A Direct Encoding for NNC Polyhedra

Anna Becchi and Enea Zaffanella(✉)

Department of Mathematical, Physical and Computer Sciences,
University of Parma, Parma, Italy
anna.becchi@studenti.unipr.it, enea.zaffanella@unipr.it

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. A preliminary experimental evaluation shows that the new conversion algorithm is able to achieve significant efficiency improvements.

1 Introduction

The Double Description (DD) method [28] 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 [10–12], thereby obtaining a DD pair. The procedure is incremental, 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 many software libraries and tools[1] which are used, either directly or indirectly, in research fields as diverse as bioinformatics [31,32], computational geometry [1,2], analysis of analog and hybrid systems [8,18,22,23], automatic parallelization [6,29], scheduling [16], static analysis of software [4,13,15,17,21,24].

In the classical setting, the DD method is meant to compute geometric representations for *topologically closed* polyhedra in an n-dimensional vector space. However, there are applications requiring the ability to also 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 [8,18,22,23], static analysis tools such as Pagai [24], and tools for the automatic discovery of ranking functions [13].

The few DD method implementations providing support for NNC polyhedra (Apron and PPL) are all based on an *indirect* representation. The approach, proposed in [22,23] and studied in more detail in [3,5], encodes the strict inequality

[1] An incomplete list of available implementations includes cdd [19], PolyLib [27], Apron [25], PPL [4], 4ti2 [1], Skeleton [33], Addibit [20], ELINA [30].

H. Chockler and G. Weissenbacher (Eds.): CAV 2018, LNCS 10981, pp. 230–248, 2018.
https://doi.org/10.1007/978-3-319-96145-3_13

constraints by means of an additional space dimension, playing the role of a *slack variable*; the new space dimension, usually denoted as ϵ, needs to be non-negative and bounded from above, 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 $\mathcal{P}$. The main advantage of this 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. However, the addition of a slack variable carries with itself a few technical issues.

- At the implementation level, more work is needed to make the ϵ dimension *transparent* to the end user.
- The ϵ-representation causes an *intrinsic overhead*: in any generator system for an ϵ-polyhedron, most of the "proper" points (those having a positive ϵ coordinate) need to be paired with the corresponding "closure" point (having a zero ϵ coordinate), almost doubling the number of generators.
- The DD pair in minimal form computed for an ϵ-representation $\mathcal{R}$, when reinterpreted as encoding the NNC polyhedron $\mathcal{P}$, typically includes many redundant constraints and/or generators, leading to inefficiencies. To avoid this problem, *strong minimization procedures* were defined in [3,5] that are able to detect and remove those redundancies. Even though effective, these procedures are not fully integrated into the DD conversion: they can only be applied *after* the conversion, since they interfere with incrementality. Hence, during the iterations of the conversion the ϵ-redundancies are not removed, causing the computation of bigger intermediate results.

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. 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, in principle, can be integrated into any of the available implementations of the DD method. Our experimental evaluation is conducted in the context of the PPL and shows that the new algorithm, while computing the correct results for all of the considered tests, achieves impressive efficiency improvements with respect to the implementation based on the slack variable.

The paper is structured as follows. Section 2 briefly introduces the required notation, terminology and background concepts. Section 3 proposes the new representation for NNC polyhedra; the proofs of the stated results are in [7]. The extension of the Chernikova's conversion algorithm to this new representation is

presented in Sect. 4. Section 5 reports the results obtained by the experimental evaluation. We conclude in Sect. 6.

