcal{G}$ would be to force $\boldsymbol{\pi} = \mathbf{0}$ in the specification of function 'gen', which is not allowed. Hence, we obtain $\mathcal{P} = \mathrm{con}(\mathcal{C}) = \mathrm{gen}(\mathcal{G})$.

3. Representing NNC Polyhedra as Closed Polyhedra

The DD method provides a solid theoretical base for the representation and manipulation of topologically closed convex polyhedra in $\mathbb{CP}_n$. As mentioned in Section 2, at the implementation level the polyhedra are actually mapped into polyhedral cones in $\mathbb{CP}_{n+1}$ by homogenization, but it is not difficult for software libraries to make this detail completely invisible to the end user: in practice, the library developers have to add some syntactic sugar to the input and output routines for constraints and generators, also hiding the positivity constraint.

Things are less straightforward when considering the case of NNC polyhedra. To start with, many implementations of the DD method do not support NNC polyhedra at all. Also, the few supported implementations of the domain of NNC polyhedra based on the DD method (that is, the NewPolka domain embedded in the Apron library and the `NNC_Polyhedron` domain in the Parma

Polyhedra Library) adopt an *indirect* representation: namely, each NNC polyhedron $\mathcal{P} \in \mathbb{P}_n$ is mapped into a closed polyhedron $\mathcal{R} \in \mathbb{CP}_{n+1}$. The mapping encodes the distinction between strict and non-strict inequality constraints (resp, the distinction between points and closure points) by means of an additional space dimension, playing the role of a *slack variable* and usually denoted as ϵ; the new space dimension is forced to be non-negative and bounded from above,[3] i.e., the constraints $0 \leq \epsilon \leq 1$ are added to the topologically closed representation $\mathcal{R}$ (called ϵ-representation) of the NNC polyhedron $\mathcal{P}$. When translating the NNC constraint system, each strict inequality of $\mathcal{P}$ is encoded as a non-strict inequality of $\mathcal{R}$ having a negative coefficient for ϵ; the non-strict inequalities are instead encoded using a zero coefficient. When translating the NNC generator system, each line, ray and closure point of $\mathcal{P}$ is translated as a line, ray and point of $\mathcal{R}$ having a zero coordinate for ϵ; each point of $\mathcal{P}$ is instead translated into a pair of points of $\mathcal{R}$, one having a zero and one having a positive coordinate for ϵ.

The inverse map $[\![\cdot]\!]_\epsilon \colon \mathbb{CP}_{n+1} \to \mathbb{P}_n$ from an ϵ-representation $\mathcal{R}$ to the represented NNC polyhedron $\mathcal{P}$ is obtained by only considering the points of $\mathcal{R}$ having a strictly positive coordinate for the ϵ dimension:

$$\mathcal{P} = [\![\mathcal{R}]\!]_\epsilon \doteq \big\{\, \boldsymbol{x} \in \mathbb{R}^n \;\big|\; (\boldsymbol{x}^{\mathrm{T}}, \epsilon)^{\mathrm{T}} \in \mathcal{R}, \epsilon > 0 \,\big\}.$$

This encoding of NNC polyhedra into closed polyhedra was initially proposed in [40, 41] and later reconsidered and studied in more detail in [9, 14], where a proper interpretation of the ϵ dimension for the (extended) generator representation was provided.

Example 5. Consider the NNC polyhedron $\mathcal{P} = \mathrm{con}(\{x \geq 1, -x > -3\}) \in \mathbb{P}_1$, i.e, the half-open segment $[1, 3) \subseteq \mathbb{R}$; the same polyhedron can be described by a generator system $\mathcal{G} = \langle L, R, C, P \rangle$, where $L = R = \emptyset$, $C = \{c\}$, $P = \{p\}$, for closure point $c = (3)$ and point $p = (1)$. On the left-hand side of Figure 5 we show an ϵ-representation $\mathcal{R}_1 \in \mathbb{CP}_2$ for $\mathcal{P}$, obtained by translating the NNC constraint system. In $\mathcal{R}_1$, the constraint $\beta_1 \equiv (x + 0 \cdot \epsilon \geq 1)$, defining facet $[p_0, p_3]$ where $p_0 = (1, 0)$ and $p_3 = (1, 1)$, has a zero coefficient for the slack variable, so that it encodes the non-strict inequality $x \geq 1$ of the NNC polyhedron $\mathcal{P}$. In contrast, the constraint $\beta_2 \equiv (-x - \epsilon \geq -3)$, defining facet $[p_1, p_2]$ where $p_1 = (3, 0)$ and $p_2 = (2, 1)$, has a negative coefficient for the slack variable, so that it encodes the strict inequality $-x > -3$ of the NNC polyhedron $\mathcal{P}$. The other two facets $[p_0, p_1]$ and $[p_2, p_3]$ correspond to the "technical" constraints $0 \leq \epsilon \leq 1$. Note that $\mathcal{P} = [\![\mathcal{R}_1]\!]_\epsilon$; in particular, point $p_3 \in \mathcal{R}_1$, having a positive coordinate for ϵ, represents the point $p \in P$ and point $p_1 \in \mathcal{R}_1$, having a zero coordinate for ϵ, represents the closure point $c \in C$.

[3] An alternative representation can be adopted where the ϵ dimension is unbounded from below [7, 9].

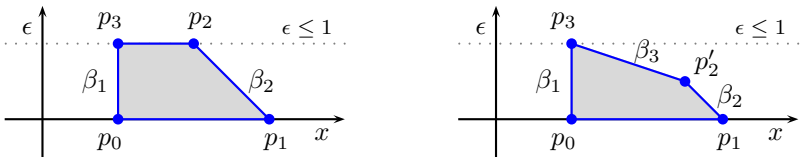

Figure 5: Two ϵ-representations in $\mathbb{CP}_2$ for $\mathcal{P} = \mathrm{con}(\mathcal{C}) \in \mathbb{P}_1$, where $\mathcal{C} = \{1 \leq x, x < 3\}$.

Besides showing its strengths, the work in [9, 14] highlighted the main weakness of the approach: the DD pair in minimal form computed for an ϵ-representation $\mathcal{R}$, when reinterpreted as encoding the NNC polyhedron $\mathcal{P} = [\![\mathcal{R}]\!]_\epsilon$, typically includes many redundant constraints and/or generators, leading to a possibly high computational overhead. To avoid this problem, *strong minimization procedures* were defined in [9, 14] that are able to detect and remove those redundancies; in practice, these procedures map the representation $\mathcal{R}$ into a different representation $\mathcal{R}'$ such that $\mathcal{P} = [\![\mathcal{R}']\!]_\epsilon$, where $\mathcal{R}'$ encodes no ϵ-redundancies.

Example 6. On the right-hand side of Figure 5 we show another ϵ-representation $\mathcal{R}_2 \in \mathbb{CP}_2$ for the same polyhedron $\mathcal{P} \in \mathbb{P}_1$ considered in Example 5. Constraints β_1 and β_2 are the same as before; the new constraint $\beta_3 \equiv (-x - 3 \cdot \epsilon \geq -4)$, defining facet $[p'_2, p_3]$ where $p'_2 = (2.5, 0.5)$, has a negative coefficient for the slack variable, so that it encodes the strict inequality $-x > -4$, which is redundant in any constraint system representing $\mathcal{P}$ (but it is not redundant for representing $\mathcal{R}_2$): hence β_3 is an ϵ-redundant constraint. The strong minimization procedure for constraints [9, 14] is able to transform representation $\mathcal{R}_2$ into representation $\mathcal{R}_1$. It is worth observing that both point $p_2 \in \mathcal{R}_1$ and point $p'_2 \in \mathcal{R}_2$, having a positive coordinate for ϵ, are encoding points that are redundant in any generator system representing $\mathcal{P}$, i.e., they are ϵ-redundant too. If the strong minimization procedure for generators is applied to $\mathcal{R}_1$ (or $\mathcal{R}_2$) one would obtain the triangle defined by points $\{p_0, p_1, p_3\}$, which is another ϵ-representation for $\mathcal{P}$.

When carefully applying strong minimization procedures, a large fraction of the overhead of the ϵ-representation is avoided, leading to implementations that easily meet the efficiency requirements of many application contexts.[4] For the users of the libraries, the addition of the ϵ dimension is almost unnoticed, to the point that quite often the domain of NNC polyhedra is adopted even when not really needed (i.e., when a domain of topologically closed polyhedra would be enough). However, the approach still suffers from two main issues.

1. The ϵ-representation brings with itself an *intrinsic* overhead: in any generator system for an ϵ-polyhedron, most of the "proper" points (those

[4] After being initially implemented and tested in the Parma Polyhedra Library, these strong minimization procedures have also been adopted in the Apron library.

having a positive ϵ coordinate, such as p_3 in Figure 5) need to be paired with the corresponding "closure" point (having a zero ϵ coordinate, such as p_0 in Figure 5). This systematically leads to almost doubling the size of the generator system.

2. The strong minimization procedures, even though effective, interfere with the *incremental* approach of the DD conversion procedures. After applying the strong minimization procedure on the constraint (resp., generator) representation of a DD pair, the dual generator (resp., constraint) representation is lost and, in order to recover it, the non-incremental conversion procedure needs to be applied once again. This also implies that the strong minimization procedures cannot be fully integrated into the DD conversion procedures: they are applied *after* the conversions. As a consequence, during the iterations of the conversion procedure, the redundancies caused by the ϵ-representation are not removed, causing the computation of bigger intermediate results. For the reasons above, the strong minimization procedures are not systematically used in the implementation of the Parma Polyhedra Library; rather, they are applied only when strictly needed for correctness. Therefore, the end user is left with the responsibility of *guessing* whether or not the strong minimization procedures are going to improve efficiency.

The two main problems above will be later recalled when discussing the experimental evaluation in Section 7: in particular, we will provide an evidence for the intrinsic overhead in Table 3. As far as the second problem is concerned, incremental strong minimization procedures could 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.

The adoption of the ϵ-representation approach also incurs another issue that, while being less significant, is worth mentioning. At the implementation level, more work is needed to make the ϵ dimension *invisible* to the end user and, as a matter of fact, its adoption can sometimes become evident. For instance, a strict constraint such as $x > 30$ may be encoded as $2x - \epsilon \geq 60$, which is then shown to the user as the (unsimplified) strict constraint $2x > 60$.[5] Apart from being annoying, the growth in the magnitude of the (arbitrary precision) integer coefficients may cause a significant computational overhead.

4. Direct Representations for NNC Polyhedra

The issues discussed in the previous section were known since [14]. As a matter of fact, both [9] and [12] put forward the possibility of devising an alternative approach regarding the representation and manipulation of NNC polyhedra in the DD framework. Quoting from [12]:

[5] See `https://www.cs.unipr.it/mantis/view.php?id=428`.

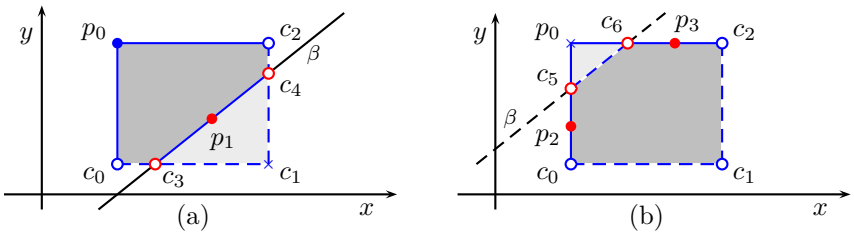

Figure 6: Examples of linear combinations for generators.

> It would be interesting, from both a theoretical and practical point of view, to provide a more direct encoding of NNC polyhedra, i.e., one that is not based on the use of slack variables [...]

The main obstacle on the road towards such a goal is the definition of a conversion procedure that is not only correct, but also competitive with respect to the highly tuned implementations available in software libraries such as Apron and the Parma Polyhedra Library. It is worth stressing that several experimental evaluations, including recent ones [3], confirm that the Parma Polyhedra Library is a state-of-the-art implementation of the DD method for a wide spectrum of application contexts.

4.1. A simple but inefficient approach

As briefly recalled in Section 2, an NNC polyhedron can be directly described by using an extended constraint system $\mathcal{C} = \langle C^{=}, C^{\geq}, C^{>} \rangle$, possibly containing strict inequalities, and/or an extended generator system $\mathcal{G} = \langle L, R, C, P \rangle$, possibly containing closure points. These representations are said to be *geometric*, meaning that they provide a precise description of the position of all the elements in the constraint/generator system.

A first attempt to the specification of a conversion procedure based on these direct (i.e., with no additional ϵ-dimension) geometric encodings for NNC polyhedra was performed in [49] (see also [55]). The Chernikova-like conversion algorithm proposed in [49] works like the classical one (presented informally in Section 2.2), incrementally adding one strict or non-strict inequality constraint at each iteration and partitioning the generators into the sets $\mathcal{G}^{+}$, $\mathcal{G}^{0}$, $\mathcal{G}^{-}$. When linearly combining the generators, it performs a systematic case analysis on the two input generators, so as to identify the kind of generator(s) resulting from each combination.

In Figure 6 we show a few examples of the combinations of generators performed when adding a new constraint to a polyhedron. As done in previous figures, solid (resp., dashed) lines represent non-strict (resp., strict) inequality constraints; as for generators, unfilled (resp., filled) circles represent closure (resp., proper) points; moreover, we denote by blue crosses those generators that are being removed and we color in red those generators that are being added by the incremental conversion procedure.

The input polyhedron is an NNC rectangle in $\mathbb{R}^2$ defined by point p_0 and closure points c_0, c_1, c_2. In case (a), shown on the left-hand side of Figure 6, a new non-strict inequality is added: the algorithm requires the linear combination of generators lying on different sides of the constraint. Here, $c_0 \in \mathcal{G}^+$ is combined with $c_1 \in \mathcal{G}^-$, yielding c_3, and $c_2 \in \mathcal{G}^+$ is combined with $c_1 \in \mathcal{G}^-$, yielding c_4. Being obtained by combining a pair of closure points, these newly introduced generators are closure points too. The combination of point $p_0 \in \mathcal{G}^+$ with closure point $c_1 \in \mathcal{G}^-$, instead, leads to the creation of point p_1, which is needed to keep the facet (c_3, c_4) included in the resulting polyhedron (the dark shaded part of the rectangle). The latter case shows the main drawback of the approach: also the pairs of generators that are *not* adjacent in the input polyhedron need to be combined. It is worth to consider that this combination is computed even in the classical approach based on the addition of the slack variable: c_1 and p_0 would be encoded as points $(c_1^{\mathrm{T}}, 0)^{\mathrm{T}}$ and $(p_0^{\mathrm{T}}, \epsilon)^{\mathrm{T}}$, with $\epsilon > 0$, that are adjacent in the ϵ-representation in $\mathbb{CP}_3$ (assuming that there are no ϵ-redundant generators) and the classical conversion algorithm works as expected. Roughly speaking, the ϵ-representation makes explicitly adjacent those pairs that, even if not adjacent in the direct encoding, have to be combined. It follows that, when using a plain direct approach, the conversion procedure cannot exploit the adjacency test: all pairs of generators lying on different sides of the constraint have to be combined and many redundant elements are produced, making the procedure unusable from a practical perspective.

In case (b) of Figure 6, a strict constraint is added to the same original polyhedron of case (a). Here, point p_0 is combined with c_0 and c_2, yielding c_5 and c_6 respectively (the combination with c_1 is later found to be redundant); the new generators c_5 and c_6 are defined as closure points because they saturate a strict constraint. But again, with respect to the original procedure, some additional points have to be created: points p_2 and p_3 are introduced to "fill" the segments (c_0, c_5) and (c_2, c_6) respectively.

As the examples show, the conversion procedure for the direct approach differs from the classical one mainly in the additional creation of some *points*. These insertions are not foreseen by the classical procedure in that, firstly, non adjacent combinations are required, and secondly, the new points are not necessarily collocated on the added constraint. The new representation described in the following sections is based on the observation that the points added in these "spurious" cases are created only to keep some faces included in the resulting polyhedron; moreover, their geometric position can actually vary inside those faces. Hence, we will now focus on the role of a point inside a face, aiming to better characterize the problematic cases and recover an efficient procedure.

4.2. Towards a refined, efficient approach

For a closed polyhedron $\mathcal{P} \in \mathbb{CP}_n$, the use of completely geometric representations is an adequate choice: it is possible to provide a DD pair $(\mathcal{C}, \mathcal{G})$ that is

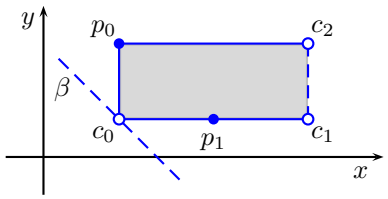

Figure 7: An NNC polyhedron having no "canonical" geometric representations.

"canonical".[6] In the case of an NNC polyhedron $\mathcal{P} \in \mathbb{P}_n$, the adoption of a completely geometric representation can be seen as an overkill, since the knowledge of the precise geometric position of some of the elements is not really needed.

Example 7. As running example, consider the NNC polyhedron $\mathcal{P} \in \mathbb{P}_2$ in Figure 7. The polyhedron can be seen to be described by generator system $\mathcal{G} = \langle L, R, C, P \rangle$, where $L = R = \emptyset$, $C = \{c_0, c_1, c_2\}$ and $P = \{p_0, p_1\}$. However, there is no need to know the precise position of point p_1, since it can be replaced by any other point on the open segment (c_0, c_1). Similarly, when considering the constraint representation, there is no need to know the exact slope of the strict inequality constraint β, as it can be replaced by any other strict inequality that is satisfied by all the points in $\mathcal{P}$ and saturated by closure point c_0.

In other words, some of the elements in the geometric representations of NNC polyhedra are better described by combinatorial information, rather than geometric.[7] The following section introduces the terminology and notation needed to reason on this alternative.

4.3. The combinatorial structure of convex polyhedra

A linear inequality or equality constraint $\beta \equiv (\boldsymbol{a}^{\mathrm{T}}\boldsymbol{x} \bowtie b)$ is said to be *valid* for the polyhedron $\mathcal{P} \in \mathbb{CP}_n$ if all the points in $\mathcal{P}$ satisfy β; for each such β, the subset $F = \{\, \boldsymbol{p} \in \mathcal{P} \mid \boldsymbol{a}^{\mathrm{T}}\boldsymbol{p} = b \,\}$ is a *face* of $\mathcal{P}$. We write $\mathit{cFaces}_{\mathcal{P}}$, omitting the subscript when clear from context, to denote the finite set of faces of $\mathcal{P} \in \mathbb{CP}_n$; the set $\mathit{cFaces}_{\mathcal{P}}$ is a meet sublattice of $\mathbb{CP}_n$, having the empty face as bottom element and the whole polyhedron $\mathcal{P}$ as top element. Note that we have

$$\mathcal{P} = \bigcup \{\, \mathrm{relint}(F) \mid F \in \mathit{cFaces}_{\mathcal{P}} \,\}.$$

The face lattice is also known as the combinatorial structure of the polyhedron. If the polyhedron is bounded (i.e., it is a polytope, having no rays and lines),

[6] Strictly speaking, the canonical form for constraints (resp., generators) still depends on the specific representation chosen for the non-redundant set of equality constraints (resp., generating lines). Even those can be made canonical and each software library typically provides its own canonical form.

[7] Note that in Figure 6 point p_1, which is obtained by combining non-adjacent generators, and the additionally required points p_2 and p_3 are indeed points that do not need to be described geometrically.

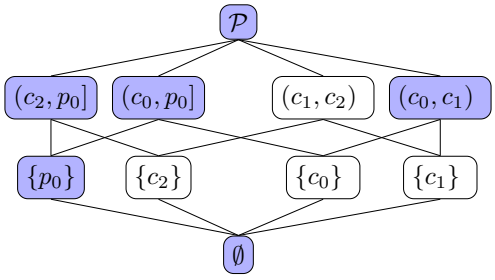

Figure 8: The face lattice $nncFaces_{\mathcal{P}}$ is a join sublattice of $cFaces_{\mathrm{cl}(\mathcal{P})}$.

then the lattice is atomistic, meaning that each face can be obtained as the convex polyhedral hull of the vertices contained in the face.

Even in the case of an NNC polyhedron $\mathcal{P} \in \mathbb{P}_n$ it is possible to define the finite set $nncFaces_{\mathcal{P}}$ of its faces, which is a meet sublattice of $\mathbb{P}_n$; hence, each face is an NNC polyhedron and, as before, we have

$$\mathcal{P} = \bigcup \big\{\, \mathrm{relint}(F) \;\big|\; F \in nncFaces_{\mathcal{P}} \,\big\}.$$

In this case, however, the lattice may be non-atomistic even when the polyhedron is bounded. Letting $\mathcal{Q} = \mathrm{cl}(\mathcal{P})$, the operator $\mathrm{cl}\colon nncFaces_{\mathcal{P}} \to cFaces_{\mathcal{Q}}$ maps each NNC face of $\mathcal{P}$ into a corresponding (closed) face of $\mathcal{Q}$. The image $\mathrm{cl}(nncFaces_{\mathcal{P}})$ is a join sublattice of $cFaces_{\mathcal{Q}}$; meets are generally not preserved, since there may exist $F_1, F_2 \in nncFaces_{\mathcal{P}}$ such that

$$F_1 \cap F_2 = \emptyset \neq \mathrm{cl}(F_1) \cap \mathrm{cl}(F_2).$$

The image of the set of non-empty faces $\mathrm{cl}\big(nncFaces_{\mathcal{P}} \setminus \{\emptyset\}\big)$ is an upward closed subset of $cFaces_{\mathcal{Q}}$; hence, it can be efficiently described by recording just the set of its minimal elements. For each NNC face $F \subseteq \mathcal{P}$ corresponding to one of these minimal elements (that is, for each atom of the $nncFaces_{\mathcal{P}}$ lattice), we have $F = \mathrm{relint}(F)$. As a consequence, the combinatorial structure of $\mathcal{P} \in \mathbb{P}_n$ can be described by integrating the combinatorial structure of its topological closure $\mathcal{Q} \in \mathbb{CP}_n$ with the information identifying the atoms of $nncFaces_{\mathcal{P}}$.