2 Preliminaries

We assume some familiarity with the basic notions of lattice theory [9]. 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 $\bot \sqsubset b \sqsubset a$. For $S \subseteq L$, the *upward closure* of S is defined as $\uparrow S \stackrel{\text{def}}{=} \{\, x \in L \mid \exists s \in S \,.\, s \sqsubseteq x \,\}$. 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* [14] 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. 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

$$\mathrm{con}(\mathcal{C}) \stackrel{\text{def}}{=} \big\{\, \boldsymbol{p} \in \mathbb{R}^n \;\big|\; \forall \beta = (\boldsymbol{a}^{\mathrm{T}}\boldsymbol{x} \bowtie b) \in \mathcal{C}, \bowtie \in \{\geq, =\} \,.\, \boldsymbol{a}^{\mathrm{T}}\boldsymbol{p} \bowtie b \,\big\}.$$

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, $\forall \boldsymbol{p} \in \mathcal{P}$ and $\forall \rho \in \mathbb{R}_+$, it holds $\boldsymbol{p} + \rho\boldsymbol{r} \in \mathcal{P}$. The empty polyhedron has no rays. If both $\boldsymbol{r}$ and $-\boldsymbol{r}$ are rays of $\mathcal{P}$, then $\boldsymbol{r}$ is a *line* of $\mathcal{P}$. The set $\mathcal{P} \subseteq \mathbb{R}^n$ is a closed polyhedron if there exist finite sets $L, R, P \subseteq \mathbb{R}^n$ such that $\boldsymbol{0} \notin (L \cup R)$ and $\mathcal{P} = \mathrm{gen}\big(\langle L, R, P \rangle\big)$, where

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

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 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 $P \subseteq P'$.

The DD method due to Motzkin et al. [28] allows combining the constraints and the generators of a polyhedron $\mathcal{P}$ into a DD pair $(\mathcal{C}, \mathcal{G})$: a *conversion* procedure [10–12] is used to obtain each description starting from the other one, also removing the redundant elements. For presentation purposes, we focus on the conversion from constraints to generators; the opposite conversion works in the same way, using duality to switch the roles of constraints and generators. We do not describe lower level details such as the *homogenization* process, mapping the polyhedron into a polyhedral cone, or the *simplification* step, needed for computing DD pairs in minimal form.

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 $\{(\mathcal{C}_k, \mathcal{G}_k)\}_{0 \le k \le m+1}$ 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 $\mathcal{G}_{k+1} \stackrel{\text{def}}{=} \mathcal{G}_k^+ \cup \mathcal{G}_k^0 \cup \mathcal{G}_k^\star$, where $\mathcal{G}_k^\star = \mathrm{comb\_adj}_{\beta_k}(\mathcal{G}_k^+, \mathcal{G}_k^-)$ and

$$\mathrm{comb\_adj}_{\beta_k}(\mathcal{G}_k^+, \mathcal{G}_k^-) \stackrel{\text{def}}{=} \big\{\, \mathrm{comb}_{\beta_k}(g^+, g^-) \;\big|\; g^+ \in \mathcal{G}_k^+, g^- \in \mathcal{G}_k^-, \mathrm{adj}_{\mathcal{P}_k}(g^+, g^-) \,\big\}.$$

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 select only those pairs of generators that are *adjacent* in $\mathcal{P}_k$.

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 constraint $\beta = (\boldsymbol{a}^{\mathrm{T}}\boldsymbol{x} \bowtie b)$ is said to be *valid* for $\mathcal{P} \in \mathbb{CP}_n$ 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 $\mathit{cFaces}_{\mathcal{P}}$ (possibly omitting the subscript) to denote the finite set of faces of $\mathcal{P} \in \mathbb{CP}_n$. This is a meet sublattice of $\mathbb{CP}_n$ and $\mathcal{P} = \bigcup\{\, \mathrm{relint}(F) \;\big|\; F \in \mathit{cFaces}_{\mathcal{P}} \,\}$.