Example 8. Figure 8 shows the face lattice for $\mathrm{cl}(\mathcal{P})$ and its join sublattice $nncFaces_{\mathcal{P}}$ (the shaded nodes) for the polyhedron $\mathcal{P}$ in Figure 7. The atoms of $nncFaces_{\mathcal{P}}$ are the 0-dimension face $\{p_0\}$ and the 1-dimension open segment (c_0, c_1); note that both atoms are relatively open sets so that, for instance, $\mathrm{relint}(\{p_0\}) = \{p_0\}$. Also note that, even if $\mathcal{P}$ is an NNC polytope, the lattice is not atomistic: for instance, the half-open segment $(c_0, p_0]$ is a 1-dimension face that cannot be obtained by joining the atoms.

4.4. Skeleton and non-skeleton

Let $\mathcal{P} \in \mathbb{P}_n$ be an NNC polyhedron and $\mathcal{Q} = \mathrm{cl}(\mathcal{P}) \in \mathbb{CP}_n$ be its topological closure. As explained above, a description of $\mathcal{P}$ can be obtained by combining

a geometric representation of $\mathcal{Q}$, which will be called the *skeleton*[8] component, with some combinatorial information related to $nncFaces_{\mathcal{P}}$ (the *non-skeleton* component). We now provide formal definitions that allow for splitting a fully geometric representation for $\mathcal{P}$ into these two components. For exposition purposes, here we will consider the generator system representation only; the definitions for the constraint system representation are similar and will be briefly described in a later section.

Definition 1 (Skeleton of a generator system). Let $\mathcal{G} = \langle L, R, C, P\rangle$ be a generator system in minimal form, $\mathcal{P} = \mathrm{gen}(\mathcal{G})$ and $\mathcal{Q} = \mathrm{cl}(\mathcal{P})$. The *skeleton* of $\mathcal{G}$ is the generator system

$$\mathcal{SK}_{\mathcal{Q}} \doteq \langle L, R, C \cup SP, \emptyset\rangle,$$

where $SP \subseteq P$ is the set of points that cannot be obtained as a combination of the other generators in $\mathcal{G}$.

Note that the skeleton has no points at all, so that $\mathrm{gen}(\mathcal{SK}_{\mathcal{Q}}) = \emptyset$. However, we can define a variant function '$\overline{\mathrm{gen}}$', that reinterprets the closure points to be points,

$$\overline{\mathrm{gen}}\big(\langle L, R, C, P\rangle\big) \doteq \mathrm{gen}\big(\langle L, R, \emptyset, C \cup P\rangle\big),$$

so as to obtain the following result.

Proposition 1. *Let $\mathcal{P} = \mathrm{gen}(\mathcal{G})$ and $\mathcal{Q} = \mathrm{cl}(\mathcal{P})$. Then*

$$\overline{\mathrm{gen}}(\mathcal{G}) = \overline{\mathrm{gen}}(\mathcal{SK}_{\mathcal{Q}}) = \mathcal{Q}.$$

Also, there does not exist $\mathcal{G}' \subset \mathcal{SK}_{\mathcal{Q}}$ such that $\overline{\mathrm{gen}}(\mathcal{G}') = \mathcal{Q}$.

In other words, the skeleton of an NNC polyhedron can be seen to provide a non-redundant representation of its topological closure. The elements of $SP \subseteq P$ are called *skeleton points*; the non-skeleton points in $P \setminus SP$ are redundant when representing the topological closure, since they can be obtained by combining the lines in L, the rays in R and the closure points in C; these *non-skeleton points* are the elements in $\mathcal{G}$ that need not to be represented geometrically.

Example 9. For the polyhedron in Figure 7, $\mathcal{SK}_{\mathcal{Q}} = \langle \emptyset, \emptyset, \{c_0, c_1, c_2, p_0\}, \emptyset\rangle$, so that p_0 is a skeleton point and p_1 is a non-skeleton point (it can be generated by combining c_0 and c_1). Therefore, $\overline{\mathrm{gen}}(\mathcal{SK}_{\mathcal{Q}}) = \mathcal{Q}$ is the topologically closed rectangle having vertices c_0, c_1, c_2 and p_0.

Having modeled the skeleton component for $\mathcal{P} = \mathrm{gen}\big(\langle L, R, C, P\rangle\big)$, we now turn our attention to the non-skeleton component. As discussed in Section 4.3, our goal is to provide a combinatorial representation for the set of points P.

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

Reasoning slightly more generally, consider a point $\boldsymbol{p} \in \mathcal{Q} = \mathrm{cl}(\mathcal{P})$ (not necessarily in P). There exists a single face $F \in \mathit{cFaces}_{\mathcal{Q}}$ such that $\boldsymbol{p} \in \mathrm{relint}(F)$. By definition of function 'gen', point $\boldsymbol{p}$ behaves as a *filler* for $\mathrm{relint}(F)$, meaning that, when combined with the skeleton, it generates $\mathrm{relint}(F)$. Note that $\boldsymbol{p}$ also behaves as a filler for the relative interiors of all the faces in the set $\uparrow F$. The choice of $\boldsymbol{p} \in \mathrm{relint}(F)$ is actually arbitrary: any other point of $\mathrm{relint}(F)$ would be equivalent as a filler.

Proposition 2. *Consider a polyhedron $\mathcal{P} = \mathrm{gen}(\mathcal{G})$, where $\mathcal{G} = \langle L, R, C, P\rangle$. For $\boldsymbol{p} \in P$, let F be the face of $\mathcal{Q} = \mathrm{cl}(\mathcal{P})$ such that $\boldsymbol{p} \in \mathrm{relint}(F)$; let $\boldsymbol{p}' \in \mathrm{relint}(F)$, and $P' = P \setminus \{\boldsymbol{p}\} \cup \{\boldsymbol{p}'\}$. Then $\mathcal{P} = \mathrm{gen}\big(\langle L, R, C, P'\rangle\big)$.*

Thus, a less arbitrary representation for $\mathrm{relint}(F)$ is provided by its own skeleton, i.e., the system $\mathcal{SK}_F \subseteq \mathcal{SK}_{\mathcal{Q}}$ such that $\overline{\mathrm{gen}}(\mathcal{SK}_F) = F$: namely, each (geometric) filler $\boldsymbol{p} \in \mathcal{P}$ can be mapped into a more abstract (combinatorial) representation, the subset of $\mathcal{SK}_{\mathcal{Q}}$ identifying the corresponding face. For each face $F \in \mathit{cFaces}_{\mathcal{Q}}$, we say that $\mathcal{SK}_F$ is the *support* for the points in $\mathrm{relint}(F)$ and that any point $\boldsymbol{p}' \in \mathrm{relint}\big(\overline{\mathrm{gen}}(\mathcal{SK}_F)\big) = \mathrm{relint}(F)$ is a *materialization* of $\mathcal{SK}_F$.

Definition 2 (Support sets for a skeleton). Let $\mathcal{SK}$ be the skeleton of an NNC polyhedron and let $\mathcal{Q} = \overline{\mathrm{gen}}(\mathcal{SK}) \in \mathbb{CP}_n$. Then the set $\mathbb{NS}_{\mathcal{SK}}$ of all supports for $\mathcal{SK}$ is defined as

$$\mathbb{NS}_{\mathcal{SK}} \doteq \{\, \mathcal{SK}_F \subseteq \mathcal{SK} \mid F \in \mathit{cFaces}_{\mathcal{Q}} \,\}.$$

By definition, the set $\mathbb{NS}_{\mathcal{SK}}$ is a lattice isomorphic to $\mathit{cFaces}_{\mathcal{Q}}$; we will drop the subscripts $\mathcal{SK}$ and $\mathcal{Q}$ when clear from context.

We now define a pair of abstraction and concretization functions mapping a subset of the (geometric) points of an NNC polyhedron into the set of supports that are filled by these points, and vice versa.

Definition 3 (Filled supports). Let $\mathcal{SK}$ be the skeleton of the polyhedron $\mathcal{P} \in \mathbb{P}_n$, $\mathcal{Q} = \mathrm{cl}(\mathcal{P})$ and $\mathbb{NS}$ be the corresponding set of supports. The abstraction function $\alpha_{\mathcal{SK}} \colon \wp(\mathcal{Q}) \to \wp_{\uparrow}(\mathbb{NS})$ is defined, for each $S \subseteq \mathcal{Q}$, as

$$\alpha_{\mathcal{SK}}(S) \doteq \bigcup \big\{\, \uparrow \mathcal{SK}_F \;\big|\; \exists \boldsymbol{p} \in S, F \in \mathit{cFaces} \,.\, \boldsymbol{p} \in \mathrm{relint}(F) \,\big\}.$$

The concretization function $\gamma_{\mathcal{SK}} \colon \wp_{\uparrow}(\mathbb{NS}) \to \wp(\mathcal{Q})$, for each $\mathit{NS} \in \wp_{\uparrow}(\mathbb{NS})$, is defined as

$$\gamma_{\mathcal{SK}}(\mathit{NS}) \doteq \bigcup \Big\{\, \mathrm{relint}\big(\overline{\mathrm{gen}}(\mathit{ns})\big) \;\Big|\; \mathit{ns} \in \mathit{NS} \,\Big\}.$$

Proposition 3. *The pair of functions $(\alpha_{\mathcal{SK}}, \gamma_{\mathcal{SK}})$ is a Galois connection.*

By Proposition 3, the composition $(\gamma_{\mathcal{SK}} \circ \alpha_{\mathcal{SK}})$ is an upper closure operator mapping each non-empty set of points $S \subseteq \mathcal{Q}$ into the smallest NNC polyhedron containing S and having $\mathcal{SK}$ as the skeleton component. In particular, the following result holds.

Proposition 4. *Let $\mathcal{P} = \mathrm{gen}\big(\langle L, R, C, P\rangle\big) \in \mathbb{P}_n$ and let $\mathcal{SK}$ be the corresponding skeleton component. Then $\mathcal{P} = (\gamma_{\mathcal{SK}} \circ \alpha_{\mathcal{SK}})(P)$.*

The non-skeleton component of a geometric generator system can be abstracted by '$\alpha_{\mathcal{SK}}$' and described as a combination of skeleton generators.

Definition 4 (Non-skeleton of a generator system). Let $\mathcal{P} \in \mathbb{P}_n$ be defined by generator system $\mathcal{G} = \langle L, R, C, P\rangle$ and let $\mathcal{SK}$ be the corresponding skeleton component. The *non-skeleton* component of $\mathcal{G}$ is defined as $NS_{\mathcal{G}} \doteq \alpha_{\mathcal{SK}}(P)$.

Even in this case, we will drop the subscript when clear from context. Note that, by definition of the abstraction function '$\alpha_{\mathcal{SK}}$', the non-skeleton component NS contains an upward closed set of supports representing *all* the faces of the NNC polyhedron.

Example 10. We now show the non-skeleton component for the polyhedron in Figure 7. As seen in Example 9, $\mathcal{SK} = \langle \emptyset, \emptyset, C, \emptyset\rangle$, where $C = \{c_0, c_1, c_2, p_0\}$; since in this case we will have $L = R = \emptyset$ in all the skeletons we are going to represent, we will adopt a simplified notation, identifying each support with the C component only. By Definition 3, we have:

$$\alpha_{\mathcal{SK}}\big(\{p_0\}\big) = \big\{\, \{p_0\}, \{c_0, p_0\}, \{c_2, p_0\}, \{c_0, c_1, c_2, p_0\} \,\big\},$$
$$\alpha_{\mathcal{SK}}\big(\{p_1\}\big) = \big\{\, \{c_0, c_1\}, \{c_0, c_1, c_2, p_0\} \,\big\};$$

hence, the non-skeleton component is computed as

$$NS_{\mathcal{G}} = \alpha_{\mathcal{SK}}\big(\{p_0, p_1\}\big) = \big\{\, \{p_0\}, \{c_0, p_0\}, \{c_2, p_0\}, \{c_0, c_1, c_2, p_0\}, \{c_0, c_1\} \,\big\}.$$

By combining Definition 4 with Proposition 4 we obtain the following result, stating that the new representation is semantically equivalent to the fully geometric one.

Corollary 1. *For a polyhedron $\mathcal{P} = \mathrm{gen}(\mathcal{G}) \in \mathbb{P}_n$, let $\langle \mathcal{SK}, NS\rangle$ be the skeleton and non-skeleton components for $\mathcal{G}$. Then $\mathcal{P} = \gamma_{\mathcal{SK}}(NS)$.*

4.5. An efficient encoding for the new representation

In the previous sections we have shown how the geometric generator system $\mathcal{G}$ can be equivalently represented by the pair $\langle \mathcal{SK}, NS\rangle$, where $\mathcal{SK} = \langle L, R, C \cup SP, \emptyset\rangle$ is the skeleton component and $NS \subseteq \wp_{\uparrow}(\mathbb{NS})$ is the non-skeleton component. We now discuss a few minor adaptations to this representation that are meant to result in efficiency improvements at the implementation level.

First, observe that every support $ns \in NS$ always includes all of the lines in the L skeleton component; hence, these lines can be left *implicit* in the representation of the supports in NS. Note that, even after removing the lines, each $ns \in NS$ is still a non-empty set, since it includes at least one closure point.

When lines are implicit, those supports $ns \in NS$ that happen to be singletons[9] can be seen to play a special role: they correspond to the combinatorial encoding of the skeleton points in SP (see Definition 1). These points are not going to benefit from the combinatorial representation, since their geometric position is uniquely identified (modulo the lines component). Therefore, we will remove them from the non-skeleton NS and directly include them in the point component of the skeleton $\mathcal{SK}$; namely, the skeleton $\mathcal{SK} = \langle L, R, C \cup SP, \emptyset \rangle$ will be actually represented as $\mathcal{SK} = \langle L, R, C, SP \rangle$.

Moreover, since NS is a finite upward closed set, the representation only needs to record its minimal elements. At the implementation level, each support $ns \in NS$ can be encoded by using a *set of indices* on the data structure representing the skeleton component $\mathcal{SK}$: hence, the non-minimal elements can be efficiently identified (and removed) by performing appropriate inclusion tests on these sets. When also considering the optimization for skeleton points mentioned before, we can adopt the following definition of redundancy.

Definition 5 (Redundant support). A support $ns \in NS$ is said to be *redundant in* $\langle \mathcal{SK}, NS \rangle$ if there exists $ns' \in NS$ such that $ns' \subset ns$ or if $ns \cap SP \neq \emptyset$, where $\mathcal{SK} = \langle L, R, C, SP \rangle$.

In the following, we will write $NS_1 \oplus NS_2$ to denote the non-redundant union of the support sets $NS_1, NS_2 \subseteq \mathbb{NS}_{\mathcal{SK}}$.

When adopting this more efficient encoding, a polyhedron $\mathcal{P} \in \mathbb{P}_n$ can be defined by a generator system $\mathcal{G} = \langle \mathcal{SK}, NS \rangle$ as follows:

$$\mathcal{P} = \mathrm{gen}(\langle \mathcal{SK}, NS \rangle) \doteq \mathrm{gen}(\mathcal{SK}) \cup \gamma_{\mathcal{SK}}(\uparrow NS).$$

We stress that this is just an optimization: the formalization presented in the previous section is still valid.

Example 11. The polyhedron shown in Figure 7 is $\mathcal{P} = \mathrm{gen}(\langle \mathcal{SK}, NS \rangle)$ with $\mathcal{SK} = \langle L, R, C, SP \rangle = \langle \emptyset, \emptyset, \{c_0, c_1, c_2\}, \{p_0\} \rangle$, $NS = \{ns\}$ and $ns = \{c_0, c_1\}$.

4.6. Representing constraints: duality

The definitions and observations given in the previous sections for a geometric generator system have their dual versions working on a geometric *constraint* system. In the following we provide a brief overview of these correspondences, which are also summarized in Table 1.

For a non-empty $\mathcal{P} = \mathrm{con}(\mathcal{C}) \in \mathbb{P}_n$, the skeleton component of the geometric constraint system $\mathcal{C} = \langle C^=, C^\geq, C^> \rangle$ includes the non-redundant constraints defining the topological closure $\mathcal{Q} = \mathrm{cl}(\mathcal{P})$. Denoting by $SC^>$ the set of *skeleton strict inequalities* (i.e., those in $C^>$ whose corresponding non-strict inequality

[9] Since the support ns is a subset of the skeleton $\mathcal{SK}$, by 'singleton' here we mean a system $ns = \langle \emptyset, \emptyset, \{\boldsymbol{p}\}, \emptyset \rangle$.

	Generators	Constraints
Geometric skeleton		
singular	line	equality
non-singular	ray or closure point	non-strict inequality
semantics	$\mathrm{gen}(\mathcal{SK}) = \emptyset$	$\mathrm{con}(\mathcal{SK}) = \mathrm{cl}(\mathcal{P})$
Combinatorial non-skeleton		
abstracts	point	strict inequality
element role	face filler	face cutter
represents	upward closed set	downward closed set
encoding	minimal support	minimal support
singleton	skeleton point	skeleton strict inequality

Table 1: Correspondences between generator and constraint concepts.

is not redundant for $\mathcal{Q}$), we can define $\mathcal{SK}_{\mathcal{Q}} \doteq \langle C^{=}, C^{\geq} \cup SC^{>}, \emptyset \rangle$, so that $\mathcal{Q} = \mathrm{con}(\mathcal{SK}_{\mathcal{Q}})$.

The *ghost* faces of $\mathcal{P}$ are the faces of the topological closure $\mathcal{Q}$ that do not intersect $\mathcal{P}$:

$$gFaces_{\mathcal{P}} \doteq \{\, F \in cFaces_{\mathcal{Q}} \mid F \cap \mathcal{P} = \emptyset \,\};$$

as a consequence, we obtain $\mathcal{P} = \mathrm{con}(\mathcal{SK}_{\mathcal{Q}}) \setminus \bigcup gFaces_{\mathcal{P}}$.

With the only exception of the empty face, the elements in *gFaces* are exactly those not occurring in $\mathrm{cl}(nncFaces)$. The set $gFaces' \doteq gFaces \cup \{\mathcal{Q}\}$ is a meet sublattice of *cFaces*; moreover, *gFaces* is downward closed and thus can be efficiently represented by its *maximal* elements (with respect to the set inclusion relation on faces), which are the dual-atoms of $gFaces'$.

We define function '$\overline{\mathrm{con}}$' as a variant of function 'con' that reinterprets the inequalities to be equalities. Hence, the skeleton support of a face $F \in cFaces_{\mathcal{Q}}$ is the system $\mathcal{SK}_F \subseteq \mathcal{SK}_{\mathcal{Q}}$ such that $\mathcal{Q} \cap \overline{\mathrm{con}}(\mathcal{SK}_F) = F$. Each face $F \in gFaces$ saturates a strict inequality $\beta^{>} \in C^{>}$: we can represent such a face using its skeleton support $\mathcal{SK}_F$ of which $\beta^{>}$ is a possible materialization. Thus, a constraint system non-skeleton component $NS \subseteq \mathbb{NS}$ is a combinatorial representation of the *strict inequalities* of the polyhedron.

Hence, the non-skeleton components for generators and constraints have a complementary role: in the case of generators they are face *fillers*, marking the minimal faces that are *included* in *nncFaces*; in the case of constraints they are face *cutters*, marking the maximal faces that are *excluded* from *nncFaces*. Note however that, when representing a cutter in *gFaces* using its skeleton support, the non-redundant cutters are again those having a *minimal* skeleton support, as is the case for the fillers.

As it happens with lines, all the equalities in $C^{=}$ are included in all the supports $ns \in NS$ so that, for efficiency, they are not represented explicitly. After removing the equalities, a singleton $ns = \{\beta\} \in NS$ stands for a *skeleton strict inequality* constraint, which is better represented in the skeleton compo-

nent, thereby obtaining $\mathcal{SK} = \langle C^{=}, C^{\geq}, SC^{>} \rangle$. Hence, a support $ns \in NS$ is redundant if there exists $ns' \in NS$ such that $ns' \subset ns$ or if $ns \cap SC^{>} \neq \emptyset$.

A polyhedron $\mathcal{P} \in \mathbb{P}_n$ can be defined by a constraint system in the new representation $\mathcal{C} = \langle \mathcal{SK}, NS \rangle$ as follows:

$$\mathcal{P} = \mathrm{con}(\langle \mathcal{SK}, NS \rangle) \doteq \mathrm{con}(\mathcal{SK}) \setminus \bigcup_{ns \in NS} \overline{\mathrm{con}}(ns).$$

Example 12. The polyhedron shown in Figure 7 is $\mathcal{P} = \mathrm{con}(\langle \mathcal{SK}, NS \rangle)$ with $\mathcal{SK} = \langle C^{=}, C^{\geq}, SC^{>} \rangle = \langle \emptyset, \{x \geq 2, y \geq 1, y \leq 3\}, \{x < 7\} \rangle$, $NS = \{ns\}$ and $ns = \{x \geq 2, y \geq 1\}$; as a matter of fact, the geometric constraint β in Figure 7 is a possible materialization of ns, excluding point $\{(2, 1)\} = \overline{\mathrm{con}}(ns)$.

The handling of the empty face deserves a technical observation (which can be skipped when adopting a higher level point of view). The empty face is always cut away from the polyhedron, hence it belongs to *gFaces* even when $\mathcal{P}$ is topologically closed. The skeleton support for the empty face can be given by a set of skeleton constraints whose hyperplanes have an empty intersection or by a constraint that is saturated by no points or closure points: the latter happens to be the case for the positivity constraint '$1 \geq 0$' (see Section 2), which is saturated by the generators in $R \cup L$. It follows that, when the positivity constraint is not redundant, the empty face should be represented by the non-skeleton support $ns = \{1 \geq 0\}$; being a singleton, this will be promoted into the skeleton component, thereby encoding the positivity constraint as a *strict* inequality '$1 > 0$'. Otherwise, when the positivity constraint is redundant, the empty face will be cut by the support $ns = \mathrm{sat}(R \cup L, \mathcal{SK})$.

5. The New Conversion Algorithm

Having introduced the new representation for NNC polyhedra, we are now ready to extend the Chernikova's conversion algorithm to directly handle strict inequalities and closure points. Building on the distinction between the skeleton and non-skeleton components, we will pursue a corresponding separation in the conversion procedure: while the skeleton component can be handled by following a minor variant of the classical procedure for closed polyhedra (performing the usual adjacency tests to avoid redundancies), for the non-skeleton component we develop a few brand new procedures that can correctly deal with closure points and strict inequalities without incurring into a significant overhead. As already pointed out in Section 2, we will focus on the conversion from constraints to generators. The conversion working the other way round will be obtained by applying duality arguments.

When describing the algorithm we will proceed top-town, so that helper functions and procedures will be explained only after their uses. Also note that we will write in small capitals (e.g., 'CONVERSION') the names of functions for which we provide the pseudocode; in contrast, we will write in roman the names of those functions that are provided with a mathematical specification

Pseudocode 1 Conversion from constraints to generators.

```
   function CONVERSION(C_in, G)
2:     let C_in = ⟨SK^c_in, NS^c_in⟩;
       for all β ∈ SK^c_in do
4:         G ← SKEL_CONVERSION(β, G);
           if G = ⟨∅, ∅⟩ then
6:             return G;                                  ▷ P is empty
       for all ns ∈ NS^c_in do
8:         G ← NONSKEL_CONVERSION(ns, G);
       return G;
```

and no pseudocode (e.g., 'points_become_closure_points') or even left unspecified because they are trivial (e.g., 'is_strict_ineq').

The 'CONVERSION' function in Pseudocode 1 works incrementally, adding the constraints in $\mathcal{C}_{in}$ one at a time to the generator system $\mathcal{G}$, keeping it up to date. The function first processes all of the geometric constraints from the skeleton component $\mathcal{SK}^{\mathrm{c}}_{in}$ of the constraint system, checking after each iteration if the polyhedron has become empty (lines 5 to 6); then it processes the combinatorial constraints from the non-skeleton component NS^{c}_{in} in input.

5.1. Processing a geometric constraint

When modeling the incremental addition of the geometric constraint $\beta \in \mathcal{SK}^{\mathrm{c}}_{in}$ to the generator system $\mathcal{G} = \langle \mathcal{SK}, NS \rangle$, function 'SKEL_CONVERSION' in Pseudocode 2 operates on the two components $\mathcal{SK}$ and NS separately, integrating the results at the end. We first focus on the skeleton component $\mathcal{SK} = \langle L, R, C, SP \rangle$.

Handling the skeleton component.

The first processing step (line 3) is the partitioning of the skeleton $\mathcal{SK}$ according to the signs of the scalar products with constraint β. Since the skeleton component is entirely geometric, it can be split into $\mathcal{SK}^+$, $\mathcal{SK}^0$ and $\mathcal{SK}^-$ exactly as done in the Chernikova's algorithm. In the pseudocode, this partition info is kept implicit inside the data structure encoding $\mathcal{SK}$: we will use the superscripts to refer to each component as needed.

Lines 5 to 6 of 'SKEL_CONVERSION' are meant to take care of a line violating β, whereas lines 7 to 22 are meant to efficiently handle those special cases when $\mathcal{SK}^+$ or $\mathcal{SK}^-$ happens to be empty; these will be briefly discussed later on.

The second main processing step for the skeleton component occurs in lines 24 to 25, where function 'comb_adj$_\beta$' combines the generators in $\mathcal{SK}^+$ and $\mathcal{SK}^-$ to produce $\mathcal{SK}^\star$, which is then merged into $\mathcal{SK}^0$. This step too is quite similar to the one for closed polyhedra described in Section 2, except that we now have to consider how the different generator kinds combine with each other, according to the kind of constraint β: the systematic case analysis is presented in Table 2 and it is restricted to the combinations of adjacent generators (which

Pseudocode 2 Adding a geometric constraint to a generator system.

```
    function SKEL_CONVERSION(β, ⟨SK, NS⟩)
2:      let G = ⟨SK, NS⟩;
        skel_partition(β, SK);
4:      nonskel_partition(⟨SK, NS⟩);
        if line l ∈ SK^+ ∪ SK^− then                          ▷ β violates line l
6:          VIOLATING_LINE(β, l, ⟨SK, NS⟩);
        else if SK^− = ∅ then
8:          if is_equality(β) then
                if SK^0 = ∅ then
10:                 return ⟨∅, ∅⟩;                            ▷ P is empty
                else
12:                 return ⟨SK^0, NS^0⟩;
            else if is_strict_ineq(β) then
14:             if SK^+ = ∅ then
                    return ⟨∅, ∅⟩;                            ▷ P is empty
16:             else if SK^0 ≠ ∅ then
                    STRICT_ON_EQ_POINTS(β, ⟨SK, NS⟩);
18:     else if SK^+ = ∅ then
            if is_strict_ineq(β) or SK^0 = ∅ then
20:             return ⟨∅, ∅⟩;                                ▷ P is empty
            else
22:             return ⟨SK^0, NS^0⟩;
        else                                                  ▷ SK^+ ≠ ∅ and SK^− ≠ ∅
24:         SK* ← comb_adj_β(SK^+, SK^−);
            SK^0 ← SK^0 ∪ SK*;
26:         NS* ← MOVE_NS(β, ⟨SK, NS⟩);
            NS* ← NS* ∪ CREATE_NS(β, ⟨SK, NS⟩);
28:         if is_equality(β) then
                ⟨SK, NS⟩ ← ⟨SK^0, NS^0 ⊕ NS*⟩;
30:         else if is_nonstrict_ineq(β) then
                ⟨SK, NS⟩ ← ⟨SK^+ ∪ SK^0, (NS^+ ∪ NS^0) ⊕ NS*⟩;
32:         else                                              ▷ is_strict_ineq(β)
                SK^0 ← points_become_closure_points(SK^0);
34:             ⟨SK, NS⟩ ← ⟨SK^+ ∪ SK^0, NS^+ ⊕ NS*⟩;
            PROMOTE_SINGLETONS(⟨SK, NS⟩);
36:     return ⟨SK, NS⟩;
```

	$\mathcal{SK}^+$	R	R	R	C	C	C	SP	SP	SP
	$\mathcal{SK}^-$	R	C	SP	R	C	SP	R	C	SP
$\beta^=$ or $\beta^\geq$	$\mathcal{SK}^\star$	R	C	SP	C	C	SP	SP	SP	SP
$\beta^>$		R	C	C	C	C	C	C	C	C

Table 2: Case analysis for function 'comb$_\beta$' when adding an equality ($\beta^=$), a non-strict ($\beta^\geq$) or a strict ($\beta^>$) inequality constraint to a pair of generators from $\mathcal{SK}^+$ and $\mathcal{SK}^-$ (R = ray, C = closure point, SP = skeleton point).

is crucial for efficiency). The inclusion of the skeleton points SP in $\mathcal{SK}$ (discussed in Section 4.5), besides simplifying the non-skeleton representation, allows for processing them using the adjacency tests; nonetheless, since the points in SP behave as fillers, they will have to be properly reconsidered when processing the non-skeleton component NS.

The final processing steps for the skeleton component, occurring in lines 28 to 34, are those meant to purge from the skeleton components those generators that are violating the constraint β. An additional processing step (line 33) is needed for the case of a strict inequality constraint: the helper function

$$\text{points\_become\_closure\_points}\big(\langle L, R, C, SP\rangle\big) \doteq \langle L, R, C \cup SP, \emptyset\rangle,$$

applied to $\mathcal{SK}^0$, makes sure that all of the skeleton points saturating β are transformed into closure points having the same position.

Handling the non-skeleton component.

We now consider those parts of Pseudocode 2 that operate on the non-skeleton component NS, which is clearly where the 'SKEL_CONVERSION' function significantly differs from the corresponding algorithm working on closed polyhedra.

The first processing step (line 4) is the partitioning of the supports in NS, so as to detect their position with respect to the constraint β. To this end, we can exploit the partition info already computed for the skeleton $\mathcal{SK}$ (line 3) to obtain the corresponding partition info for NS, without computing any additional scalar product. Namely, each support $ns \in NS$ is classified as follows:

$$\begin{aligned}
ns \in NS^+ &\iff ns \subseteq (\mathcal{SK}^+ \cup \mathcal{SK}^0) \wedge ns \cap \mathcal{SK}^+ \neq \emptyset;\\
ns \in NS^0 &\iff ns \subseteq \mathcal{SK}^0;\\
ns \in NS^- &\iff ns \subseteq (\mathcal{SK}^- \cup \mathcal{SK}^0) \wedge ns \cap \mathcal{SK}^- \neq \emptyset;\\
ns \in NS^\pm &\iff ns \cap \mathcal{SK}^+ \neq \emptyset \wedge ns \cap \mathcal{SK}^- \neq \emptyset.
\end{aligned}$$

Note that the partitioning above is fully consistent with respect to the one computed for skeleton elements. For instance, if $ns \in NS^+$, then for every possible materialization $\boldsymbol{p} \in \text{relint}(\overline{\text{gen}}(ns))$ the scalar product of $\boldsymbol{p}$ and β is strictly positive. Things are similar when $ns \in NS^0$ and $ns \in NS^-$. The supports in $NS^\pm$

are those whose materializations can indifferently satisfy, saturate or violate the constraint β (i.e., the corresponding face *crosses* the hyperplane induced by the constraint). As before, the partition info is kept implicit inside the data structure encoding NS.

Pseudocode 3 Helper procedure for promoting singleton supports.

```
procedure PROMOTE_SINGLETONS(⟨SK, NS⟩)
    let SK = ⟨L, R, C, SP⟩;
    for all ns ∈ NS such that ns = ⟨∅, ∅, {c}, ∅⟩ do
        NS ← NS \ {ns};
        C ← C \ {c};
        SP ← SP ∪ {c};
```

After partitioning, a set $NS^\star$ of brand new supports is built using functions 'MOVE_NS' and 'CREATE_NS' (lines 26 and 27). This set will be non-redundantly merged, using operator '$\oplus$', into the appropriate portions of the non-skeleton component, chosen according to the constraint kind (lines 28 to 34). The final processing step (line 35) calls helper procedure 'PROMOTE_SINGLETONS', shown in Pseudocode 3, making sure that all singleton supports get promoted to skeleton points (so as to satisfy the new representation invariant).

Moving supports.

The 'MOVE_NS' function, shown in Pseudocode 4, processes the supports in $NS^\pm$. As hinted by its name, the goal of this function is to "move" the fillers of the faces that are crossed by constraint β, making sure they lie on the correct side.

Pseudocode 4 Helper function for moving supports.

```
    function MOVE_NS(β, ⟨SK, NS⟩)
2:      NS* ← ∅;
        for all ns ∈ NS^± do
4:          NS* ← NS* ∪ {proj^β(supp_cl(ns))};
        return NS*;
```

Let $ns \in NS^\pm$ and consider the face $F = \text{relint}(\overline{\text{gen}}(ns))$. Note that F is a face of the polyhedron *before* the addition of the new constraint β; at this point, the elements in $\mathcal{SK}^\star$ have been added to $\mathcal{SK}^0$, but this change still has to be propagated to the non-skeleton component NS. Therefore, we compute the *support closure* 'supp_cl(ns)' of the support ns according to the updated skeleton $\mathcal{SK}$. Intuitively, supp_cl(ns) $\subseteq \mathcal{SK}$ is the subset of all the skeleton elements that are included in face F. At the implementation level, the support closure operator can be efficiently computed by exploiting the same *saturation information* that is needed to quickly perform the adjacency tests. If $\mathcal{SK}^{\text{c}}$ and $\mathcal{SK}^{\text{g}} = \langle L, R, C, SP\rangle$ are the currently computed skeleton components for the

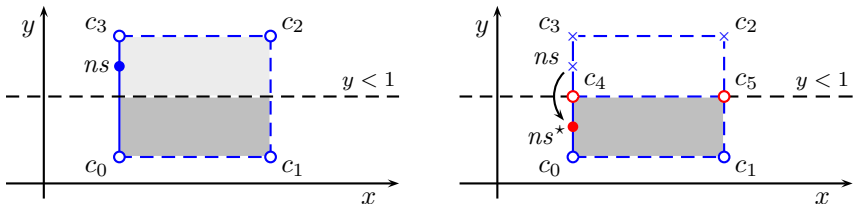

Figure 9: Application of 'MOVE_NS' to $ns \in NS^{\pm}$ when adding a strict inequality.

polyhedron, for each support $ns \in NS^{\mathrm{g}}$ we compute its closure as follows [44]:

$$\mathrm{supp\_cl}(ns) \doteq \mathrm{sat}\big(\mathrm{sat}(ns, \mathcal{SK}^{\mathrm{c}}), \mathcal{SK}^{\mathrm{g}}\big) \setminus L.$$

Face F is split by constraint β into the three subsets F^+, F^0 and F^-. When β is a strict inequality, only F^+ shall be kept in the polyhedron; when β is a non-strict inequality, both F^+ and F^0 shall be kept. When working with the updated support, a non-skeleton representation for these subsets can be obtained by *projecting* the support on the corresponding portions of the skeleton. Namely, we can define the function

$$\mathrm{proj}^{\beta}(ns) \doteq \begin{cases} ns \setminus \mathcal{SK}^{-}, & \text{if } \beta \text{ is a strict inequality;} \\ ns \cap \mathcal{SK}^{0}, & \text{otherwise.} \end{cases}$$

Since the projection operator is applied *after* having computed the support closure, when β is a non-strict inequality we have $ns \cap \mathcal{SK}^0 \neq \emptyset$; hence, the support of F^0 is a subset of the support of F^+ and $\mathrm{proj}^{\beta}(ns)$ will be a filler for F^+ too.

To summarize, by composing support closure and projection in line 4 of 'MOVE_NS', each support in $NS^{\pm}$ is moved to the correct side of β.

Example 13. Consider the polyhedron $\mathcal{P} \in \mathbb{P}_2$ on the left-hand side of Figure 9, described by the skeleton and non-skeleton components $\langle \mathcal{SK}, NS \rangle$. The skeleton $\mathcal{SK} = \langle \emptyset, \emptyset, C, \emptyset \rangle$ is composed by the four closure points in $C = \{c_0, c_1, c_2, c_3\}$; the non-skeleton $NS = \{ns\}$ contains a single support $ns = \{c_0, c_3\}$, which makes sure that the open segment (c_0, c_3) is included in $\mathcal{P}$; in the figure, we show just one of the many possible materializations for ns.

When processing the strict inequality constraint $\beta \equiv (y < 1)$, we obtain the polyhedron on the right-hand side of the figure. In the skeleton phase of the 'SKEL_CONVERSION' function the adjacent skeleton generators are combined: c_4 (combining $c_0 \in \mathcal{SK}^+$ and $c_3 \in \mathcal{SK}^-$) and c_5 (combining $c_1 \in \mathcal{SK}^+$ and $c_2 \in \mathcal{SK}^-$) are added to $\mathcal{SK}^0$. Since the non-skeleton support ns belongs to $NS^{\pm}$, it is processed in the 'MOVE_NS' function:

$$\begin{aligned} ns^* &= \mathrm{proj}^{\beta}\big(\mathrm{supp\_cl}(ns)\big) \\ &= \mathrm{supp\_cl}\big(\{c_0, c_3\}\big) \setminus \mathcal{SK}^{-} \\ &= \{c_0, c_3, c_4\} \setminus \{c_2, c_3\} \\ &= \{c_0, c_4\}. \end{aligned}$$

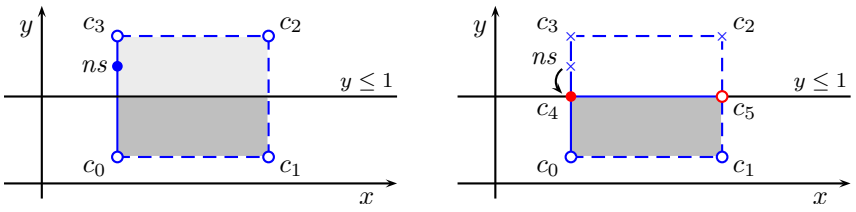

Figure 10: Application of 'MOVE_NS' to $ns \in NS^{\pm}$, adding a non-strict inequality.

Intuitively, we have moved ns to $ns^{\star}$: again, for the new support we show only one of its many possible materializations, but it is clear that now they all satisfy constraint β.

Example 14. On the left-hand side of Figure 10, we reconsider the same polyhedron of Example 13, but we now add the non-strict inequality $\beta' \equiv (y \leq 1)$. The skeleton phase of the 'SKEL_CONVERSION' procedure behaves exactly as shown before, producing closure points c_4 and c_5. We then process $ns \in NS^{\pm}$ in the 'MOVE_NS' function:

$$\begin{aligned} ns^* &= \mathrm{proj}^{\beta'}\big(\mathrm{supp\_cl}(ns)\big) \\ &= \mathrm{supp\_cl}\big(\{c_0, c_3\}\big) \cap \mathcal{SK}^0 \\ &= \{c_0, c_3, c_4\} \cap \{c_4, c_5\} \\ &= \{c_4\}. \end{aligned}$$

Since $ns^{\star}$ is a singleton, it will be upgraded to become a skeleton point by procedure 'PROMOTE_SINGLETONS', thereby obtaining the new skeleton component $\mathcal{SK} = \langle \emptyset, \emptyset, C, SP \rangle$, where $C = \{c_0, c_1, c_5\}$ and $SP = \{c_4\}$, and the new non-skeleton component $NS = \emptyset$. Hence, we obtain the polyhedron on the right-hand side of the figure; note that the skeleton point c_4 is responsible for the inclusion of the facets $(c_0, c_4]$ and $[c_4, c_5)$ in the polyhedron.

Creating new supports.

On the one hand, the choice of representing only the minimal elements of the upward closed set NS enables many efficiency improvements; on the other hand, it also means that some care has to be taken before removing these minimal elements.

As an example, consider the case of a support $ns \in NS^{-}$ when dealing with a non-strict inequality constraint β: this support is going to be removed from NS in line 31 of the 'SKEL_CONVERSION' function. However, by doing so, we are also implicitly removing other supports from the set $\uparrow ns$, here included some supports that do not belong to NS^{-} and therefore should be kept in NS. Thus, at each iteration, we have to explore the set of filled faces and detect the ones that are going to lose their filler: the corresponding minimal supports will be added to $NS^{\star}$. Moreover, when processing a non-strict inequality constraint,

we also need to consider the new faces introduced by the constraint: the corresponding supports can be found by projecting on the constraint hyperplane those faces that are possibly filled by an element in SP^+ or NS^+.

Pseudocode 5 Helper functions for creating new supports.

```
    function CREATE-NS(β, ⟨SK, NS⟩)
2:      NS* ← ∅;
        let SK = ⟨L, R, C, SP⟩;
4:      for all ns ∈ NS⁻ ∪ {{p} | p ∈ SP⁻} do
            NS* ← NS* ∪ ENUMERATE-FACES(β, ns, SK⁺, SK);
6:      if is_strict_ineq(β) then
            for all ns ∈ NS⁰ ∪ {{p} | p ∈ SP⁰} do
8:              NS* ← NS* ∪ ENUMERATE-FACES(β, ns, SK⁺, SK);
        else
10:         for all ns ∈ NS⁺ ∪ {{p} | p ∈ SP⁺} do
                NS* ← NS* ∪ ENUMERATE-FACES(β, ns, SK⁻, SK);
12:     return NS*;
    function ENUMERATE_FACES(β, ns, SK', SK)
2:      NS* ← ∅;
        let SK' = ⟨L', R', C', SP'⟩;
4:      for all g ∈ (R' ∪ C') do
            NS* ← NS* ∪ {proj^β(supp_cl(ns ∪ {g}))};
6:      return NS*;
```

This is the task of the 'CREATE_NS' function, shown in Pseudocode 5. This function uses 'ENUMERATE_FACES' as a helper:[10] the latter provides an enumeration of all the (higher dimensional) faces that contain the initial support ns. The new faces are obtained by adding to ns a new generator g and then composing the projection and support closure functions, as done in function 'MOVE_NS'.

For efficiency purposes, in function 'CREATE_NS' a case analysis is performed so as to suitably restrict the search area of the enumeration phase. Since the faces we are going to compute have to be projected, it is enough to consider those that can cross the constraint: hence, when adding a new generator g to a non-skeleton support ns, we consider only those coming from the opposite side of the constraint (for instance, when processing $ns \in NS^-$ in lines 5 we consider $g \in \mathcal{SK}^+$, disregarding the generators in $\mathcal{SK}^-$ and $\mathcal{SK}^0$). We also avoid adding a point to ns, since this would definitely yield a redundant support.

Example 15. Consider the polyhedron $\mathcal{P} \in \mathbb{P}_2$ on the left-hand side of Figure 11. The skeleton $\mathcal{SK} = \langle \emptyset, \emptyset, C, \emptyset \rangle$ is composed by the four closure points in $C = \{c_0, c_1, c_2, c_3\}$; the non-skeleton $NS = \{ns\}$ contains a single support

[10] This enumeration phase is inspired by the algorithm in [44].

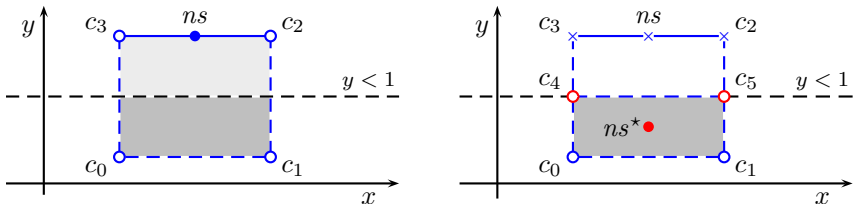

Figure 11: Application of 'CREATE_NS' when adding a strict inequality.

$ns = \{c_2, c_3\}$, which makes sure that the open segment (c_2, c_3) is included in $\mathcal{P}$. By upward closure, this non-skeleton point is also the filler for the whole polyhedron; in particular, it fills relint($\mathcal{P}$).

The strict inequality makes $ns \in NS^-$, since all the generators in the support are in $\mathcal{SK}^-$; hence, support ns is processed by line 5 of function 'CREATE_NS'. The call to function 'ENUMERATE_FACES' will produce new supports by adding to ns a generator from $\mathcal{SK}^+$ and then computing the corresponding support closure and projection. Namely, it will compute

$$\begin{aligned}
\text{proj}^\beta\big(\text{supp_cl}(ns \cup \{c_0\})\big) &= \text{supp_cl}\big(ns \cup \{c_0\}\big) \setminus \mathcal{SK}^- \\
&= \{c_0, c_1, c_2, c_3, c_4, c_5\} \setminus \{c_2, c_3\} \\
&= \{c_0, c_1, c_4, c_5\}, \\
\text{proj}^\beta\big(\text{supp_cl}(ns \cup \{c_1\})\big) &= \text{supp_cl}\big(ns \cup \{c_1\}\big) \setminus \mathcal{SK}^- \\
&= \{c_0, c_1, c_2, c_3, c_4, c_5\} \setminus \{c_2, c_3\} \\
&= \{c_0, c_1, c_4, c_5\}.
\end{aligned}$$

Hence, the new (minimal) support $ns^\star = \{c_0, c_1, c_4, c_5\}$ will be added to $NS^\star$. The resulting polyhedron, shown on the right-hand side of the figure, is described by the skeleton $\mathcal{SK} = \langle \emptyset, \emptyset, \{c_0, c_1, c_4, c_5\}, \emptyset \rangle$ and the non-skeleton $NS = \{ns^\star\}$.