When $\mathcal{C}$ is extended to allow for *strict* inequalities, $\mathcal{P} = \mathrm{con}(\mathcal{C})$ is an NNC (not necessarily closed) polyhedron. The set $\mathbb{P}_n$ of all NNC polyhedra on $\mathbb{R}^n$ 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 [3, Theorem 4.4], a description of an NNC polyhedron $\mathcal{P} \in \mathbb{P}_n$ can be obtained by extending the generator system with a finite set C of *closure points*. Namely, for $\mathcal{G} = \langle L, R, C, P \rangle$, we define $\mathcal{P} = \mathrm{gen}(\mathcal{G})$, where

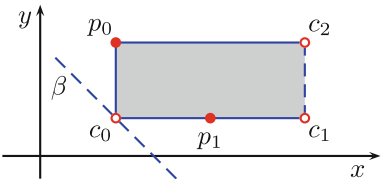
$$\mathrm{gen}\big(\langle L, R, C, P \rangle\big) \stackrel{\text{def}}{=} \left\{ L\boldsymbol{\lambda} + R\boldsymbol{\rho} + C\boldsymbol{\gamma} + P\boldsymbol{\pi} \in \mathbb{R}^n \;\middle|\; \begin{array}{l} \boldsymbol{\lambda} \in \mathbb{R}^\ell, \boldsymbol{\rho} \in \mathbb{R}^r_+, \\ \boldsymbol{\gamma}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

For an NNC polyhedron $\mathcal{P} \in \mathbb{P}_n$, the finite set $\mathit{nncFaces}_{\mathcal{P}}$ of its faces is a meet sublattice of $\mathbb{P}_n$ and $\mathcal{P} = \bigcup\{\, \mathrm{relint}(F) \;\big|\; F \in \mathit{nncFaces}_{\mathcal{P}} \,\}$. Letting $\mathcal{Q} = \mathrm{cl}(\mathcal{P})$, the closure operator $\mathrm{cl}\colon \mathit{nncFaces}_{\mathcal{P}} \to \mathit{cFaces}_{\mathcal{Q}}$ maps each NNC face of $\mathcal{P}$ into a face of $\mathcal{Q}$. The image $\mathrm{cl}(\mathit{nncFaces}_{\mathcal{P}})$ is a join sublattice of $\mathit{cFaces}_{\mathcal{Q}}$ and its nonempty elements form an *upward closed subset*, which can be described by recording the minimal elements only (i.e., the atoms of the $\mathit{nncFaces}_{\mathcal{P}}$ lattice).

3 Direct Representations for NNC Polyhedra

An NNC polyhedron can be described by using an extended constraint system $\mathcal{C} = \langle C_=, C_\geq, C_> \rangle$ and/or an extended generator system $\mathcal{G} = \langle L, R, C, P \rangle$. These representations are said to be *geometric*, meaning that they provide a precise description of the position of their elements. For a closed polyhedron $\mathcal{P} \in \mathbb{CP}_n$, the use of completely geometric representations is an adequate choice. In the case of an NNC polyhedron $\mathcal{P} \in \mathbb{P}_n$ such a choice is questionable, since the precise geometric position of some of the elements is not really needed.

Example 1. Consider the NNC polyhedron $\mathcal{P} \in \mathbb{P}_2$ in the next figure, where the (strict) inequality constraints are denoted by (dashed) lines and the (closure) points are denoted by (unfilled) circles.



$\mathcal{P}$ is described by $\mathcal{G} = \langle L, R, C, P \rangle$, 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 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.

We now show that $\mathcal{P} \in \mathbb{P}_n$ can be more appropriately represented by integrating a geometric description of $\mathcal{Q} = \mathrm{cl}(\mathcal{P}) \in \mathbb{CP}_n$ (the *skeleton*) with a combinatorial description of $nncFaces_{\mathcal{P}}$ (the *non-skeleton*). We consider here the generator system representation; the extension to constraints will be briefly outlined 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 $\mathcal{SK}_{\mathcal{Q}} = \mathrm{skel}(\mathcal{G}) \stackrel{\mathrm{def}}{=} \langle L, R, C \cup SP, \emptyset \rangle$, where $SP \subseteq P$ holds the points that can not be obtained by combining 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 $\overline{\mathrm{gen}}\big(\langle L, R, C, P \rangle\big) \stackrel{\mathrm{def}}{=} \mathrm{gen}\big(\langle L, R, \emptyset, C \cup P \rangle\big)$, showing that the skeleton of an NNC polyhedron provides a non-redundant representation of its topological closure.