At the implementation level, the face enumeration phase can be further optimized by exploiting saturation information: the search space for generator g (line 4 of function 'ENUMERATE_FACES') can be dynamically pruned so as to avoid the recomputation of those faces that have already been discovered. For instance, in Example 15, it is possible to skip the addition of closure point c_1 to ns, since this would lead to the same support obtained from c_0.

In the following examples we show how function 'CREATE_NS' tackles both the problems identified in Figure 6 of Section 4.1 when applying the naive approach: it takes care of the introduction of the points that were either coming from combinations of non-adjacent generators (case (a) of Figure 6, analyzed in Example 16), or completely additional to the usual combination phase (case (b) of Figure 6, analyzed in Example 17).

Example 16. Consider polyhedron $\mathcal{P} \in \mathbb{P}_2$ on the left-hand side of Figure 12, described by skeleton $\mathcal{SK} = \langle \emptyset, \emptyset, \{c_0, c_1, c_2\}, \{p\} \rangle$ and non-skeleton $NS = \emptyset$. When a new non-strict inequality β is added, function 'comb_adj$_\beta$' combines

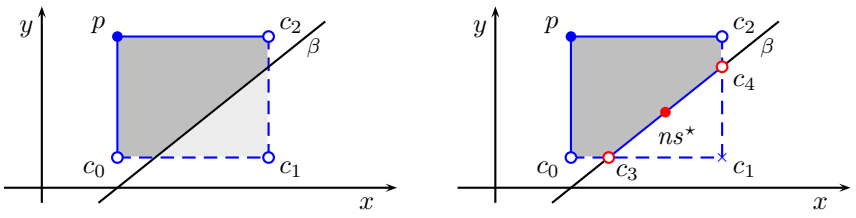

Figure 12: Application of 'CREATE_NS' when adding a non-strict inequality.

only the adjacent generators in $\mathcal{SK}^+$ and $\mathcal{SK}^-$ to produce $\mathcal{SK}^\star = \{c_3, c_4\} = \mathcal{SK}^0$. The processing of the non-skeleton component has to take care that the new facet introduced (c_3, c_4) is filled, since β is a non-strict inequality. The skeleton point in SP^+ is considered in line 11 of function 'CREATE_NS': the corresponding call to function 'ENUMERATE_FACES' produces new supports by first adding to $\{p\}$ each generator in $\mathcal{SK}^-$ and then computing the corresponding support closure and projection. Namely, it will compute

$$\begin{aligned} ns^\star &= \text{proj}^\beta\big(\text{supp\_cl}(\{c_1\} \cup \{p\})\big) \\ &= \text{supp\_cl}\big(\{c_1, p\}\big) \cap \mathcal{SK}^0 \\ &= \{c_0, c_1, c_2, c_3, c_4, p\} \cap \{c_3, c_4\} \\ &= \{c_3, c_4\}, \end{aligned}$$

thereby producing the filler for the open segment (c_3, c_4). The resulting polyhedron, shown on the right-hand side of Figure 12, is described by the skeleton $\mathcal{SK} = \langle \emptyset, \emptyset, \{c_0, c_2, c_3, c_4\}, \{p\} \rangle$ and the non-skeleton $NS = \{ns^\star\}$.

Example 17. On the left-hand side of Figure 13 we show the insertion of a strict constraint β' to the same polyhedron of the previous example; closure points c_3 and c_4 are created as usual by combining adjacent skeleton generators. The face enumeration phase induced by line 5 of function 'CREATE_NS' explores the faces that are filled by point $p \in SP^-$, adding the generators in $\mathcal{SK}^+ = \{c_0, c_1, c_2\}$:

$$\begin{aligned} ns_0^\star &= \text{proj}^{\beta'}\big(\text{supp\_cl}(\{c_0\} \cup \{p\})\big) = \text{supp\_cl}\big(\{c_0, p\}\big) \setminus \mathcal{SK}^- \\ &= \{c_0, c_3, p\} \setminus \{p\} = \{c_0, c_3\}, \\ ns_1^\star &= \text{proj}^{\beta'}\big(\text{supp\_cl}(\{c_1\} \cup \{p\})\big) = \text{supp\_cl}\big(\{c_1, p\}\big) \setminus \mathcal{SK}^- \\ &= \{c_0, c_1, c_2, c_3, c_4, p\} \setminus \{p\} = \{c_0, c_1, c_2, c_3, c_4\}, \\ ns_2^\star &= \text{proj}^{\beta'}\big(\text{supp\_cl}(\{c_2\} \cup \{p\})\big) = \text{supp\_cl}\big(\{c_2, p\}\big) \setminus \mathcal{SK}^- \\ &= \{c_2, c_4, p\} \setminus \{p\} = \{c_2, c_4\}. \end{aligned}$$

Keeping only the minimal supports, we obtain that the resulting polyhedron, shown on the right-hand side of Figure 13, has $NS = \{ns_0^\star, ns_2^\star\}$.

Distinguishing between the skeleton and non-skeleton components, the overall conversion procedure obtains a twofold benefit: first, the combinations of

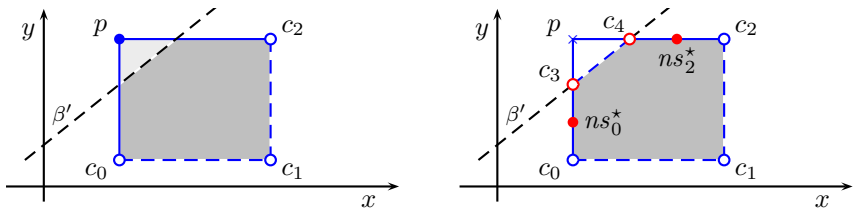

Figure 13: Application of 'CREATE_NS' when adding a strict inequality.

non-adjacent generators is limited to the non-skeleton processing phase, thereby recovering the corresponding optimizations on the skeleton part; second, by exploiting the combinatorial representation, the non-skeleton component can be processed by using set index operations only, i.e., computing no linear combination at all.

Handling the special cases.

We now briefly discuss those portions of Pseudocode 2 that are meant to efficiently handle special cases. Note that, being just optimizations, these portions could be removed without compromising correctness.

Pseudocode 6 Processing a line violating constraint β.

```
    procedure VIOLATING_LINE(β, l, ⟨SK, NS⟩)
2:      split l into rays r+ satisfying β and r− violating β;
        l ← r+;
4:      for all g ∈ SK do
            g ← comb_β(g, l);
                                        ▷ now l ∈ SK+ and all other g ∈ SK0
6:      if is_equality(β) then
            SK ← SK0;
8:      else if is_strict_ineq(β) then
            STRICT_ON_EQ_POINTS(β, ⟨SK, NS⟩);
```

Pseudocode 7 Processing points saturating a strict inequality.

```
    procedure STRICT_ON_EQ_POINTS(β, ⟨SK, NS⟩)
2:      NS⋆ ← ∅;
        let SK0 = ⟨L0, R0, C0, SP0⟩;
4:      for all ns ∈ NS0 ∪ {{p} | p ∈ SP0} do
            NS⋆ ← NS⋆ ∪ ENUMERATE-FACES(β, ns, SK+, SK);
6:      SK0 ← points_become_closure_points(SK0);
        ⟨SK, NS⟩ ← ⟨SK+ ∪ SK0, NS+ ⊕ NS⋆⟩;
```

In lines 5 to 6 of 'SKEL_CONVERSION' we consider the case when constraint β is violated by a line. This special case is handled in procedure 'VIOLATING_LINE'

in Pseudocode 6. The pseudocode is similar to the corresponding special case for topologically closed polyhedra except that, when processing a strict inequality constraint, the helper procedure 'STRICT_ON_EQ_POINTS' gets called: this can be seen as a tailored version of the 'CREATE_NS' function, also including the final updating of $\mathcal{SK}$ and NS.

In lines 7 to 22 of 'SKEL_CONVERSION' we consider instead the cases when $\mathcal{SK}^+$ or $\mathcal{SK}^-$ (or both) are empty. Here we perform a few additional checks to see if an inconsistency has been detected, making the polyhedron empty and thereby allowing for an early exit from the overall conversion procedure. If this is not the case, we efficiently update the $\mathcal{SK}$ and NS components, possibly calling helper procedure 'STRICT_ON_EQ_POINTS'.

5.2. Processing a combinatorial constraint

After having processed all of the geometric constraints in $\mathcal{SK}^{\mathrm{c}}_{in}$, the 'CONVERSION' function in Pseudocode 1 continues by incrementally processing the combinatorial constraints in the non-skeleton component NS^{c}_{in}. In principle, it would be possible to *materialize* each support $ns^{\mathrm{c}} \in NS^{\mathrm{c}}_{in}$ to obtain a geometric constraint β and keep calling function 'SKEL_CONVERSION'. However, such an approach would incur an obvious overhead, which is avoided by calling function 'NONSKEL_CONVERSION' in Pseudocode 8.

Pseudocode 8 Adding a combinatorial constraint to a generator system.

```
    function NONSKEL_CONVERSION(ns^c, G)
2:      let G = ⟨SK, NS⟩;
        SK^- ← ∅; SK^0 ← sat(ns^c, SK); SK^+ ← SK \ SK^0;
4:      nonskel_partition(⟨SK, NS⟩);
        STRICT_ON_EQ_POINTS(ns^c, ⟨SK, NS⟩);
6:      return ⟨SK, NS⟩;
```

When processing the combinatorial constraint $ns^{\mathrm{c}} \in NS^{\mathrm{c}}$, we exploit the knowledge that all of the geometric constraints in the support have already been processed, so that ns^{c} represents a valid constraint for $\mathcal{P}$. Therefore, in lines 3 to 4 of Pseudocode 8 we know that ns^{c} always partitions the generators so that $\mathcal{SK}^- = NS^- = NS^{\pm} = \emptyset$ and we can avoid all the scalar products that would have been computed in the case of a geometric input. In particular, $\mathcal{SK}^0$ (and consequently NS^0) is easily obtained as the intersection of the generators saturating all the constraints in ns^{c}, namely, by reusing the saturation information computed before. As a consequence, the function can directly call the helper 'STRICT_ON_EQ_POINTS'.[11]

[11] The first parameter β of 'STRICT_ON_EQ_POINTS' in Pseudocode 7 seems to require a geometric constraint, but this is not really the case. The parameter is only used to check the constraint *kind* (equality, non-strict inequality or strict inequality): in the special case of a non-skeleton element ns^{c}, we always have a strict inequality.

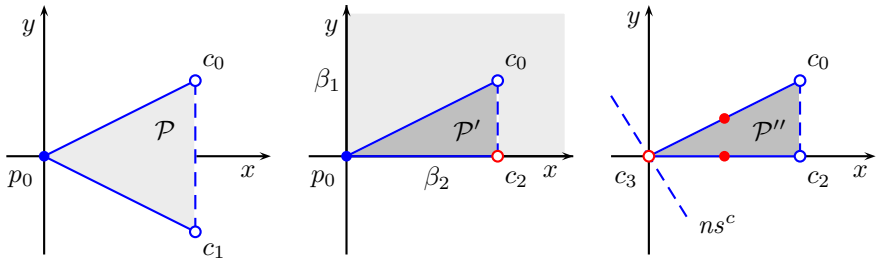

Figure 14: Incremental addition of geometric and combinatorial constraints to a polyhedron $\mathcal{P} \subseteq \mathbb{R}^2$.

Example 18. Consider the constraint system $\mathcal{C}_{in} = \langle \mathcal{SK}^{\mathrm{c}}_{in}, NS^{\mathrm{c}}_{in} \rangle$, with $\mathcal{SK}^{\mathrm{c}}_{in} = \{\beta_1, \beta_2\}$, $NS^{\mathrm{c}}_{in} = \{ns^{\mathrm{c}}\}$, $ns^{\mathrm{c}} = \{\beta_1, \beta_2\}$, where $\beta_1 \equiv (x \geq 0)$, $\beta_2 \equiv (y \geq 0)$. These constraints define the first quadrant of $\mathbb{R}^2$, without the origin. Consider now the polyhedron $\mathcal{P} \in \mathbb{P}_2$ shown on the left-hand side of Figure 14, defined by constraints $\{x - 2y \geq 0, x + 2y \geq 0, x < 2\}$ and described by $\mathcal{G} = \langle \mathcal{SK}^{\mathrm{g}}, NS^{\mathrm{g}} \rangle$, where $\mathcal{SK}^{\mathrm{g}} = \langle \emptyset, \emptyset, \{c_0, c_1\}, \{p_0\} \rangle$ and NS^{g} is empty. We want to add to $\mathcal{P}$ the constraints in $\mathcal{C}_{in}$, i.e., compute CONVERSION$(\mathcal{G}, \mathcal{C}_{in})$. As shown in Pseudocode 1, first we have to add to $\mathcal{G}$ the geometric constraints in $\mathcal{SK}^{\mathrm{c}}_{in}$, namely β_1 and β_2: the result $\mathcal{P}'$ of this intermediate step is shown in the middle of Figure 14. Note that constraint β_1 is redundant for $\mathcal{P}$, because it is satisfied by all the generators in $\mathcal{SK}^{\mathrm{g}}$, while with the insertion of β_2 closure point c_1 is removed and closure point c_2 is added to $\mathcal{SK}^{\mathrm{g}}$.

Then we process the combinatorial constraint $ns^c = \{\beta_1, \beta_2\}$: firstly, we exploit the saturation information already computed (line 3 of Pseudocode 8), finding that ns^c is saturated by

$$\mathrm{sat}(\{\beta_1, \beta_2\}, \mathcal{SK}^{\mathrm{g}}) = \mathrm{sat}(\beta_1, \mathcal{SK}^{\mathrm{g}}) \cap \mathrm{sat}(\beta_2, \mathcal{SK}^{\mathrm{g}}) = \{p_0\} \cap \{p_0, c_2\} = \{p_0\};$$

next, with function 'STRICT_ON_EQ_POINTS', point p_0 is substituted by the closure point c_3 and new non-skeleton points are created to fill the segments (c_3, c_0) and (c_3, c_2) with the usual step for the enumeration of faces. On the right-hand side of Figure 14 we show the resulting polyhedron $\mathcal{P}''$, for which the described conversion procedure has computed the generator system: $\mathcal{SK}^{\mathrm{g}} = \langle \emptyset, \emptyset, \{c_0, c_2, c_3\}, \emptyset \rangle$ and $NS^{\mathrm{g}} = \{\{c_0, c_3\}, \{c_2, c_3\}\}$.

As mentioned in Section 2, the constraint system of $\mathcal{P}''$ can be put in minimal form in a further simplification step: without entering in the details of this phase, we briefly show how the saturation information can be exploited once more. Firstly, constraint $(x + 2y \geq 0)$ is removed from the skeleton component since it is made redundant by the insertion of β_2, obtaining

$$\mathcal{SK}^{\mathrm{c}} = \langle C^{=}, C^{\geq}, C^{>} \rangle = \langle \emptyset, \{x - 2y \geq 0, y \geq 0\}, \{x < 2\} \rangle.$$

Note that, in order to correctly add the newly introduced non-skeleton constraint ns^{c} in NS^{c}, we have to recompute its support on the updated skeleton $\mathcal{SK}^{\mathrm{c}}$.

This can be done considering the closure of its old support; namely,

$$ns^{\mathrm{c}} \leftarrow \mathrm{supp\_cl}(ns^{\mathrm{c}}) = \mathrm{supp\_cl}(\{\beta_1, \beta_2\}) = \mathrm{sat}(\{p_0\}, \mathcal{SK}^{\mathrm{c}}) = \{x - 2y \geq 0, y \geq 0\}.$$

Example 18 shows that, at the implementation level, a little additional care has to be taken when processing the skeleton component: if a geometric constraint $\beta \in \mathcal{SK}^{\mathrm{c}}_{in}$ is detected to be redundant, it cannot be eagerly dropped, because it might occur in a support $ns \in NS^{\mathrm{c}}_{in}$ and therefore be needed to compute the corresponding saturation information; thus, the removal of β is delayed till completion of the whole procedure.

5.3. Duality: converting from generators to constraints

By building on the correspondences established in Section 4.6, it is possible to define a minor variant of the conversion procedure that maps a generator representation into a constraint representation.

One of the few differences, only occurring when performing a non-incremental conversion, is in the *initialization* phase preceding the actual call to the conversion procedure. When converting from constraints to generators, we start from a generator system representing the universe polyhedron (obtained by preprocessing the positivity constraint only). In contrast, when converting from generators to constraints, we look for a point in the input generator system: if such a point does not exists, the polyhedron is empty; otherwise, we preprocess it to obtain a skeleton constraint system being made of n linear equality constraints (plus the strict positivity constraint).

Another minor difference is in the handling of the special cases in lines 7 to 22 of SKEL_CONVERSION (as well as lines 5 and 6 of CONVERSION). When converting from generators to constraints, since we incrementally add new generators to a non-empty polyhedron, there is no way we can obtain an inconsistency, so that the checks corresponding to the comments '$\mathcal{P}$ is empty' can be omitted.

6. Operators on the New Representation

In principle, when adopting the new representation recalled in the previous section, each operator on the abstract domain of NNC polyhedra could be implemented indirectly, by first *materializing* the non-skeleton elements and then applying the operator on the fully geometric descriptions obtained. In this section we show that such a materialization step (and its computational overhead) is not really needed: all of the classical operators required for static analysis can be *directly* computed on the new representation, by distinguishing their effects on the geometric and the combinatorial components, also exploiting this division to simplify some of the procedures.

Emptiness, inclusion and equality.

$\mathcal{P} = \mathrm{gen}(\langle \mathcal{SK}^{\mathrm{g}}, NS^{\mathrm{g}} \rangle)$ is empty if and only if it has no point, i.e., $\mathcal{SK}^{\mathrm{g}}$ contains no point and $NS^{\mathrm{g}} = \emptyset$.

The inclusion $\mathcal{P}_1 \subseteq \mathcal{P}_2$ holds if and only if each generator of $\mathcal{P}_1$ satisfies all of the constraints of $\mathcal{P}_2$. Note that the lines, rays and closure points of $\mathcal{P}_1$ need to be checked only against the *skeleton* constraints of $\mathcal{P}_2$; only the points of $\mathcal{P}_1$ need to be checked against the non-skeleton strict inequalities of $\mathcal{P}_2$. Also, when checking a non-skeleton element, no additional scalar product needs to be computed: the result of the check is derived from the saturation information already computed (and cached) for skeleton elements. For instance, a skeleton point $\boldsymbol{p}_1 \in \mathcal{SK}_1^{\mathrm{g}}$ violates a non-skeleton constraint $ns_2 \in NS_2^{\mathrm{c}}$ when $\boldsymbol{p}_1 \in \mathrm{sat}(ns_2, \mathcal{SK}_1^{\mathrm{g}})$.

Equivalence $\mathcal{P}_1 = \mathcal{P}_2$ can be checked by performing two inclusion tests. Since the new representations satisfy a stronger form of normalization,[12] optimizations are possible: for instance, the test can be quickly answered negatively when the cardinalities of the minimized representations do not match.

Conditional and "forget".

A conditional test checking an affine predicate on program variables is modeled by adding the corresponding constraint to the polyhedron defining the program state. Similarly, a non-deterministic (or non-linear) assignment can be modeled by "forgetting" all the constraints mentioning the variable assigned to, i.e., by adding the corresponding line as a generator.[13] Hence, these two operators can be directly implemented by a call to the incremental 'CONVERSION' procedure presented in Section 5.

Meet and join.

When a conversion procedure is available, the computation of meets (i.e., set intersections) and joins (i.e., convex polyhedral hulls) on the domain of convex polyhedra is straightforward. Namely, if $\mathcal{P}_1 \equiv (\mathcal{C}_1, \mathcal{G}_1)$ and $\mathcal{P}_2 \equiv (\mathcal{C}_2, \mathcal{G}_2)$, then the DD pair for $\mathcal{P} = \mathcal{P}_1 \cap \mathcal{P}_2$ is obtained by incrementally adding to $(\mathcal{C}_1, \mathcal{G}_1)$ the constraints in $\mathcal{C}_2$; similarly, the DD pair for $\mathcal{P} = \mathcal{P}_1 \uplus \mathcal{P}_2$ is obtained by adding to $(\mathcal{C}_1, \mathcal{G}_1)$ the generators in $\mathcal{G}_2$.

Note that the incremental conversion described in Example 18 is in fact computing the intersection between the polyhedron $\mathcal{P}$ and the polyhedron defined by $\mathcal{C}_{in}$.

Assignment of an affine expression.

The assignment $x_i := \boldsymbol{a}^{\mathrm{T}}\boldsymbol{x} + b$ is modeled by computing the image of the polyhedron under the affine map $f\colon \mathbb{R}^n \to \mathbb{R}^n$, where $\boldsymbol{q} = f(\boldsymbol{p})$ is such that $q_i = \boldsymbol{a}^{\mathrm{T}}\boldsymbol{p} + b$ and $q_j = p_j$, when $i \neq j$. If f is *invertible* (i.e., $a_i \neq 0$), then the image and its inverse f^{-1} can be easily applied to the skeleton components of the generator and constraint representations, respectively; the non-skeleton components are not affected at all. If f is not invertible (i.e., $a_i = 0$), then it is

[12] It is meant, with respect to those available for ϵ-representations.

[13] Another application of the "forget" operator will be shown later, in Example 19 and Figure 15, when discussing the implementation of non-invertible affine assignments.