Proposition 1. *If $\mathcal{P} = \mathrm{gen}(\mathcal{G})$ 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 $\overline{\mathrm{gen}}(\mathcal{G}') = \mathcal{Q}$.*

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; these *non-skeleton points* are the elements in $\mathcal{G}$ that need not be represented geometrically.

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. A less arbitrary representation for $\mathrm{relint}(F)$ is thus provided by its own skeleton $\mathcal{SK}_F \subseteq \mathcal{SK}_{\mathcal{Q}}$; 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 $\mathcal{SK}_F$.

In the following we will sometimes omit subscripts when clear from context.

Definition 2 (Support sets for a skeleton). *Let $\mathcal{SK}$ be the skeleton of an NNC polyhedron and let $\mathcal{Q} = \overline{\mathrm{gen}}(\mathcal{SK}) \in \mathbb{CP}_n$. The set of all supports for $\mathcal{SK}$ is defined as $\mathbb{NS}_{\mathcal{SK}} \stackrel{\mathrm{def}}{=} \{\, \mathcal{SK}_F \subseteq \mathcal{SK} \mid F \in \mathit{cFaces}_{\mathcal{Q}} \,\}$.*

We now define functions mapping a subset of the (geometric) points of an NNC polyhedron into the set of supports 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*

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

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

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

Proposition 2. *The pair of functions $(\alpha_{\mathcal{SK}}, \gamma_{\mathcal{SK}})$ is a Galois connection. If $\mathcal{P} = \mathrm{gen}\big(\langle L, R, C, P\rangle\big) \in \mathbb{P}_n$ and $\mathcal{SK}$ is its skeleton, 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 $NS_{\mathcal{G}} \stackrel{\mathrm{def}}{=} \alpha_{\mathcal{SK}}(P)$.*

Example 2. Consider the generator system $\mathcal{G}$ of polyhedron $\mathcal{P}$ from Example 1. Its skeleton is $\mathcal{SK} = \big\langle \emptyset, \emptyset, \{c_0, c_1, c_2, p_0\}, \emptyset \big\rangle$, so that p_1 is not a skeleton point. By Definition 3, $NS_{\mathcal{G}} = \alpha_{\mathcal{SK}}\big(\{p_0, p_1\}\big) = \uparrow\{p_0\} \cup \uparrow\{c_0, c_1\}$[2] The minimal elements in $NS_{\mathcal{G}}$ can be seen to describe the atoms of $\mathit{nncFaces}_{\mathcal{P}}$, i.e., the 0-dimension face $\{p_0\}$ and the 1-dimension open segment (c_0, c_1).

The new representation is semantically equivalent to the fully geometric one.

[2] Since there are no rays and no lines, we adopt a simplified notation, identifying each support with the set of its closure points. Also note that $\mathrm{relint}(\{p_0\}) = \{p_0\}$.

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 The New Conversion Algorithm

The CONVERSION function in Pseudocode 1 incrementally processes each of the input constraints $\beta \in \mathcal{C}_{in}$ keeping the generator system $\langle \mathcal{SK}, NS \rangle$ up-to-date. The distinction between the skeleton and non-skeleton allows for a corresponding separation in the conversion procedure. Moreover, a few minor adaptations to their representation, discussed below, allow for efficiency improvements.

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[3] 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, hence we move them from the non-skeleton to the skeleton component; namely, $\mathcal{SK} = \langle L, R, C \cup SP, \emptyset \rangle$ is represented as $\mathcal{SK} = \langle L, R, C, SP \rangle$. The formalization presented in Sect. 3 is still valid, replacing '$\gamma_{\mathcal{SK}}$' with $\gamma'_{\mathcal{SK}}(NS) \stackrel{\text{def}}{=} \mathrm{gen}(\mathcal{SK}) \cup \gamma_{\mathcal{SK}}(NS)$.

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 $\mathcal{SK}$. Since NS is a finite upward closed set, the representation only needs to record its minimal elements. A support $ns \in NS$ is *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 $\mathcal{SK} = \langle L, R, C, SP \rangle$. We write $NS_1 \oplus NS_2$ to denote the non-redundant union of $NS_1, NS_2 \subseteq \mathbb{NS}_{\mathcal{SK}}$.

4.1 Processing the Skeleton

Line 3 of CONVERSION partitions the skeleton $\mathcal{SK}$ into $\mathcal{SK}^+$, $\mathcal{SK}^0$ and $\mathcal{SK}^-$, according to the signs of the scalar products with constraint β. Note that the partition information is *logically* computed (no copies are performed) and it is stored in the $\mathcal{SK}$ component itself; therefore, any update to $\mathcal{SK}^+$, $\mathcal{SK}^0$ and $\mathcal{SK}^-$ directly propagates to $\mathcal{SK}$. In line 7 the generators in $\mathcal{SK}^+$ and $\mathcal{SK}^-$ are combined to produce $\mathcal{SK}^\star$, which is merged into $\mathcal{SK}^0$. These steps are similar to the ones for closed polyhedra, except that we now have to consider more kinds of combinations: the systematic case analysis is presented in Table 1. For instance, when processing a non-strict inequality $\beta_{\geq}$, if we combine a closure point in $\mathcal{SK}^+$ with a ray in $\mathcal{SK}^-$ we obtain a closure point in $\mathcal{SK}^\star$ (row 3, column 6). Since it is restricted to work on the skeleton component, this combination phase can safely apply the adjacency tests to quickly get rid of redundant elements.

[3] By 'singleton' here we mean a system $ns = \langle \emptyset, \emptyset, \{\boldsymbol{p}\}, \emptyset \rangle$.

Pseudocode 1. Incremental conversion from constraints to generators.

```
    function CONVERSION(C_in, ⟨SK, NS⟩)
 2:     for all β ∈ C_in do
            skel_partition(β, SK);
 4:         nonskel_partition(⟨SK, NS⟩);
            if line l ∈ SK+ ∪ SK− then VIOLATING-LINE(β, l, ⟨SK, NS⟩);
 6:         else
                SK* ← comb_adj_β(SK+, SK−); SK0 ← SK0 ∪ SK*;
 8:             NS* ← MOVE-NS(β, ⟨SK, NS⟩);
                NS* ← NS* ∪ CREATE-NS(β, ⟨SK, NS⟩);
10:             if is_equality(β) then ⟨SK, NS⟩ ← ⟨SK0, NS0 ⊕ NS*⟩;
                else if is_strict_ineq(β) then
12:                 SK0 ← points_become_closure_points(SK0);
                    ⟨SK, NS⟩ ← ⟨SK+ ∪ SK0, NS+ ⊕ NS*⟩;
14:             else ⟨SK, NS⟩ ← ⟨SK+ ∪ SK0, (NS+ ∪ NS0) ⊕ NS*⟩;
                PROMOTE-SINGLETONS(⟨SK, NS⟩);
16:    return ⟨SK, NS⟩;
```

Table 1. Case analysis for function 'comb_β' 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).

	$\mathcal{SK}^+$	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
$\beta_>$	$\mathcal{SK}^\star$	R	C	C	C	C	C	C	C	C

4.2 Processing the Non-skeleton

Line 4 partitions the supports in NS by exploiting the partition information for the skeleton $\mathcal{SK}$, so that no additional scalar product is computed. Namely, each support $ns \in NS$ is classified as follows:

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

This partitioning is consistent with the previous one. 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. The supports in $NS^\pm$ are those whose materializations can satisfy, saturate and violate the constraint β (i.e., the corresponding face *crosses* the constraint hyperplane).

In lines 8 and 9, we find the calls to the two main functions processing the non-skeleton component. A set $NS^\star$ of new supports is built as the union of the contributes provided by functions MOVE-NS and CREATE-NS.