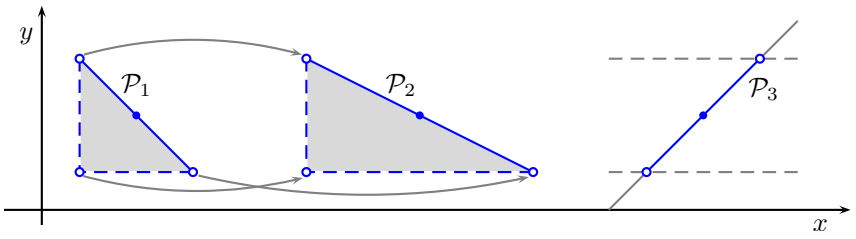

Figure 15: Application of affine images to polyhedra in $\mathbb{R}^2$.

computed by first "forgetting" the constraints on x_i, adding the corresponding line, and then adding constraint $\beta \equiv (x_i = \boldsymbol{a}^{\mathrm{T}}\boldsymbol{x} + b)$. In both cases, the minimal form of the input DD pair is *incrementally* maintained (i.e., there is no need to invoke the full conversion procedure).

Example 19. Consider polyhedron $\mathcal{P}_1 = \mathrm{con}(\mathcal{C}_1) \subseteq \mathbb{R}^2$ shown on the left-hand side of Figure 15, where $\mathcal{C}_1 = \{x > 1, y > 1, x + y \leq 5\}$; the corresponding generator system includes, in the skeleton component, the three closure points $C_1 = \{(1,1), (4,1), (1,4)\}$, as well as a single point in the non-skeleton component. Polyhedron $\mathcal{P}_2 = \mathrm{con}(\mathcal{C}_2)$ in the middle of the figure results from the application to $\mathcal{P}_1$ of the invertible affine map

$$f\left(\begin{pmatrix} x \\ y \end{pmatrix}\right) = \begin{pmatrix} 2x + 5 \\ y \end{pmatrix}.$$

Its constraint system $\mathcal{C}_2 = \{x > 7, y > 1, x + 2y \leq 15\}$ is obtained by applying the inverse map

$$f^{-1}\left(\begin{pmatrix} x \\ y \end{pmatrix}\right) = \begin{pmatrix} \frac{x-5}{2} \\ y \end{pmatrix}.$$

to the constraints in $\mathcal{C}_1$ so that, for instance, $x > 1$ is mapped to constraint $(\frac{x-5}{2} > 1) \equiv (x > 7)$. Its three closure points $C_2 = \{(7,1), (13,1), (7,4)\}$ are obtained by applying the direct map f to the closure points in C_1.

Thanks to the combinatorial representation, there is no need to apply the map to the non-skeleton point, since it simply inherits the transformation from its support. In general, the same observation holds for the non-skeleton constraints, which do not need to be translated using the inverse map (the considered example has no non-skeleton constraints). In contrast, an implementation using a completely geometric representation would require the mapping of the (materialized) non-skeleton points, incurring a corresponding overhead. Clearly, the overhead is even higher for implementations adopting the ϵ-representation approach, due to the inherent doubling of the number of points as well as the potential presence of ϵ-redundant constraints and generators.

Polyhedron $\mathcal{P}_3 = \mathrm{con}(\mathcal{C}_3)$, shown on the right-hand side of the figure, where $\mathcal{C}_3 = \{y > 1, y < 4, x = y - 15\}$, is obtained from $\mathcal{P}_2$ with the (non-invertible) affine map

$$g\left(\begin{pmatrix} x \\ y \end{pmatrix}\right) = \begin{pmatrix} y - 15 \\ y \end{pmatrix}.$$

Its computation is in two steps: we first forget all the constraints on x, adding line generator $\ell = (1, 0)$, thereby obtaining the infinite open stripe described by $\mathcal{C}'_3 = \{y > 1, y < 4\}$ (whose boundaries are outlined in gray on the right hand side of the figure); then, we add the new equality constraint $x = y - 15$. Both computational steps rely on the incremental conversion procedure.

6.1. A semantic widening for NNC polyhedra

The design of appropriate widening operators is considered both a key component and a main challenge in the development of abstract domains [8, 10, 25, 26, 29], in particular when targeting numerical properties [5, 6, 48], because their accuracy mostly depends on the particular context of application.

For ease of exposition, in the specifications of the widening operators we will only consider the case when the polyhedra arguments are non-empty, leaving implicit the common requirement that $\emptyset \nabla \mathcal{P}_2 = \mathcal{P}_2$. We start by considering the well known *standard widening* [39].

Definition 6 (Standard widening on $\mathbb{CP}_n$). Let $\mathcal{P}_1, \mathcal{P}_2 \in \mathbb{CP}_n$ be such that $\mathcal{P}_i = \mathrm{con}(\mathcal{C}_i)$, where $\mathcal{P}_1 \neq \emptyset$ and $\mathcal{C}_1$ is in minimal form. Let also $\mathcal{I}_i = \mathrm{ineqs}(\mathcal{C}_i)$;[14] then $\mathcal{P}_1 \nabla_{\mathrm{C}} \mathcal{P}_2 \doteq \mathrm{con}(\mathcal{I}'_1 \cup \mathcal{I}'_2)$, where

$$\mathcal{I}'_1 = \big\{\, \beta_1 \in \mathcal{I}_1 \;\big|\; \mathcal{P}_2 \subseteq \mathrm{con}(\{\beta_1\}) \,\big\},$$
$$\mathcal{I}'_2 = \big\{\, \beta_2 \in \mathcal{I}_2 \;\big|\; \exists \beta_1 \in \mathcal{I}_1 \,.\, \mathcal{P}_1 = \mathrm{con}(\mathcal{I}_1 \setminus \{\beta_1\} \cup \{\beta_2\}) \,\big\}.$$

When $\mathcal{P}_1 \subseteq \mathcal{P}_2$, the following is an equivalent specification (see [5, Theorem 5]), more appropriate for implementations based on the DD method.

Definition 7. Let $\mathcal{P}_1, \mathcal{P}_2 \in \mathbb{CP}_n$ be such that $\mathcal{P}_1 \equiv (\mathcal{C}_1, \mathcal{G}_1)$, $\mathcal{P}_2 = \mathrm{con}(\mathcal{C}_2)$, $\emptyset \neq \mathcal{P}_1 \subseteq \mathcal{P}_2$ and $\mathcal{C}_1$ is in minimal form. Then $\mathcal{P}_1 \nabla_{\mathrm{C}} \mathcal{P}_2 \doteq \mathrm{con}(\mathcal{C})$, where

$$\mathcal{C} = \big\{\, \beta_2 \in \mathcal{C}_2 \;\big|\; \exists \beta_1 \in \mathcal{C}_1 \,.\, \mathrm{sat}(\beta_1, \mathcal{G}_1) = \mathrm{sat}(\beta_2, \mathcal{G}_1) \,\big\}.$$

A "semantic" operator *op* on abstract domain $\mathcal{A}$ is a *well-defined* function $op \colon \mathcal{A}^k \to \mathcal{A}$; in contrast, a "syntactic" operator is not well-defined on $\mathcal{A}$: its result depends on the specific, non-canonical syntax adopted to represent the elements of $\mathcal{A}$ used as arguments (e.g., a constraint representation $\mathcal{C}$). More often than desired, the specification of widening operators relies on the syntactic representations of the abstract domain elements, rather than their semantics. This was the case for the original proposal of widening on closed polyhedra [30], which was refined into a semantic widening in [39]. Similarly, the widenings defined in [46] for the graph-based representations of bounded differences and octagons were refined into semantic widenings in [4, 13]. Available implementations of the domain of NNC polyhedra $\mathbb{P}_n$ based on the DD method are affected

[14] We write $\mathrm{ineqs}(\mathcal{C})$ to denote the constraint system obtained by splitting each equality in $\mathcal{C}$ into a pair of non-strict inequalities.

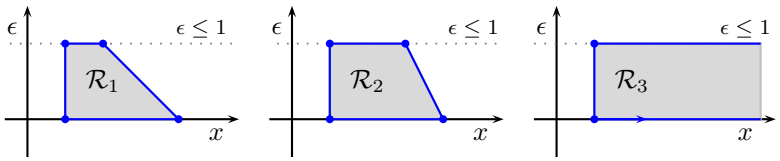

Figure 16: Widening NNC polyhedra delegating to widening on ϵ-representations.

by the same issue, because they compute the widening on the underlying ϵ-representations in $\mathbb{CP}_{n+1}$, which are not canonical (hence, syntactic). This happens for all of the widening variants defined on polyhedra, including the one proposed in [5, 6], as well as the improved versions that can be obtained by applying generic techniques, such as the *widening up-to* [41].

Example 20. Consider the ϵ-representation polyhedra in Figure 16. The two polyhedra $\mathcal{R}_1, \mathcal{R}_2 \in \mathbb{CP}_2$ on the left-hand side and the middle of the figure are encoding the same NNC polyhedron $\mathcal{P} = \mathrm{con}(\mathcal{C}) \in \mathbb{P}_1$, where $\mathcal{C} = \{1 \leq x, x < 4\}$. Both representations encode no redundant constraint; they only differ in the slope of the facet representing the strict inequality constraint. As a consequence, when computing $\mathcal{R}_3 = \mathcal{R}_1 \nabla_{\mathrm{C}} \mathcal{R}_2$, shown on the right-hand side of the figure, the standard widening operator on the ϵ-representations fails to detect the stability of the strict inequality constraint, which is dropped. $\mathcal{R}_3$ represents the NNC polyhedron $\mathcal{P}' = \mathrm{con}(\{1 \leq x\})$: even though correct from a theoretical point of view, the widening depends on the syntactic encoding of strict inequalities. As a side note, the user of the abstract domain might reasonably expect that a property such as $\mathcal{P} \nabla_{\mathrm{C}} \mathcal{P} = \mathcal{P}$ always holds, but this is not the case.

When implementing the widening on NNC polyhedra by delegating to the underlying widening on closed polyhedra, some precautions are required too:

- Definition 7 assumes that $\mathcal{P}_1 \subseteq \mathcal{P}_2$; note however that, for NNC polyhedra, $\mathcal{P}_1 \subseteq \mathcal{P}_2$ does not automatically imply that property $\mathcal{R}_1 \subseteq \mathcal{R}_2$ holds for the corresponding ϵ-representations;
- the implementation has to make sure that the result of the widening is still a valid ϵ-representation, i.e., the bounds for ϵ cannot be dropped;
- in order to ensure the finite convergence guarantee, the first argument $\mathcal{P}_1$ should be described by a constraint system encoding no redundant elements; however, a non-redundant description for the ϵ-representation $\mathcal{R}_1$ can still encode many redundant constraints; these have to be removed by applying the *strong minimization* procedures defined in [9, 14].

As a consequence, the overall approach may also incur a significant overhead.

In contrast, when using the direct encoding of Section 4, we can adopt a variant of Definition 7 to obtain a semantic widening on NNC polyhedra, because all of the materializations of a non-skeleton strict inequality constraint share the same saturation information, no matter for the variation in their slopes.

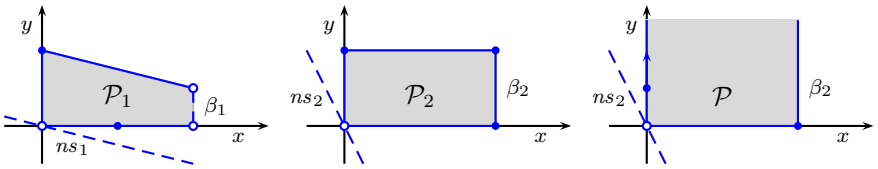

Figure 17: From left to right: $\mathcal{P}_1$, $\mathcal{P}_2$ and $\mathcal{P} = \mathcal{P}_1 \nabla_{\mathrm{N}} \mathcal{P}_2$.

Definition 8 (Widening on $\mathbb{P}_n$). Let $\mathcal{P}_1, \mathcal{P}_2 \in \mathbb{P}_n$ be such that $\mathcal{P}_1 \equiv (\mathcal{C}_1, \mathcal{G}_1)$, $\mathcal{P}_2 = \mathrm{con}(\mathcal{C}_2)$, $\emptyset \neq \mathcal{P}_1 \subseteq \mathcal{P}_2$, each $\mathcal{C}_i = \langle \mathcal{SK}_i^{\mathrm{c}}, NS_i^{\mathrm{c}} \rangle$ is in minimal form and $\mathcal{G}_1 = \langle \mathcal{SK}_1^{\mathrm{g}}, NS_1^{\mathrm{g}} \rangle$. Then $\mathcal{P}_1 \nabla_{\mathrm{N}} \mathcal{P}_2 \doteq \mathrm{con}(\langle \mathcal{SK}^{\mathrm{c}}, NS^{\mathrm{c}} \rangle)$, where

$$\mathcal{SK}^{\mathrm{c}} = \{\, \beta_2 \in \mathcal{SK}_2^{\mathrm{c}} \mid \exists \beta_1 \in \mathcal{SK}_1^{\mathrm{c}} \,.\, \mathrm{sat}(\beta_1, \mathcal{SK}_1^{\mathrm{g}}) = \mathrm{sat}(\beta_2, \mathcal{SK}_1^{\mathrm{g}}) \,\};$$
$$NS^{\mathrm{c}} = \{\, ns_2 \in NS_2^{\mathrm{c}} \mid ns_2 \subseteq \mathcal{SK}^{\mathrm{c}} \,\}.$$

The next proposition states that '∇_{N}' is a *semantic* widening and it is indeed an *extension* on the domain $\mathbb{P}_n$ of the standard widening '∇_{C}' defined on $\mathbb{CP}_n$.

Proposition 5. *Definition 8 specifies a well-defined widening operator on $\mathbb{P}_n$; moreover, for all $\mathcal{P}_1, \mathcal{P}_2 \in \mathbb{P}_n$ such that $\mathcal{P}_1 \subseteq \mathcal{P}_2$, we have*

$$\mathrm{cl}(\mathcal{P}_1) \nabla_{\mathrm{N}} \mathrm{cl}(\mathcal{P}_2) = \mathrm{cl}(\mathcal{P}_1) \nabla_{\mathrm{C}} \mathrm{cl}(\mathcal{P}_2). \tag{1}$$

The new widening satisfies both $\mathcal{P} \nabla_{\mathrm{N}} \mathcal{P} = \mathcal{P}$ and $\mathcal{P} \nabla_{\mathrm{N}} \mathrm{cl}(\mathcal{P}) = \mathrm{cl}(\mathcal{P})$, which is not the case for the widening based on the ϵ-dimension approach.

Example 21. Reconsider polyhedron $\mathcal{P} = \mathrm{con}(\{1 \leq x, x < 4\})$, for which a couple of possible ϵ-representations were shown in Figure 16. When directly encoding the strict inequalities and applying Definition 8, constraint $\beta \equiv (x < 4)$ is detected to be stable, so that $\mathcal{P} \nabla_{\mathrm{N}} \mathcal{P} = \mathcal{P}$. Moreover, letting $\beta' \equiv (x \leq 4)$, we also have $\mathcal{P} \nabla_{\mathrm{N}} \mathrm{cl}(\mathcal{P}) = \mathrm{cl}(\mathcal{P})$, because $\mathrm{sat}(\beta, \mathcal{SK}_1^{\mathrm{g}}) = \mathrm{sat}(\beta', \mathcal{SK}_1^{\mathrm{g}})$.

In Definition 8, the non-skeleton constraints and generators in NS_1^{c} and NS_1^{g} play no role in the computation of the widening, simplifying its implementation. As shown in Example 21, a *non-strict* inequality in $\beta_2 \in \mathcal{SK}_2^{\mathrm{c}}$ can be detected as stable (i.e., enter the result $\mathcal{SK}^{\mathrm{c}}$) even when it weakens a corresponding *strict* inequality in $\mathcal{SK}_1^{\mathrm{c}}$; this is not the case when blindly extending Definition 6. Also note that a non-skeleton constraint $ns \in NS^{\mathrm{c}}$ is stable only if it is supported by a set of skeleton constraints that are all stable.

Example 22. Consider $\mathcal{P}_1 = \mathrm{con}(\{0 \leq x < 4, 0 \leq y, 0 < x + 4y \leq 8\})$ and $\mathcal{P}_2 = \mathrm{con}(\{0 \leq x \leq 4, 0 \leq y \leq 2, 0 < 2x + y\})$, shown on the left and middle of Figure 17, respectively. Constraint $\beta_2 \equiv (x \leq 4)$ is stable, as it shares on $\mathcal{P}_1$ the same saturation information of $\beta_1 \equiv (x < 4)$. Support $ns_2 = \{x \geq 0, y \geq 0\} \equiv (0 < 2x + y)$ is stable, no matter if the shown materialization differs from the one chosen for $ns_1 \equiv (0 < x + 4y)$, because the skeleton constraints defining it are both stable. Thus, $\mathcal{P}_1 \nabla_{\mathrm{N}} \mathcal{P}_2 = \mathrm{con}(\{0 \leq x \leq 4, 0 \leq y, 0 < 2x + y\})$, shown

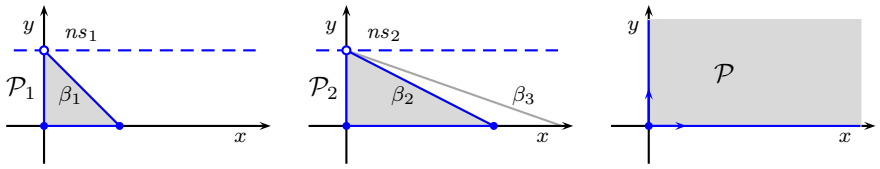

Figure 18: An increasing chain in $\mathbb{P}_2$ where Definition 7 is not stabilizing; $\mathcal{P} = \mathcal{P}_1 \nabla_{\mathrm{N}} \mathcal{P}_2$ is the result obtained when using Definition 8.

on the right-hand side of the figure. An implementation based on Definition 6 would drop β_2 and, depending on the chosen materializations, maybe also ns_2, thereby computing a less precise result.

In the next example we show that a blind extension of Definition 7 to the case of NNC polyhedra, where the non-skeleton component NS_2^{c} is treated the same as the skeleton component $\mathcal{SK}_2^{\mathrm{c}}$, would not result in a proper widening, since the finite convergence guarantee is compromised.

Example 23. For each $i \in \mathbb{N} \setminus \{0\}$, let $\beta_i \equiv (x + iy \leq i)$, $\mathcal{C}_i = \{0 \leq x, 0 \leq y < 1, \beta_i\}$ and $\mathcal{P}_i = \mathrm{con}(\mathcal{C}_i)$; note that $\mathcal{P}_i \subset \mathcal{P}_{i+1}$. Polyhedra $\mathcal{P}_1$ and $\mathcal{P}_2$ are shown on the left-hand side and middle of Figure 18. Note that $\mathcal{C}_i = \langle \mathcal{SK}_i^{\mathrm{c}}, NS_i^{\mathrm{c}} \rangle$, where $\mathcal{SK}_i^{\mathrm{c}} = \{0 \leq x, 0 \leq y, \beta_i\}$, $NS_i^{\mathrm{c}} = \{ns_i\}$ and $ns_i = \{0 \leq x, \beta_i\}$ (in the figure we show constraint $y < 1$ as one of the possible materializations of the supports ns_i). By using Definition 7 as is, we would obtain $\mathcal{P}_i \nabla_{\mathrm{C}} \mathcal{P}_{i+1} = \mathcal{P}_{i+1}$; namely, the skeleton constraint β_{i+1} and the non-skeleton constraint ns_{i+1} are detected to be stable, since in $\mathcal{P}_i$ they share the same saturation information of ns_i (they are only saturated by closure point $(0, 1)$). Hence, $\{\mathcal{P}_i\}_{i \in \mathbb{N}}$ would form an infinite increasing chain. In contrast, when using Definition 8 to compute $\mathcal{P}_1 \nabla_{\mathrm{N}} \mathcal{P}_2$, shown on the right of the figure, constraints β_2 and ns_2 are both dropped.

6.2. Improving the new widening operator

The precision of '∇_{N}' can be further enhanced leveraging on the general framework proposed in [5, 6]. This schema allows to obtain the convergence requirement by focusing on a class of preorders: a *limited growth order* (lgo) on a join semi-lattice $\langle L, \bot, \sqsubseteq, \sqcup \rangle$ is the strict version of any finitely computable preorder on L that satisfies the ascending chain condition; given a widening operator '∇' on L, a lgo $\curvearrowright \subseteq L \times L$ is ∇-*compatible* if $\forall x, y \in L : x \sqsubset y \Longrightarrow x \curvearrowright x \nabla y$. For any upper bound operator $h \colon L \times L \to L$, we can define a new widening at least as precise as '∇' [6, Theorem 9]: for all $x, y \in L$ such that $x \sqsubseteq y$,

$$x \tilde{\nabla} y \doteq \begin{cases} h(x, y), & \text{if } x \curvearrowright h(x, y) \sqsubset x \nabla y; \\ x \nabla y & \text{otherwise.} \end{cases}$$

In other words, by defining a compatible lgo it is possible to combine any finite set of upper bound operators, called heuristics, with an existing widening:

when an upper bound $h(x, y)$ happens to follow the (finitely) increasing chain defined by the lgo, we can choose this heuristic result instead of $x \nabla y$, preserving convergence and possibly improving precision.

By following this approach, we now apply the framework of [6] to the '∇_{N}' widening of NNC polyhedra, enhancing it using a similar set of heuristics. Since '∇_{N}' is a proper extension of '∇_{C}', the lgo relation defined in [6] on $\mathbb{CP}_n$ only needs a minor adaptation to be applicable to lattice $\mathbb{P}_n$.

For each finite multiset M over $\mathbb{N}$, let $\#(n, M)$ denote the number of occurrences of $n \in \mathbb{N}$ in M; then, the multiset partial order $M \sqsubseteq_{\mathrm{ms}} N$ holds when either $M = N$ or there exists $j \in \mathbb{N}$ such that $\#(j, M) > \#(j, N)$ and, for each $k \in \mathbb{N}$ with $k > j$, $\#(k, M) = \#(k, N)$. The relation '$\sqsubseteq_{\mathrm{ms}}$' satisfies the ascending chain condition.

Definition 9 ($\rightsquigarrow_{\mathrm{N}} \subseteq \mathbb{P}_n \times \mathbb{P}_n$). For $i = 1, 2$, let $\mathcal{P}_i = \mathrm{con}(\mathcal{C}_i) = \mathrm{gen}(\mathcal{G}_i)$ be non-empty polyhedra, where the systems are in minimal form and $\mathcal{C}_i = \langle \mathcal{SK}_i^{\mathrm{c}}, NS_i^{\mathrm{c}} \rangle$, $\mathcal{G}_i = \langle \mathcal{SK}_i^{\mathrm{g}}, NS_i^{\mathrm{g}} \rangle$, $\mathcal{SK}_i^{\mathrm{c}} = \langle C_i^{=}, C_i^{\geq}, C_i^{>} \rangle$ and $\mathcal{SK}_i^{\mathrm{g}} = \langle L_i, R_i, C_i, SP_i \rangle$.