Moving Supports. The MOVE-NS function, shown in Pseudocode 2, processes the supports in $NS^\pm$: this function "moves" the fillers of the faces that are crossed by the new constraint, making sure they lie on the correct side.

Let $ns \in NS^\pm$ and $F = \mathrm{relint}(\overline{\mathrm{gen}}(ns))$. Note that $ns = \mathcal{SK}_F$ *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* '$\mathrm{supp.cl}_{\mathcal{SK}}(ns)$' according to the updated skeleton $\mathcal{SK}$. Intuitively, $\mathrm{supp.cl}_{\mathcal{SK}}(ns) \subseteq \mathcal{SK}$ is the subset of all the skeleton elements that are included in face F.

At the implementation level, support closures can be efficiently computed by exploiting the same *saturation information* used for the adjacency tests. Namely, for constraints $\mathcal{C}$ and generators $\mathcal{G}$, we can define

$$\mathrm{sat.inter}_{\mathcal{C}}(\mathcal{G}) \stackrel{\mathrm{def}}{=} \{\, \beta' \in \mathcal{C} \mid \forall g \in \mathcal{G} : g \text{ saturates } \beta' \,\},$$
$$\mathrm{sat.inter}_{\mathcal{G}}(\mathcal{C}) \stackrel{\mathrm{def}}{=} \{\, g \in \mathcal{G} \mid \forall \beta' \in \mathcal{C} : g \text{ saturates } \beta' \,\}.$$

Then, if $\mathcal{C}$ and $\mathcal{SK} = \langle L, R, C, SP \rangle$ are the constraint system and the skeleton generator system for the polyhedron, for each $ns \in NS$ we can compute [26]:

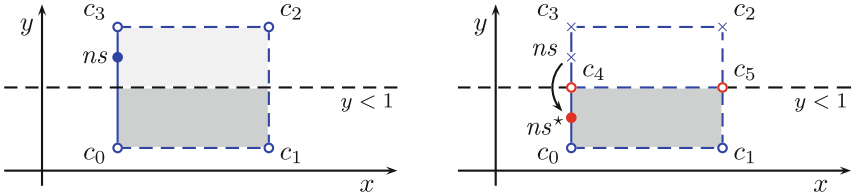
$$\mathrm{supp.cl}_{\mathcal{SK}}(ns) \stackrel{\mathrm{def}}{=} \mathrm{sat.inter}_{\mathcal{SK}}\big(\mathrm{sat.inter}_{\mathcal{C}}(ns)\big) \setminus L.$$

Face F is split by constraint β into F^+, F^0 and F^-. When β is a strict inequality, only F^+ shall be kept in the polyhedron; when the new constraint is a non-strict inequality, both F^+ and F^0 shall be kept. A minimal non-skeleton representation for these subsets can be obtained by *projecting* the support:

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

To summarize, by composing support closure and projection in line 3 of MOVE-NS, each support in $NS^\pm$ is moved to the correct side of β.

Example 3. Consider $\mathcal{P} \in \mathbb{P}_2$ in the left hand side of the next figure.



The skeleton $\mathcal{SK} = \langle \emptyset, \emptyset, C, \emptyset \rangle$ contains the 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 $\mathcal{P}$; the figure shows a single materialization for ns.

When processing $\beta = (y < 1)$, we obtain the polyhedron in the right hand side of the figure. In the skeleton phase of the CONVERSION function the adjacent skeleton generators are combined: c_4 (from $c_0 \in \mathcal{SK}^+$ and $c_3 \in \mathcal{SK}^-$) and c_5 (from $c_1 \in \mathcal{SK}^+$ 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:

$$ns^* = \mathrm{proj}^{\beta}_{\mathcal{SK}}\big(\mathrm{supp.cl}_{\mathcal{SK}}(ns)\big) = \mathrm{proj}^{\beta}_{\mathcal{SK}}\big(\{c_0, c_3, c_4\}\big) = \{c_0, c_4\}.$$