Let $S_i \doteq \{ \{\beta\} \mid \beta \in C_i^{>} \} \cup NS_i^{\mathrm{c}}$ denote the supports of all (skeleton and non-skeleton) strict inequality constraints in $\mathcal{C}_i$ and consider the multiset $\zeta(S_i) \doteq \{ |\mathcal{SK}_i^{\mathrm{c}}| - |s| \in \mathbb{N} \mid s \in S_i \}$. Let also $\kappa(R_i)$ denote the multiset of the numbers of non-null coordinates of all the rays in R_i. The preorders $\preceq_d, \preceq_\ell, \preceq_c, \preceq_s, \preceq_p, \preceq_r \subseteq \mathbb{P}_n \times \mathbb{P}_n$ are defined respectively as the $\emptyset$-lifting of the following relations:

$$
\begin{array}{lcl}
\mathcal{P}_1 \preceq_d \mathcal{P}_2 & \iff & |C_1^{=}| \geq |C_2^{=}| \\
\mathcal{P}_1 \preceq_\ell \mathcal{P}_2 & \iff & |L_1| \leq |L_2| \\
\mathcal{P}_1 \preceq_c \mathcal{P}_2 & \iff & |\mathcal{SK}_1^{\mathrm{c}}| \geq |\mathcal{SK}_2^{\mathrm{c}}| \\
\mathcal{P}_1 \preceq_s \mathcal{P}_2 & \iff & \zeta(S_1) \sqsubseteq_{\mathrm{ms}} \zeta(S_2) \\
\mathcal{P}_1 \preceq_p \mathcal{P}_2 & \iff & |SP_1 \cup C_1| \geq |SP_2 \cup C_2| \\
\mathcal{P}_1 \preceq_r \mathcal{P}_2 & \iff & \kappa(R_1) \sqsubseteq_{\mathrm{ms}} \kappa(R_2)
\end{array}
$$

The relation $\rightsquigarrow_{\mathrm{N}} \subseteq \mathbb{P}_n \times \mathbb{P}_n$ is the strict version of the lexicographic product $\preceq_{\mathrm{N}} \doteq \preceq_{d\ell cspr} \subseteq \mathbb{P}_n \times \mathbb{P}_n$ of the six relations '$\preceq_d$', '$\preceq_\ell$', '$\preceq_c$', '$\preceq_s$', '$\preceq_p$' and '$\preceq_r$', taken in this order.

Note that, at the implementation level, when computing $\zeta(S_i)$ we have to exclude from S_i the redundant constraints introduced for technical reasons only, such as the positivity constraint or the cutter of the empty face.

Proposition 6. *The relation '$\rightsquigarrow_{\mathrm{N}}$' is a ∇_{N}-compatible lgo on $\mathbb{P}_n$.*

We now define a "bhrz03" improvement of the '∇_{N}' operator by leveraging on the same set of heuristics 'h_c', 'h_p' and 'h_r' ("combining constraints", "evolving (closure) points" and "evolving rays") proposed in [5, 6].

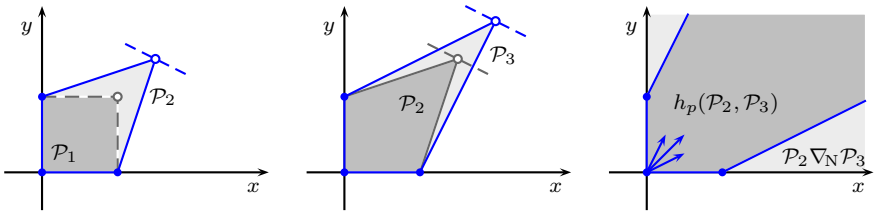

Figure 19: Improvements of the widening operator '∇_{N}' exploiting the ascending chain $\mathcal{P}_1 \curvearrowright_{\mathrm{N}} \mathcal{P}_2 \curvearrowright_{\mathrm{N}} h_p(\mathcal{P}_2, \mathcal{P}_3) \curvearrowright_{\mathrm{N}} \mathcal{P}_1 \nabla_{\mathrm{N}} \mathcal{P}_2 = \mathcal{P}_2 \nabla_{\mathrm{N}} \mathcal{P}_3$.

Definition 10 (The '$\tilde{\nabla}_{\mathrm{N}}$' widening.). Let $\mathcal{P}_1, \mathcal{P}_2 \in \mathbb{P}_n$ where $\emptyset \neq \mathcal{P}_1 \subset \mathcal{P}_2$. Then

$$\mathcal{P}_1 \tilde{\nabla}_{\mathrm{N}} \mathcal{P}_2 \doteq \begin{cases} \mathcal{P}_2 & \text{if } \mathcal{P}_1 \curvearrowright_{\mathrm{N}} \mathcal{P}_2 \\ h_c(\mathcal{P}_1, \mathcal{P}_2) & \text{if } \mathcal{P}_1 \curvearrowright_{\mathrm{N}} h_c(\mathcal{P}_1, \mathcal{P}_2) \subset \mathcal{P}_1 \nabla_{\mathrm{N}} \mathcal{P}_2 \\ h_p(\mathcal{P}_1, \mathcal{P}_2) & \text{if } \mathcal{P}_1 \curvearrowright_{\mathrm{N}} h_p(\mathcal{P}_1, \mathcal{P}_2) \subset \mathcal{P}_1 \nabla_{\mathrm{N}} \mathcal{P}_2 \\ h_r(\mathcal{P}_1, \mathcal{P}_2) & \text{if } \mathcal{P}_1 \curvearrowright_{\mathrm{N}} h_r(\mathcal{P}_1, \mathcal{P}_2) \subset \mathcal{P}_1 \nabla_{\mathrm{N}} \mathcal{P}_2 \\ \mathcal{P}_1 \nabla_{\mathrm{N}} \mathcal{P}_2 & \text{otherwise.} \end{cases}$$

Example 24. Consider the polyhedra $\mathcal{P}_1 \subset \mathcal{P}_2 \in \mathbb{P}_2$ on the left-hand side of Figure 19. Note that polyhedron $\mathcal{P}_1$ is defined by a skeleton constraint system $\mathcal{SK}_1^{\mathrm{c}}$ having 2 non-strict and 2 strict inequality constraints, whereas $\mathcal{P}_2$ has 4 non-strict inequalities in $\mathcal{SK}_2^{\mathrm{c}}$ and a non-skeleton strict inequality in NS_2^{c}. When computing the widening $\mathcal{P}_1 \tilde{\nabla}_{\mathrm{N}} \mathcal{P}_2$, we check if a heuristic upper bound happens to follow the lgo relation '$\curvearrowright_{\mathrm{N}}$': in particular, we note that $\mathcal{P}_1 \curvearrowright_{\mathrm{N}} \mathcal{P}_2$, because $\mathcal{P}_1 \equiv_{d\ell c} \mathcal{P}_2$ and $\mathcal{P}_1 \prec_s \mathcal{P}_2$. To see this, observe that the 2 skeleton strict constraints in S_1 (cutting two facets) both have a support of cardinality 1, whereas the only non-skeleton strict constraint in S_2 (cutting a 0-dimension face) has a support of cardinality 2; hence, $\zeta(S_1) = \{4-1, 4-1\} = \{3, 3\} \sqsubset_{\mathrm{ms}} \{2\} = \{4-2\} = \zeta(S_2)$. Thus we obtain $\mathcal{P}_1 \tilde{\nabla}_{\mathrm{N}} \mathcal{P}_2 = \mathcal{P}_2$, improving the precision of $\mathcal{P}_1 \nabla_{\mathrm{N}} \mathcal{P}_2 = \mathrm{con}(\{x \geq 0, y \geq 0\})$.

Consider now the polyhedra $\mathcal{P}_2 \subset \mathcal{P}_3 \in \mathbb{P}_2$ in the middle of Figure 19. In this case we have $\mathcal{P}_2 \equiv_{d\ell cspr} \mathcal{P}_3$, so that $\mathcal{P}_2 \not\curvearrowright_{\mathrm{N}} \mathcal{P}_3$. The upper bound $\mathcal{P}_4 = h_p(\mathcal{P}_2, \mathcal{P}_3)$, shown on the right-hand side of Figure 19, is obtained by applying the "evolving (closure) point" heuristics (i.e., the obvious extension of the "evolving point" heuristics defined in [5]). The heuristics detects that the closure point of $\mathcal{P}_3$ is "moving", defining evolving directions with respect to the generators of $\mathcal{P}_2$; namely, three rays are added to $\mathcal{P}_2$ (one of them being redundant), yielding polyhedron $\mathcal{P}_4$. The latter is defined by 4 skeleton non-strict inequalities and, being topologically closed, is such that $S_4 = \emptyset$. Therefore we obtain $\mathcal{P}_2 \curvearrowright_{\mathrm{N}} \mathcal{P}_4$, because $\mathcal{P}_2 \equiv_{d\ell c} \mathcal{P}_4$ and $\mathcal{P}_2 \prec_s \mathcal{P}_4$, as we have $\zeta(S_2) = \{2\} \sqsubset_{\mathrm{ms}} \emptyset = \zeta(S_4)$. Hence, we can use this heuristics and improve the precision of the widening: $\mathcal{P}_2 \tilde{\nabla}_{\mathrm{N}} \mathcal{P}_3 = \mathcal{P}_4 = h_p(\mathcal{P}_2, \mathcal{P}_3) \subset \mathcal{P}_2 \nabla_{\mathrm{N}} \mathcal{P}_3$.

7. Experimental Evaluation

The new representation for NNC polyhedra, the conversion algorithms and the operators presented in the previous sections are implemented in the PPLite library (`http://www.cs.unipr.it/~zaffanella/PPLite/`). Derived from the Parma Polyhedra Library, PPLite has a different goal: it provides a simpler framework for experimenting with new ideas and algorithms in the context of polyhedral computations. Its main characteristics are:

- both closed and NNC (rational) convex polyhedra are supported;
- exact computations are based on FLINT (`http://www.flintlib.org/`), which improves on GMP (`https://gmplib.org/`) by adopting a specialized representation for those integer values having small magnitude;
- encapsulation is not fully enforced: a knowledgeable user can directly change the contents of data structures, e.g., to experiment with alternative implementations of domain operators;
- preconditions on operators are not systematically checked at runtime: a user error typically leads to undefined behavior, rather than throwing an exception;
- performance and portability, even though deemed important, are not the main concern: ease of implementation and readability are given priority.

Evaluating the efficiency of the new conversion algorithm.

A preliminary experimental evaluation of the new representation and conversion algorithms for (closed and NNC) polyhedra was reported in [16], showing impressive efficiency gains with respect to the PPL. Those results were obtained when using a version of the algorithm implemented inside the PPL framework. Here we repeat those experiments using PPLite's implementation.[15]

The first experiment considers all the 98 test benchmarks distributed with the `ppl_lcdd` demo application of the PPL, which solves the *vertex/facet enumeration problem*. In Table 3 we report the timing results obtained on a selection of these tests when using: the PPL conversion algorithm for closed polyhedra (column 'cl'); the PPL conversion algorithm for the ϵ-representation of NNC polyhedra (column 'ϵ'); and the PPLite conversion algorithm for the new representation of NNC polyhedra (column 'new'). The tests shown in the table are those where the time in column 'cl' is above 0.02 seconds. The results show that the ϵ-representation incurs a significant overhead when compared to the closed representation (see the time ratio in column 'ϵ/cl'); this provides an evidence for the first main issue affecting ϵ-representations mentioned in Section 3. The

[15] All experiments have been performed on a laptop with an Intel Core i7-3632QM CPU, 16 GB of RAM and running GNU/Linux 4.13.0-36.

	PPL			PPLite	
test	cl	ϵ	ϵ/cl	new	cl/new
cross12.ine	0.16	0.22	1.38	0.14	1.14
in7.ine	0.04	0.14	3.50	0.03	1.33
kkd38_6.ine	0.46	1.72	3.74	0.19	2.42
kq20_11_m.ine	0.04	0.14	3.50	0.03	1.33
metric80_16.ine	0.04	0.07	1.75	0.01	4.00
mit31-20.ine	1.03	33.60	32.62	0.48	2.15
mp6.ine	0.08	0.20	2.50	0.11	0.73
reg600-5_m.ext	0.86	3.11	3.62	0.74	1.16
sampleh8.ine	5.14	38.00	7.39	2.76	1.86
trunc10.ine	1.18	4.88	4.14	0.01	118.00

Table 3: Conversion times (in seconds) and speedup ratios for closed polyhedra

results also confirm that such an overhead is absent when using the new representation adopted in PPLite: for all the tests, the PPLite version compares favorably with respect to the PPL implementation working on ϵ-representations and, more importantly, on most tests it even compares favorably with respect to the PPL implementation working on closed polyhedra (see the time ratio in column 'cl/new').

The second experiment is meant to evaluate the efficiency gains obtained when processing polyhedra that are not topologically closed. This is based on a PPL's synthetic benchmark which was originally proposed in [9], showing that an *enhanced* evaluation strategy (where a knowledgeable user carefully applies the strong minimization procedures to eliminate ϵ-redundancies) easily outperforms the *standard* evaluation strategy. In Table 4 we compare these two PPL scenarios with the PPLite implementation, which is similar to the *standard* evaluation approach since it requires no clever guess from the user of the library. The new representations and conversion procedures achieve impressive efficiency improvements (column 'time'). In the next two columns we report the number of non-incremental ('full') and incremental ('incr') calls to the conversion procedures: these show that the strong minimization procedures adopted in the enhanced computation strategy interfere with the incrementality of the computation, as discussed at the end of Section 3. Moreover, the comparisons of the total number of iterations of the conversion procedures ('# iter') and of the sizes of the intermediate results ('iter size') provide further evidence that the new approach is able to maintain a non-redundant description even *during* the iterations of a conversion.

Evaluating the efficiency of PPLite in a static analysis context.

In Table 5 we summarize the results of a more thorough experimental evaluation, where the PPLite implementation of the abstract domain of NNC polyhedra is used in a program analysis based on Abstract Interpretation. To this end, we have interfaced the PPLite library to the Apron library [43], so as to

repr, strategy	time	# conv		# iter	iter size	
		full	incr		median	max
ϵ-repr, standard	28.385	4	3	1142	3706	7259
ϵ-repr, enhanced	0.227	7	0	525	109	1661
new repr, standard	0.011	4	3	355	33	154

Table 4: Comparing ϵ-representation based (standard and enhanced) computations for NNC polyhedra with the new conversion procedures.

make it available to PAGAI [42], a static analyzer for invariant generation built on top of the LLVM infrastructure. PAGAI is a highly configurable tool: besides selecting a specific abstract domain among those made available by the Apron library, the user can also choose among different static analysis techniques, such as *path focusing* [42], which improves the precision by exploiting SMT model-checking techniques. Since the more sophisticated techniques add a corresponding overhead which is not directly related to the abstract domain implementation, in our experimental evaluation we opted for the *simple abstract interpretation* analysis technique. Moreover, to better focus on the efficiency of the abstract domains themselves, we report the time spent during the calls to the abstract operators made available via Apron (rather than the total time of the analysis, which would also include, for instance, the time spent in the LLVM framework to implement bitcode transformations such as function inlining).

The experimental evaluation considers 38 C source files distributed with PAGAI: most of these are variants of benchmarks taken from the SNU real-time benchmark suite for worst-case execution time analysis. Note that we show the results for a selection of the tests, those for which the time reported in column 'pplite' is above 0.5 seconds; column 'size' shows the size of the corresponding LLVM bitcode file. When using PAGAI, it is possible to choose between several abstract domains. In columns 'box' and 'oct' we report the times obtained when selecting the abstract domain of boxes [27] and octagons [46], respectively. The next three columns report the times obtained when selecting different implementations of the domain of NNC polyhedra: the domain which is natively distributed with Apron (column 'pk'), the Apron layer for the Parma Polyhedra Library domain (column 'ppl') and the newly developed Apron layer for the PPLite domain (column 'pplite').

All domains use the default implementation of the widening operator, which in the case of the polyhedra domains is the standard widening [39]. For the domains 'pk' and 'ppl', the standard widening applies the specification in Definition 6 (equivalently, Definition 7) to the ϵ-representations of the NNC polyhedra: since these representations are not canonical, the result of the widening could be nondeterministic. For the domain 'pplite', the standard widening applies the specification in Definition 8 to the new representation of the NNC polyhedra, so that the result is deterministic (as stated in Proposition 5).

It can be seen that the PPLite library performs significantly better than the other NNC polyhedra domains, being also competitive with respect to the do-

	size	Apron's time (seconds)					
test	KB	box	oct	pk	ppl	pplite	$\mathbb{U}$
decompress	549	7.01	41.88	203.73	93.07	38.61	10.31
filter	15	1.18	5.90	77.89	83.85	18.96	13.70
adpcm	67	0.75	3.17	20.63	13.27	4.76	1.82
decompress-opt	71	0.60	9.99	12.90	7.40	2.80	1.56
nsichneu	527	0.62	0.64	2.32	2.99	1.60	1.45
cover	33	0.42	0.45	1.56	2.12	1.22	1.14
fft1	20	0.19	0.53	1.99	1.67	0.76	0.58
compress	30	0.16	0.69	2.57	1.72	0.73	0.37
ndes	45	0.21	0.30	1.17	1.30	0.72	0.54
edn	57	0.20	0.35	1.64	1.48	0.72	0.66
minver	30	0.14	0.27	1.03	1.01	0.51	0.42

Table 5: Efficiency comparison for PAGAI's domains.

main of octagons on the biggest benchmarks. These efficiency gains have been obtained even if the only use of strict inequalities in PAGAI is when modeling floating point values: among the tests reported in Table 5, only 'fft1' and 'minver' declare floating point variables.

It should be stressed that the static analysis implemented in PAGAI applies no variable packing technique at all: hence, all the relational domains incur avoidable overheads [51]. In principle, these overheads are orthogonal with respect to the chosen implementation of NNC polyhedra. In order to test this hypothesis, we enriched our experimental evaluation by also considering the '$\mathbb{U}$' *wrapper* proposed in [55], which provides a restricted form of Cartesian factoring. The '$\mathbb{U}$' wrapper enhances the underlying abstract domain by keeping track of the *unconstrained* space dimensions (which are dynamically recorded in the "shell" of the wrapper), therefore avoiding the need to represent them in the "kernel" (i.e., the wrapped polyhedron). In addition to reducing the size of the polyhedra, a fast access to the unconstrained dimensions allows to optimize the implementation of some abstract operators, such as the convex polyhedral hull and the invertible affine images. The results obtained when applying the '$\mathbb{U}$' wrapper to the NNC polyhedra of PPLite (column '$\mathbb{U}$' in Table 5) show that significant efficiency improvements are possible. We conjecture that the PPLite domain could equally benefit from the adoption of more sophisticated variable packing techniques.

Evaluating the correctness and precision of the implementation.

PAGAI can be configured to perform a precision comparison among two different abstract domains: in this case, the analyzer records the invariant properties computed by the two domains for each of the relevant program points; then it compares them and provides a final report made of four numbers, counting the program points on which the invariant computed by the first domain is, respectively, *equivalent*, *stronger*, *weaker* and *uncomparable* with respect to the invariant computed by the second domain. When using this precision comparison mode on the experimental evaluation discussed in the previous section, the

polyhedra domains 'pk', 'ppl', 'pplite' and '$\mathbb{U}$' are shown to compute equivalent invariants for all benchmarks and all program points, thereby providing a first assessment of the correctness and precision of the integration of PPLite with Apron and PAGAI.

In order to further increase the confidence on the correctness and precision comparison, we developed in PAGAI a new static analysis technique where, after each and every invocation of an abstract operator, the result computed by the PPLite domain is systematically compared with the result computed by PPL: this allows to highlight all differences, even those only occurring on the intermediate values computed *during* the analysis process, which cannot be detected by the previous approach. When using this new analysis technique, no difference is observed on all the abstract operators used in PAGAI, with the only exception of the widening operator, for which a very limited number of differences are detected on just a few tests. Note that, on the considered benchmarks, these precision differences affect neither the number of iterations of the analysis process nor the final invariant property computed, which are exactly the same for all the considered polyhedra domains.

A further investigation on the few differences on widenings shows that the standard widening '∇_{N}' of PPLite is sometimes more precise than the standard widening of PPL (which is computed on the ϵ-representation). Moreover, the same experiments show that, when using the ϵ-representations of PPL and its corresponding syntactic widening operator, the PAGAI analyzer indeed exhibits a nondeterministic behavior, computing different (correct) results at each run; such a nondeterminism of the analysis is not observable when using the PPLite domain, which is based on the semantic widening '∇_{N}'. This provides a practical evidence for the theoretical observations made in Section 6.1.

The PPLite library also borrows from the PPL library a wide set of synthetic benchmarks, which are systematically used for regression testing: once again, the abstract operators for NNC polyhedra implemented in PPLite obtain the same results as PPL on all tests (modulo the already observed increase in precision for the widening operator).

Evaluating the improved widening operator.

Table 6 shows a comparison of the efficiency of the PPLite polyhedra domain when using the standard widening presented in Section 6.1 and the improved widening presented in Section 6.2. In the column labeled 'h79/bhrz03' we report the time ratio of the two analyses: values above 1.0 mean that, for the considered test, the analysis using the improved widening operator 'bhrz03' is faster than the analysis using the standard widening 'h79'. Note that the absolute time for the 'h79' analysis has already been reported in column 'pplite' of Table 5.[16]

The results suggest that the computation of the lgo relation on the NNC

[16] We do not report a comparison for 'filter', as in this test the analysis never applies the widening operator; apparently, the code of the test has been obtained by completely unrolling sixty iterations of a given loop.

	time ratio	∇ calls		asc iters		desc iters	
test	h79/bhrz03	h79	bhrz03	h79	bhrz03	h79	bhrz03
adpcm	1.99	126	127	4744	5056	20148	1679
compress	3.65	33	33	1053	1051	6589	1198
cover	4.31	6	6	2398	2280	32316	5386
decompress	1.74	149	176	13032	14409	135528	67764
decompress-opt	1.00	180	218	2507	2714	17728	17728
edn	3.55	36	40	2047	2620	9696	808
fft1	0.92	126	156	1681	1967	8652	8652
minver	2.99	59	62	1402	1724	7092	591
ndes	4.03	34	34	1795	2076	10860	905
nsichneu	1.91	2	2	13230	13230	39681	13227

Table 6: Comparing the efficiency of the improved widening operator.

polyhedra in PPLite incurs no significant overhead; rather, the adoption of a more precise widening operator quite often results in an efficiency gain. Note that similar time ratios are obtained when comparing the 'h79' and the 'bhrz03' widenings on the ϵ-representation of the PPL implementation.

The remaining pairs of columns in Table 6 provide a justification for the observed speedup. In the pair of columns labeled '∇ calls' we report the total number of calls to the widening operator: this increases when using the 'bhrz03' widening. In the next two pairs of columns (labeled 'asc iters' and 'desc iters') we show the number of *node iterations* reported by PAGAI, i.e., the number of times a node (i.e., a basic block of LLVM bitcode) is processed by the analyzer. These are divided into the ascending and the descending iteration phases: in the ascending phase the analyzer is still trying to over-approximate the fixpoint computation (i.e., computing a post-fixpoint), possibly using the widening operator to enforce convergence; in the descending phase, having already computed a post-fixpoint, the analyzer is trying to improve its precision possibly using a narrowing operator. When using the 'bhrz03' widening we observe an increase in the number of ascending iterations, which intuitively matches the increased number of calls of the widening operator. As a consequence of the improved precision obtained by using the 'bhrz03' widening, the descending phase stabilizes more quickly: for most of the benchmarks, we record a significant reduction in column 'desc iters'. Hence, the speedup in the descending iteration phase masks the slowdown, if any, incurred during the ascending iteration phase.

Regarding precision, the PPLite and PPL polyhedra domains obtain the same results on all benchmarks when both using the 'bhrz03' widening operator (a comparison with domain 'pk' is not possible, as it does not implement the improved widening operator). Finally, we briefly summarize the precision comparison results obtained when replacing the 'h79' with the 'bhrz03' widening operator (no matter if using PPLite or PPL). No difference was recorded for 21 of the 38 benchmarks; for the remaining 17 benchmarks all differences are improvements (i.e., the new widening never causes a degradation in precision). When focusing on these 17 tests, a precision improvement is observed on 60% (97 on 161) of the invariant properties computed by the analyzer (one

invariant for each widening point). A more thorough analysis of the precision improvements of 'bhrz03' is beyond the scope of this paper: for all what we said before, it would just confirm the observations already made in previous work based on the PPL implementation. Hence, we refer the interested reader to the evaluations presented in [6, Table 1] and [42, Table 4].

8. Conclusion

We have proposed a new approach for the representation of NNC polyhedra in the Double Description framework, dispensing with the use of slack variables and distinguishing between a geometric and a combinatorial component. We have defined a corresponding variant of the Chernikova conversion algorithm, which exploits the peculiarities of the new representation to avoid the overheads incurred by the more classical approaches. Based on these new proposals, we have then presented an implementation of the abstract domain of NNC polyhedra, focusing our work on the specification of the operators needed for defining static analyses based on Abstract Interpretation; in particular, we have provided new, semantics-based widening operators for NNC polyhedra. The experimental evaluation conducted shows that the new representation allows for efficiently computing on the domain of NNC polyhedra; moreover, no overhead is observed when the computed polyhedra happen to be topologically closed, thereby systematically outperforming more classical implementations based on the addition of a slack variable.

In [18] the implementation described in this paper has been further extended providing support for semantic operators needed in different application contexts (such as the *time elapse* operator for the analysis and verification of hybrid systems [35]) as well as optimizations and new specifications for more general purpose functions (such as the *constraint hull*, the *parallel affine image* and the *split* operators), once again achieving noteworthy efficiency improvements.

References

[1] 4ti2 team, 2018. 4ti2—a software package for algebraic, geometric and combinatorial problems on linear spaces. Available at www.4ti2.de.

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

[3] Assarf, B., Gawrilow, E., Herr, K., Joswig, M., Lorenz, B., Paffenholz, A., Rehn, T., 2017. Computing convex hulls and counting integer points with polymake. Math. Program. Comput. 9 (1), 1–38.

[4] Bagnara, R., Hill, P. M., Mazzi, E., Zaffanella, E., 2005. Widening operators for weakly-relational numeric abstractions. In: Hankin, C., Siveroni, I. (Eds.), Static Analysis: Proceedings of the 12th International Symposium.

Vol. 3672 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, London, UK, pp. 3–18.

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

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

[7] Bagnara, R., Hill, P. M., Zaffanella, E., 2002. A new encoding of not necessarily closed convex polyhedra. In: Carro, M., Vacheret, C., Lau, K.-K. (Eds.), Proceedings of the 1st CoLogNet Workshop on Component-based Software Development and Implementation Technology for Computational Logic Systems. Madrid, Spain, pp. 147–153, published as TR Number CLIP4/02.0, Universidad Politécnica de Madrid, Facultad de Informática.

[8] Bagnara, R., Hill, P. M., Zaffanella, E., 2003. Widening operators for powerset domains. In: Steffen, B., Levi, G. (Eds.), Verification, Model Checking and Abstract Interpretation: Proceedings of the 5th International Conference (VMCAI 2004). Vol. 2937 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, Venice, Italy, pp. 135–148.

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

[10] Bagnara, R., Hill, P. M., Zaffanella, E., 2006. Widening operators for powerset domains. Software Tools for Technology Transfer 8 (4/5), 449–466.

[11] Bagnara, R., Hill, P. M., Zaffanella, E., 2008. 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 (1–2), 3–21.

[12] Bagnara, R., Hill, P. M., Zaffanella, E., 2009. Applications of polyhedral computations to the analysis and verification of hardware and software systems. Theoretical Computer Science 410 (46), 4672–4691.

[13] Bagnara, R., Hill, P. M., Zaffanella, E., 2009. Weakly-relational shapes for numeric abstractions: Improved algorithms and proofs of correctness. Formal Methods in System Design 35 (3), 279–323.

[14] Bagnara, R., Ricci, E., Zaffanella, E., Hill, P. M., 2002. Possibly not closed convex polyhedra and the Parma Polyhedra Library. In: Hermenegildo,

M. V., Puebla, G. (Eds.), Static Analysis: Proceedings of the 9th International Symposium. Vol. 2477 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, Madrid, Spain, pp. 213–229.

[15] Bastoul, C., 2004. Code generation in the polyhedral model is easier than you think. In: Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques (PACT 2004). IEEE Computer Society, Antibes Juan-les-Pins, France, pp. 7–16.

[16] Becchi, A., Zaffanella, E., 2018. A direct encoding for NNC polyhedra. In: Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I. pp. 230–248.

[17] Becchi, A., Zaffanella, E., 2018. An efficient abstract domain for not necessarily closed polyhedra. In: Podelski, A. (Ed.), Static Analysis - 25th International Symposium, SAS 2018, Freiburg, Germany, August 29-31, 2018, Proceedings. Vol. 11002 of Lecture Notes in Computer Science. Springer, pp. 146–165.

[18] Becchi, A., Zaffanella, E., 2019. Revisiting polyhedral analysis for hybrid systems. In: Chang, B. E. (Ed.), Static Analysis - 26th International Symposium, SAS 2019, Porto, Portugal, October 8-11, 2019, Proceedings. Vol. 11822 of Lecture Notes in Computer Science. Springer, pp. 183–202.

[19] Benerecetti, M., Faella, M., Minopoli, S., 2013. Automatic synthesis of switching controllers for linear hybrid systems: Safety control. Theor. Comput. Sci. 493, 116–138.

[20] Birkhoff, G., 1967. Lattice Theory, 3rd Edition. Vol. XXV of Colloquium Publications. American Mathematical Society, Providence, Rhode Island, USA.

[21] Chernikova, N. V., 1964. 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.

[22] Chernikova, N. V., 1965. 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.

[23] Chernikova, N. V., 1968. 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.

[24] Colón, M. A., Sipma, H. B., 2001. Synthesis of linear ranking functions. In: Margaria, T., Yi, W. (Eds.), Tools and Algorithms for Construction and Analysis of Systems, 7th International Conference, TACAS 2001. Vol. 2031 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, Genova, Italy, pp. 67–81.

[25] Cortesi, A., 2008. Widening operators for abstract interpretation. In: Sixth IEEE International Conference on Software Engineering and Formal Methods, SEFM 2008, Cape Town, South Africa. pp. 31–40.

[26] Cortesi, A., Zanioli, M., 2011. Widening and narrowing operators for abstract interpretation. Computer Languages, Systems & Structures 37 (1), 24–42.

[27] Cousot, P., Cousot, R., 1977. 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. ACM Press, Los Angeles, CA, USA, pp. 238–252.

[28] Cousot, P., Cousot, R., 1979. Systematic design of program analysis frameworks. In: Proceedings of the Sixth Annual ACM Symposium on Principles of Programming Languages. ACM Press, San Antonio, TX, USA, pp. 269–282.

[29] Cousot, P., Cousot, R., 1992. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation. In: Bruynooghe, M., Wirsing, M. (Eds.), Proceedings of the 4th International Symposium on Programming Language Implementation and Logic Programming. Vol. 631 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, Leuven, Belgium, pp. 269–295.

[30] Cousot, P., Halbwachs, N., 1978. Automatic discovery of linear restraints among variables of a program. In: Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages. ACM Press, Tucson, Arizona, pp. 84–96.

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

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

[33] Fehnker, A., Ivancic, F., 2004. Benchmarks for hybrid systems verification. In: Alur, R., Pappas, G. (Eds.), Hybrid Systems: Computation and Control, 7th International Workshop, HSCC 2004, Philadelphia, PA, USA, March 25-27, 2004, Proceedings. Vol. 2993 of Lecture Notes in Computer Science. Springer, pp. 326–341.

[34] Frehse, G., 2008. PHAVer: Algorithmic verification of hybrid systems past HyTech. Software Tools for Technology Transfer 10 (3), 263–279.

[35] Frehse, G., Abate, A., Adzkiya, D., Becchi, A., Bu, L., Cimatti, A., Giacobbe, M., Griggio, A., Mover, S., Mufid, M., Riouak, I., Tonetta, S., Zaffanella, E., 2019. ARCH-COMP19 category report: Hybrid systems with piecewise constant dynamics. In: Frehse, G., Althoff, M. (Eds.), ARCH19. 6th International Workshop on Applied Verification of Continuous and Hybrid Systems. Vol. 61 of EPiC Series in Computing. EasyChair, pp. 1–13.

[36] Fukuda, K., Prodon, A., 1996. Double description method revisited. In: Deza, M., Euler, R., Manoussakis, Y. (Eds.), Combinatorics and Computer Science, 8th Franco-Japanese and 4th Franco-Chinese Conference, Brest, France, July 3-5, 1995, Selected Papers. Vol. 1120 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, pp. 91–111.

[37] Genov, B., 2014. The convex hull problem in practice: Improving the running time of the double description method. Ph.D. thesis, University of Bremen, Germany.

[38] Gopan, D., Aug. 2007. Numeric program analysis techniques with applications to array analysis and library summarization. Ph.D. thesis, University of Wisconsin, Madison, Wisconsin, USA.

[39] Halbwachs, N., Mar. 1979. 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.

[40] Halbwachs, N., Proy, Y.-E., Raymond, P., 1994. Verification of linear hybrid systems by means of convex approximations. In: Le Charlier, B. (Ed.), Static Analysis: Proceedings of the 1st International Symposium. Vol. 864 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, Namur, Belgium, pp. 223–237.

[41] Halbwachs, N., Proy, Y.-E., Roumanoff, P., 1997. Verification of real-time systems using linear relation analysis. Formal Methods in System Design 11 (2), 157–185.

[42] Henry, J., Monniaux, D., Moy, M., 2012. PAGAI: A path sensitive static analyser. Electr. Notes Theor. Comput. Sci. 289, 15–25.

[43] Jeannet, B., Miné, A., 2009. Apron: A library of numerical abstract domains for static analysis. In: Bouajjani, A., Maler, O. (Eds.), Computer Aided Verification, Proceedings of the 21st International Conference (CAV 2009). Vol. 5643 of Lecture Notes in Computer Science. Springer, Grenoble, France, pp. 661–667.

[44] Kaibel, V., Pfetsch, M. E., 2002. Computing the face lattice of a polytope from its vertex-facet incidences. Computational Geometry 23 (3), 281–290.

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

[46] Miné, A., Mar. 2005. Weakly relational numerical abstract domains. Ph.D. thesis, École Polytechnique, Paris, France.

[47] Motzkin, T. S., Raiffa, H., Thompson, G. L., Thrall, R. M., 1953. The double description method. In: Kuhn, H. W., Tucker, A. W. (Eds.), Contributions to the Theory of Games – Volume II. No. 28 in Annals of Mathematics Studies. Princeton University Press, Princeton, New Jersey, pp. 51–73.

[48] Notani, V., Giacobazzi, R., 2017. Learning based widening. Contribution to the 8th Workshop on Tools for Automatic Program Analysis (TAPAS'17), New York.

[49] Perri, S., December 2012. 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, in Italian.

[50] Pop, S., Silber, G.-A., Cohen, A., Bastoul, C., Girbal, S., Vasilache, N., 2006. GRAPHITE: Polyhedral analyses and optimizations for GCC. Tech. Rep. A/378/CRI, Centre de Recherche en Informatique, École des Mines de Paris, Fontainebleau, France, contribution to the GNU Compilers Collection Developers Summit 2006 (GCC Summit 06), Ottawa, Canada, June 28–30, 2006.

[51] Singh, G., Püschel, M., Vechev, M. T., 2017. Fast polyhedra abstract domain. In: Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017. pp. 46–59.

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

[53] Terzer, M., Stelling, J., 2008. Large-scale computation of elementary flux modes with bit pattern trees. Bioinformatics 24 (19), 2229–2235.

[54] Terzer, M., Stelling, J., 2009. Parallel extreme ray and pathway computation. In: Parallel Processing and Applied Mathematics, 8th International Conference, PPAM 2009, Wroclaw, Poland, 2009, Revised Selected Papers, Part II. pp. 300–309.

[55] Zaffanella, E., 2018. On the efficiency of convex polyhedra. Electr. Notes Theor. Comput. Sci. 334, 31–44.

[56] Zolotykh, N. Y., 2012. New modification of the double description method for constructing the skeleton of a polyhedral cone. Computational Mathematics and Mathematical Physics 52 (1), 146–156.

Appendix A. Proofs of the Results Stated in the Paper

Appendix A.1. Proofs of results stated in Section 4

Proof of Proposition 1. Let $\mathcal{G} = \langle L, R, C, P\rangle$ and consider a generator system $\mathcal{G}_m = \langle L_m, R_m, C_m, P_m\rangle$ in minimal form such that $\mathrm{gen}(\mathcal{G}_m) = \mathrm{gen}(\mathcal{G}) = \mathcal{P}$. By Definition 1, we obtain $\mathcal{SK}_\mathcal{Q} = \langle L_m, R_m, C_m \cup SP_m, \emptyset\rangle$, where $SP_m \subseteq P_m$ is the set of skeleton points in P_m. Since each point $\boldsymbol{p} \in P_m \setminus SP_m$ can be obtained by a combination of the generators in L_m, R_m and $C_m \cup SP_m$, we have the following chain of equivalences:

$$\begin{aligned}\overline{\mathrm{gen}}(\mathcal{G}) &= \overline{\mathrm{gen}}(\mathcal{G}_m)\\ &= \mathrm{gen}\big(\langle L_m, R_m, \emptyset, C_m \cup P_m\rangle\big)\\ &= \mathrm{gen}\big(\langle L_m, R_m, \emptyset, C_m \cup SP_m\rangle\big)\\ &= \overline{\mathrm{gen}}\big(\langle L_m, R_m, C_m \cup SP_m, \emptyset\rangle\big)\\ &= \overline{\mathrm{gen}}(\mathcal{SK}_\mathcal{Q}).\end{aligned}$$

Since function '$\overline{\mathrm{gen}}$' interprets closure points as points, it computes a topologically closed polyhedron, so that $\overline{\mathrm{gen}}(\mathcal{SK}_\mathcal{Q}) = \mathcal{Q} = \mathrm{cl}(\mathcal{P})$. Moreover, since $\mathcal{SK}_\mathcal{Q}$ has been built from the generator system $\mathcal{G}_m$ in minimal form, by construction it only keeps in $C_m \cup SP_m$ the non-redundant points of $\mathcal{Q} = \mathrm{cl}(\mathcal{P})$; hence, it is the minimal system such that $\overline{\mathrm{gen}}(\mathcal{SK}_\mathcal{Q}) = \mathcal{Q}$. □

Proof of Proposition 2. Let $\mathcal{SK}_F = \langle L_F, R_F, C_F, \emptyset\rangle \subseteq \mathcal{SK}_\mathcal{Q}$ be the skeleton of the face $F \subseteq \mathcal{Q}$, so that $\overline{\mathrm{gen}}(\mathcal{SK}_F) = F$. By definition of '$\overline{\mathrm{gen}}$', the points $\boldsymbol{p}, \boldsymbol{p}' \in \mathrm{relint}(F)$ can be obtained by combining the generators in $\mathcal{SK}_F$:

$$\begin{aligned}\boldsymbol{p} &= L_F\boldsymbol{\lambda} + R_F\boldsymbol{\rho} + C_F\boldsymbol{\gamma},\\ \boldsymbol{p}' &= L_F\boldsymbol{\lambda}' + R_F\boldsymbol{\rho}' + C_F\boldsymbol{\gamma}',\end{aligned}$$

where $\boldsymbol{\lambda}, \boldsymbol{\lambda}' \in \mathbb{R}^\ell$, $\boldsymbol{\rho}, \boldsymbol{\rho}' \in \mathbb{R}^r_+$, for each $i \in \{1, \ldots, r\}$, $\rho_i, \rho'_i > 0$, $\boldsymbol{\gamma}, \boldsymbol{\gamma}' \in \mathbb{R}^c_+$, $\sum_{i=1}^c \gamma_i = \sum_{i=1}^c \gamma'_i = 1$ and, for each $i \in \{1, \ldots, c\}$, both $\gamma_i, \gamma'_i > 0$. Therefore,

$$\mathrm{relint}(F) = \mathrm{gen}\big(\langle L_F, R_F, C_F, \{\boldsymbol{p}\}\rangle\big) = \mathrm{gen}\big(\langle L_F, R_F, C_F, \{\boldsymbol{p}'\}\rangle\big).$$

Hence we have shown that, in order to generate $\mathrm{relint}(F) \subseteq \mathcal{P}$, point $\boldsymbol{p} \in P$ can be replaced by any other point $\boldsymbol{p}' \in \mathrm{relint}(F)$.

By definition of 'gen', the contribution of $\boldsymbol{p} \in P$ is to generate the sets $\mathrm{relint}(F') \subseteq \mathcal{P}$, where $F' \in \mathit{nncFaces}$ is such that $\mathrm{relint}(F) \subseteq F'$ (i.e., all the faces of $\mathcal{P}$ containing $\mathrm{relint}(F)$). It follows that $\boldsymbol{p} \in P$ can be substituted by any other point $\boldsymbol{p}' \in \mathrm{relint}(F)$, obtaining the same polyhedron. □

Proof of Proposition 3. Let $\mathcal{SK}$ be the skeleton of the polyhedron $\mathcal{P} \in \mathbb{P}_n$, $\mathcal{Q} = \mathrm{cl}(\mathcal{P})$ and $\mathbb{NS}$ be the corresponding set of supports. In order to prove that $(\alpha_{\mathcal{SK}}, \gamma_{\mathcal{SK}})$ is a Galois connection between $\wp(\mathcal{Q})$ and $\wp_\uparrow(\mathbb{NS})$, we will show that '$\alpha_{\mathcal{SK}}$' and '$\gamma_{\mathcal{SK}}$' are monotonic, '$\alpha_{\mathcal{SK}} \circ \gamma_{\mathcal{SK}}$' is reductive and '$\gamma_{\mathcal{SK}} \circ \alpha_{\mathcal{SK}}$' is extensive; the result will thus follow from [28, Theorem 5.3.0.4].

The monotonicity of both '$\alpha_{\mathcal{SK}}$' and '$\gamma_{\mathcal{SK}}$' follows trivially from Definition 3.

Consider $NS \in \wp_{\uparrow}(\mathbb{NS})$. Note that, for each $ns \in NS$, there exists a face $F \in \mathit{cFaces}$ such that $ns = \mathcal{SK}_F$, so that $\overline{\mathrm{gen}}(ns) = F$. Therefore,

$$
\begin{aligned}
&\alpha_{\mathcal{SK}}\big(\gamma_{\mathcal{SK}}(NS)\big)\\
&\quad\text{[by definition of } \gamma_{\mathcal{SK}}\text{]}\\
&\quad= \alpha_{\mathcal{SK}}\Big(\bigcup\{\, \mathrm{relint}(\overline{\mathrm{gen}}(ns)) \mid ns \in NS \,\}\Big)\\
&\quad= \alpha_{\mathcal{SK}}\Big(\bigcup\{\, \mathrm{relint}(F) \mid F = \overline{\mathrm{gen}}(ns) \in \mathit{cFaces}, ns \in NS \,\}\Big)\\
&\quad\text{[by definition of } \alpha_{\mathcal{SK}}\text{]}\\
&\quad= \bigcup\{\, \uparrow ns \mid \exists \boldsymbol{p} \in \mathrm{relint}(F), F = \overline{\mathrm{gen}}(ns) \in \mathit{cFaces}, ns \in NS \,\}\\
&\quad= \bigcup\{\, \uparrow ns \mid ns \in NS \,\}\\
&\quad\text{[since } NS \text{ is an upward closed set]}\\
&\quad= NS.
\end{aligned}
$$

Hence, '$\alpha_{\mathcal{SK}} \circ \gamma_{\mathcal{SK}}$' is the identity function, which implies that it is reductive.

In order to show that '$\gamma_{\mathcal{SK}} \circ \alpha_{\mathcal{SK}}$' is extensive, let $S \subseteq \mathcal{Q}$. Note that for each point $\boldsymbol{p} \in S$, there exists a face $F \in \mathit{cFaces}$ such that $\boldsymbol{p} \in \mathrm{relint}(F)$. Hence:

$$
\begin{aligned}
&\gamma_{\mathcal{SK}}\big(\alpha_{\mathcal{SK}}(S)\big)\\
&\quad\text{[by definition of } \alpha_{\mathcal{SK}}\text{]}\\
&\quad= \gamma_{\mathcal{SK}}\Big(\bigcup\{\, \uparrow \mathcal{SK}_F \mid \exists \boldsymbol{p} \in S, F \in \mathit{cFaces} \,.\, \boldsymbol{p} \in \mathrm{relint}(F) \,\}\Big)\\
&\quad= \gamma_{\mathcal{SK}}\Big(\bigcup\{\, ns \mid \exists \boldsymbol{p} \in S, F \in \mathit{cFaces} \,.\, \boldsymbol{p} \in \mathrm{relint}(F), ns \in \uparrow \mathcal{SK}_F \,\}\Big)\\
&\quad\text{[by definition of } \gamma_{\mathcal{SK}}\text{]}\\
&\quad= \bigcup \left\{ \mathrm{relint}\big(\overline{\mathrm{gen}}(ns)\big) \;\middle|\; \begin{array}{l} \exists \boldsymbol{p} \in S, F \in \mathit{cFaces} \,.\, \boldsymbol{p} \in \mathrm{relint}(F),\\ ns \in \uparrow \mathcal{SK}_F \end{array} \right\}\\
&\quad\supseteq \bigcup\{\, \mathrm{relint}(F) \mid \exists \boldsymbol{p} \in S, F \in \mathit{cFaces} \,.\, \boldsymbol{p} \in \mathrm{relint}(F) \,\}\\
&\quad\supseteq \bigcup\{\, \boldsymbol{p} \in S \mid \exists F \in \mathit{cFaces} \,.\, \boldsymbol{p} \in \mathrm{relint}(F) \,\}\\
&\quad= S.
\end{aligned}
$$

□

Proof of Proposition 4. Applying '$\gamma_{\mathcal{SK}} \circ \alpha_{\mathcal{SK}}$' to the set of points P we obtain:

$$
\gamma_{\mathcal{SK}}\big(\alpha_{\mathcal{SK}}(P)\big) = \bigcup \left\{ \mathrm{relint}\big(\overline{\mathrm{gen}}(\mathcal{SK}_F)\big) \;\middle|\; \begin{array}{l} \exists \boldsymbol{p}' \in P, F' \in \mathit{cFaces} \,.\\ \boldsymbol{p}' \in \mathrm{relint}(F'),\\ \mathcal{SK}_F \in \uparrow \mathcal{SK}_{F'} \end{array} \right\}. \tag{A.1}
$$

By definition of function 'gen', a face is included in the polyhedron $\mathcal{P}$ if and only if it contains a point in P. In particular, letting $nncFaces' = nncFaces \setminus \{\emptyset\}$, this holds for the minimal faces in $nncFaces'$; these are the atoms of the lattice $\mathrm{cl}(nncFaces)$, which is a sublattice of $cFaces$. For these atoms $A \in \mathrm{cl}(nncFaces)$, we have $A = \mathrm{relint}(A)$; hence

$$\forall A \text{ atom of } \mathrm{cl}(nncFaces) : \exists \boldsymbol{p} \in P \,.\, \boldsymbol{p} \in \mathrm{relint}(A). \tag{A.2}$$

Moreover, since $\mathrm{cl}(nncFaces')$ is an upward closed set, we have:

$$\forall F' \in \mathrm{cl}(nncFaces') : \exists A \text{ atom of } \mathrm{cl}(nncFaces) \,.\, \mathcal{SK}_A \subseteq \mathcal{SK}_{F'}. \tag{A.3}$$

Therefore, we have the following chain of equations:

$$\begin{aligned}
\mathcal{P} &= \bigcup \{\, \mathrm{relint}(F) \mid F \in nncFaces' \,\} \\
&= \bigcup \{\, \mathrm{relint}\big(\overline{\mathrm{gen}}(\mathcal{SK}_F)\big) \mid F \in \mathrm{cl}(nncFaces') \,\} \\
&\quad \text{[by property (A.3)]} \\
&= \bigcup \{\, \mathrm{relint}\big(\overline{\mathrm{gen}}(\mathcal{SK}_F)\big) \mid \exists A \text{ atom of } \mathrm{cl}(nncFaces) \,.\, \mathcal{SK}_A \subseteq \mathcal{SK}_F \,\} \\
&\quad \text{[by property (A.2)]} \\
&= \bigcup \left\{ \mathrm{relint}\big(\overline{\mathrm{gen}}(\mathcal{SK}_F)\big) \;\middle|\; \begin{array}{l} \exists \boldsymbol{p} \in P, A \text{ atom of } \mathrm{cl}(nncFaces) \,. \\ \boldsymbol{p} \in \mathrm{relint}(A), \mathcal{SK}_F \in \uparrow \mathcal{SK}_A \end{array} \right\}.
\end{aligned} \tag{A.4}$$

We now show that (A.4) is equivalent to (A.1). The inclusion (A.4) $\subseteq$ (A.1) follows by simply taking $F' = A$; the other inclusion (A.4) $\supseteq$ (A.1) follows by applying property (A.3) while also observing that, since $\boldsymbol{p}' \in \mathrm{relint}(F')$, then $F' \in \mathrm{cl}(nncFaces)$. □

Appendix A.2. Proofs of results stated in Section 6

The proof of Proposition 5 is based on a couple of auxiliary lemmas.

Lemma 1. *Definition 8 specifies a binary operator on $\mathbb{P}_n$.*

Proof. We need to show that the result computed by '∇_{N}' is not affected by a change of representation for the two input arguments.

For $\mathcal{P}_1, \mathcal{P}_2 \in \mathbb{P}_n$, where $\mathcal{P}_1 \neq \emptyset$ and $\mathcal{P}_1 \subseteq \mathcal{P}_2$, let $\mathcal{P}_1 \nabla_{\mathrm{N}} \mathcal{P}_2$ be computed according to Definition 8; in particular, let $\mathcal{P}_1 \equiv (\mathcal{C}_1, \mathcal{G}_1)$ and $\mathcal{P}_2 = \mathrm{con}(\mathcal{C}_2)$, where $\mathcal{C}_i = \langle \mathcal{SK}_i^{\mathrm{c}}, NS_i^{\mathrm{c}} \rangle$ are arbitrary constraint representations for $\mathcal{P}_i$ satisfying the minimality hypothesis and $\mathcal{G}_1 = \langle \mathcal{SK}_1^{\mathrm{g}}, NS_1^{\mathrm{g}} \rangle$.

Note that, due to the inclusion hypothesis $\mathcal{P}_1 \subseteq \mathcal{P}_2$, all of the equality constraints in $\mathcal{SK}_2^{\mathrm{c}}$ are detected as stable. Let $\beta_1 \in \mathcal{SK}_1^{\mathrm{c}}$ be a skeleton (strict or non-strict) inequality constraint and $\beta_2 \in \mathcal{SK}_2^{\mathrm{c}}$ be a skeleton inequality constraint such that $\mathrm{sat}(\beta_1, \mathcal{SK}_1^{\mathrm{g}}) = \mathrm{sat}(\beta_2, \mathcal{SK}_1^{\mathrm{g}})$ holds; that is, $\beta_2 \in \mathcal{SK}_2^{\mathrm{c}}$ is detected to be stable due to $\beta_1 \in \mathcal{SK}_1^{\mathrm{c}}$. Being a skeleton constraint and due to the minimality assumption, β_1 identifies a *facet* F_1 of $\mathrm{cl}(\mathcal{P}_1)$; thus, any other constraint system

representation for $\mathcal{P}_1$ will always contain a constraint β'_1 (identifying the same facet F_1) such that $\mathrm{sat}(\beta_1, \mathcal{SK}_1^{\mathrm{g}}) = \mathrm{sat}(\beta'_1, \mathcal{SK}_1^{\mathrm{g}})$. The same reasoning can be repeated for β_2 and F_2. Hence, the computed skeleton component $\mathcal{SK}^{\mathrm{c}}$ does not depend on the chosen representations for $\mathcal{P}_1$ and $\mathcal{P}_2$. As a side note, if $\mathcal{P}_1 = \mathrm{cl}(\mathcal{P}_1)$ then no strict inequality in $\mathcal{SK}_2^{\mathrm{c}}$ can be detected as stable. When working on closed polyhedra, Definition 8 becomes equivalent to Definition 7 and we have:

$$\mathrm{cl}(\mathcal{P}_1)\nabla_{\mathrm{N}}\,\mathrm{cl}(\mathcal{P}_2) = \mathrm{cl}(\mathcal{P}_1)\nabla_{\mathrm{C}}\,\mathrm{cl}(\mathcal{P}_2), \tag{A.5}$$

where '∇_{C}' is known to be well-defined on $\mathbb{CP}_n$ [5, Theorem 5].

Finally, consider the non-skeleton component and let $ns_2 \in NS^{\mathrm{c}}$, so that $ns_2 \in NS_2^{\mathrm{c}}$ and $ns_2 \subseteq \mathcal{SK}^{\mathrm{c}}$. Support ns_2 identifies a *face* (not a facet) F_2 of $\mathrm{cl}(\mathcal{P}_2)$ which is cut from $\mathcal{P}_2$, i.e., $F_2 \cap \mathcal{P}_2 = \emptyset$. Let $\mathcal{F} = \{\, F_\beta \mid \beta \in ns_2 \,\}$ be the set of facets identified by the constraints in ns_2, so that $F_2 = \bigcap \mathcal{F}$. Note that, since ns_2 is non-redundant, all the facets in $\mathcal{F}$ have a non-empty intersection with $\mathcal{P}_2$ (i.e., they correspond to non-strict inequalities); moreover, all the facets in $\mathcal{F}$ are stable and, as observed in the previous paragraph, the set of stable facets does not depend on the chosen constraint representations. Therefore, in any other minimal representation for $\mathcal{P}_2$, there will be a set ns'_2 (i.e., a support) of non-strict skeleton constraints that identifies the same set of stable facets $\mathcal{F}$; namely, ns'_2 identifies the same cut face F_2 identified by ns_2. Hence, the computed non-skeleton component NS^{c} does not depend on the chosen representations for $\mathcal{P}_1$ and $\mathcal{P}_2$. □

Lemma 2. *$\nabla_{\mathrm{N}} \colon \mathbb{P}_n \times \mathbb{P}_n \to \mathbb{P}_n$ is a widening operator.*

Proof. For $\mathcal{P}_1, \mathcal{P}_2 \in \mathbb{P}_n$, where $\mathcal{P}_1 \neq \emptyset$ and $\mathcal{P}_1 \subseteq \mathcal{P}_2$, let $\mathcal{P}' = \mathcal{P}_1 \nabla_{\mathrm{N}} \mathcal{P}_2$ be computed according to Definition 8.

First we show that '∇_{N}' is an upper bound operator, i.e., it satisfies both $\mathcal{P}_1 \subseteq \mathcal{P}'$ and $\mathcal{P}_2 \subseteq \mathcal{P}'$. By Definition 8, it can be seen that $\mathcal{P}' = \mathrm{con}(\mathcal{SK}^{\mathrm{c}}, NS^{\mathrm{c}})$, $\mathcal{P}_2 = \mathrm{con}(\mathcal{SK}_2^{\mathrm{c}}, NS_2^{\mathrm{c}})$ and both $\mathcal{SK}^{\mathrm{c}} \subseteq \mathcal{SK}_2^{\mathrm{c}}$ and $NS^{\mathrm{c}} \subseteq NS_2^{\mathrm{c}}$ hold; hence, the inclusion $\mathcal{P}_2 \subseteq \mathcal{P}'$ follows from the anti-monotonicity of function 'con'; the other inclusion $\mathcal{P}_1 \subseteq \mathcal{P}'$ follows from the hypothesis $\mathcal{P}_1 \subseteq \mathcal{P}_2$.

Next we show that the systematic application of '∇_{N}' forces the upward iteration sequence to stabilize after a finite number of iterates. To this end, we define a ranking function $\mathrm{rank} \colon \mathbb{P}_n \to \mathbb{N}^{2+n}$, mapping a polyhedron into the well-founded set $(\mathbb{N}^{2+n}, \ll)$, where '$\ll$' denotes the strict lexicographic ordering. For each $\mathcal{P} = \mathrm{con}(\mathcal{C}) \in \mathbb{P}_n$ such that $\mathcal{P} \neq \emptyset$ and $\mathcal{C} = \langle \mathcal{SK}^{\mathrm{c}}, NS^{\mathrm{c}} \rangle$ is in minimal form, we define $\mathrm{rank}(\mathcal{P}) \doteq (e, s, f_{n-1}, \ldots, f_j, \ldots, f_0)$, where e is the number of equality constraints in $\mathcal{SK}^{\mathrm{c}}$, s is the total number of constraints in $\mathcal{SK}^{\mathrm{c}}$ and, for each $j \in \{0, \ldots, n-1\}$, f_j is the number of strict inequality constraints in $\mathcal{C}$ cutting a face of $\mathrm{cl}(\mathcal{P})$ having affine dimension j.[17] Note that 'rank' is well-defined, because $\mathcal{C}$ is in minimal form.

[17] As an example, f_0 is the number of strict inequality constraints cutting only a vertex from the topological closure of the polyhedron.

To complete the proof we have to show that, whenever $\mathcal{P}_1 \subset \mathcal{P}' = \mathcal{P}_1 \nabla_{\mathrm{N}} \mathcal{P}_2$, i.e., when the increasing sequence has not stabilized yet, the ranking function is decreasing, i.e., $\mathrm{rank}(\mathcal{P}') \ll \mathrm{rank}(\mathcal{P}_1)$.

Let $\mathrm{rank}(\mathcal{P}_1) = (e, s, f_{n-1}, \ldots, f_0)$ and $\mathrm{rank}(\mathcal{P}') = (e', s', f'_{n-1}, \ldots, f'_0)$.

Since the constraint systems are in minimal form and '∇_{N}' is an upper bound operator on $\mathbb{P}_n$, for the equality constraints we always have $e' \leq e$. If $e' < e$, then the ranking function is decreasing; thus, in the rest of the proof, we assume that $e' = e$. Namely, we assume that $\mathcal{P}_1$, $\mathcal{P}_2$ and $\mathcal{P}'$ all have the same affine dimension $k = n - e$.

Observe now that, by Definition 8, for the skeleton constraints we have $s' \leq s$. Namely, each skeleton (strict or non-strict) inequality constraint $\beta_2 \in \mathcal{SK}_2^{\mathrm{c}}$ that is selected to enter $\mathcal{SK}^{\mathrm{c}}$ has a unique corresponding skeleton constraint $\beta_1 \in \mathcal{SK}_1^{\mathrm{c}}$, which identifies the same facet of $\mathrm{cl}(\mathcal{P}')$ (recall that $\mathcal{P}_1$ and $\mathcal{P}'$ both have affine dimension k). Again, if $s' < s$, then the ranking function is decreasing; thus, in the rest of the proof, we assume both $e' = e$ and $s' = s$. Under such an assumption, by Definition 8, we obtain a one-to-one correspondence between the facets of $\mathrm{cl}(\mathcal{P}_1)$ and those of $\mathrm{cl}(\mathcal{P}')$: this implies

$$\mathrm{cl}(\mathcal{P}_1) = \mathrm{cl}(\mathcal{P}_2) = \mathrm{cl}(\mathcal{P}'). \tag{A.6}$$

Consider now the tuples $t = (f_{k-1}, \ldots, f_0)$ and $t' = (f'_{k-1}, \ldots, f'_0)$, where as said above $k = n - e$ is the affine dimension of the polyhedra.[18] By hypothesis, $\mathrm{cl}(\mathcal{P}) = \mathrm{cl}(\mathcal{P}')$ but $\mathcal{P}_1 \subset \mathcal{P}'$; hence we obtain $t \neq t'$. Moreover, we cannot have $t \ll t'$, since this would mean that there exists a strict inequality in $\mathcal{P}'$ cutting a face which is not cut from $\mathcal{P}_1$, contradicting $\mathcal{P}_1 \subset \mathcal{P}'$. Therefore $t' \ll t$, which implies $\mathrm{rank}(\mathcal{P}') \ll \mathrm{rank}(\mathcal{P}_1)$. □

It is interesting to note that the ranking function defined in the proof of Lemma 2 may be decreasing even though the number of non-redundant constraints is increasing.

Example 25. For $i = 1, 2$, consider $\mathcal{P}_i = \mathrm{con}(\mathcal{C}_i) \in \mathbb{P}_2$, where

$$\begin{aligned}
\mathcal{C}_1 &= \{0 < x < 1, 0 < y < 1\},\\
\mathcal{C}_2 &= \{0 \leq x \leq 1, 0 \leq y \leq 1, 0 < x + y < 2, -1 < x - y < 1\},
\end{aligned}$$

so that $\mathcal{P}_1$ is a topologically open square and $\mathcal{P}_2$ (which is neither closed nor open) is obtained from $\mathrm{cl}(\mathcal{P}_1)$ by cutting away its four vertices. It is easy to observe that $\mathcal{P}_1 \nabla_{\mathrm{N}} \mathcal{P}_2 = \mathcal{P}_2$, because all of the skeleton constraints are stable. Note that both constraint systems are in minimal form and their cardinalities are increasing: $|\mathcal{C}_2| = 8 > 4 = |\mathcal{C}_1|$. Nonetheless, the ranking function is decreasing:

$$\mathrm{rank}(\mathcal{P}_2) = (e', s', f'_1, f'_0) = (0, 4, 0, 4) \ll (0, 4, 4, 0) = (e, s, f_1, f_0) = \mathrm{rank}(\mathcal{P}_1).$$

[18] Note that for all $k \leq j \leq n - 1$, we have $f_j = f'_j = 0$.

Proof of Proposition 5. The proof follows by combining Lemmas 1 and 2. Note that in the proof of Lemma 2 we also proved equation (A.5), thereby proving property (1). □

Proof of Proposition 6. A complete proof would follow for the most part the corresponding one provided in [6, Theorem 12] for closed polyhedra. Therefore, we only provide a proof sketch, focusing on the relation '$\preceq_s$', as its introduction is the main difference with respect to the lgo of [6].

We first show that relation '$\curvearrowright_{\mathrm{N}}$' is an lgo on $\mathbb{P}_n$. Since $\mathcal{SK}_i^{\mathrm{c}}$ is a finite set, the multisets computed by function ζ are finite and computable. Hence the preorder '$\preceq_s$' is finitely computable and, by definition of the multiset ordering, it satisfies the ascending chain condition. The lexicographic product of a finite number of finitely computable relations satisfying the ascending chain condition is finitely computable and satisfies the ascending chain condition too.

We now prove that '$\curvearrowright_{\mathrm{N}}$' is ∇_{N}-compatible. Namely, for every $\mathcal{P}_1, \mathcal{P}_2 \in \mathbb{P}_n$ such that $\emptyset \neq \mathcal{P}_1 \subset \mathcal{P}_2$, we show that $\mathcal{P}_1 \curvearrowright_{\mathrm{N}} \mathcal{P}_1 \nabla_{\mathrm{N}} \mathcal{P}_2$. In the following we define $\mathcal{P}_3 = \mathcal{P}_1 \nabla_{\mathrm{N}} \mathcal{P}_2$ and use the subscript 3 for its components ($\mathcal{SK}_3^{\mathrm{c}}$, NS_3^{c}, S_3, ...).

Since the widening is an upper bound operator, we know that $\mathcal{P}_1 \subset \mathcal{P}_2 \subseteq \mathcal{P}_3$. By following the reasoning in the proof of [6, Theorem 12], we can observe that, in the lexicographic ordering, whenever the '$\preceq_s$' relation comes into play we necessarily have that $\mathcal{P}_1 \equiv_d \mathcal{P}_3$ and $\mathcal{P}_1 \equiv_c \mathcal{P}_3$, so that $\mathcal{P}_1$ and $\mathcal{P}_3$ have the same affine dimension and $|\mathcal{SK}_1^{\mathrm{c}}| = |\mathcal{SK}_3^{\mathrm{c}}|$; therefore, by repeating the reasoning in the proof of Lemma 2 leading to equation (A.6), we obtain $\mathrm{cl}(\mathcal{P}_1) = \mathrm{cl}(\mathcal{P}_3)$.

Let $\mathcal{F}_1 = \mathit{gFaces}_{\mathcal{P}_1}$ and $\mathcal{F}_3 = \mathit{gFaces}_{\mathcal{P}_3}$ be the downward closed sets of cut faces of $\mathcal{P}_1$ and $\mathcal{P}_3$, respectively (see Section 4.6). From $\mathrm{cl}(\mathcal{P}_1) = \mathrm{cl}(\mathcal{P}_3)$ and the strict inclusion $\mathcal{P}_1 \subset \mathcal{P}_3$, we obtain that $\mathcal{F}_3 \subset \mathcal{F}_1$. If $\mathcal{F}_3 = \{\emptyset\}$ then $\mathcal{P}_3$ is topologically closed: hence $S_3 = \emptyset \subset S_1$ and the strict ordering $\zeta(S_1) \sqsubset_{\mathrm{ms}} \zeta(S_3)$ holds trivially, so that $\mathcal{P}_1 \curvearrowright_{\mathrm{N}} \mathcal{P}_3$. Therefore in the following we assume that $\mathcal{F}_3 \neq \{\emptyset\}$. Each maximal face $F_3 \in \mathcal{F}_3$ is included in a maximal face $F_1 \in \mathcal{F}_1$; moreover, since the inclusion $\mathcal{F}_3 \subset \mathcal{F}_1$ is strict, there exists a maximal face $F_1 \in \mathcal{F}_1 \setminus \mathcal{F}_3$. Having a fixed topological closure $\mathrm{cl}(\mathcal{P}_1) = \mathrm{cl}(\mathcal{P}_3)$, these subset relations among the maximal cut faces are in direct correspondence with superset relations among their supports. Hence we obtain:

$$(\forall s_3 \in S_3 : \exists s_1 \in S_1 \,.\, s_3 \supseteq s_1) \wedge (\exists s_1 \in S_1 \setminus S_3).$$

The left-hand side property above implies $\zeta(S_1) \sqsubseteq_{\mathrm{ms}} \zeta(S_3)$; then, due to the right-hand side property, we also obtain $\zeta(S_1) \neq \zeta(S_3)$. These imply $\zeta(S_1) \sqsubset_{\mathrm{ms}} \zeta(S_3)$, so that $\mathcal{P}_1 \curvearrowright_{\mathrm{N}} \mathcal{P}_3$. □