In contrast, if we were processing the non-strict inequality $\beta' = (y \leq 1)$, we would have obtained $ns' = \mathrm{proj}^{\beta'}_{\mathcal{SK}}\big(\mathrm{supp.cl}_{\mathcal{SK}}(ns)\big) = \{c_4\}$. Since ns' is a singleton, it is upgraded to become a skeleton point by procedure PROMOTE-SINGLETONS. Hence, in this case the new skeleton is $\mathcal{SK} = \langle \emptyset, \emptyset, C, SP \rangle$, where $C = \{c_0, c_1, c_5\}$ and $SP = \{c_4\}$, while the non-skeleton component is empty.

Creating New Supports. Consider the case of a support $ns \in NS^-$ violating a non-strict inequality constraint β: this support has to be removed from NS. However, the upward closed set NS is represented by its minimal elements only so that, by removing ns, we are also implicitly removing other supports from the set $\uparrow ns$, including some that do not belong to NS^- and hence should be kept. Therefore, we have to explore the set of faces and detect those that are going to lose their filler: their minimal supports will be added to $NS^\star$. Similarly, when processing a non-strict inequality constraint, we 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^+.

This is the task of the CREATE-NS function, shown in Pseudocode 2. It uses ENUMERATE-FACES as a helper:[4] 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 support closure and projection functions, as done in MOVE-NS. For efficiency purposes, a case analysis is performed so as to restrict the search area of the enumeration phase, by considering only the faces crossing the constraint.

Example 4. Consider $\mathcal{P} \in \mathbb{P}_2$ in the left hand side of the next figure, described by skeleton $\mathcal{SK} = \langle \emptyset, \emptyset, \{c_0, c_1, c_2\}, \{p\} \rangle$ and non-skeleton $NS = \emptyset$.

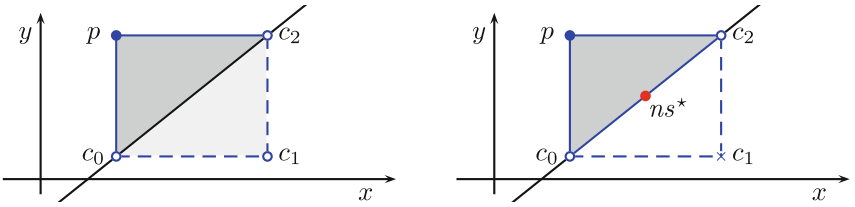
[4] This enumeration phase is inspired by the algorithm in [26].

Pseudocode 2. Helper functions for moving and creating supports.

```
    function MOVE-NS(β, ⟨SK, NS⟩)
2:      NS* ← ∅;
        for all ns ∈ NS^± do NS* ← NS* ∪ {proj^β_SK(supp.cl_SK(ns))};
4:      return NS*;
    function CREATE-NS(β, ⟨SK, NS⟩)
6:      NS* ← ∅;
        let SK = ⟨L, R, C, SP⟩;
8:      for all ns ∈ NS^− ∪ {{p} | p ∈ SP^−} do
            NS* ← NS* ∪ ENUMERATE-FACES(β, ns, SK^+, SK);
10:     if is_strict_ineq(β) then
            for all ns ∈ NS^0 ∪ {{p} | p ∈ SP^0} do
12:             NS* ← NS* ∪ ENUMERATE-FACES(β, ns, SK^+, SK);
        else
14:         for all ns ∈ NS^+ ∪ {{p} | p ∈ SP^+} do
                NS* ← NS* ∪ ENUMERATE-FACES(β, ns, SK^−, SK);
16:     return NS*;
    function ENUMERATE-FACES(β, ns, SK′, SK)
18:     NS* ← ∅; let SK′ = ⟨L′, R′, C′, SP′⟩;
        for all g ∈ (R′ ∪ C′) do NS* ← NS* ∪ {proj^β_SK(supp.cl_SK(ns ∪ {g}))};
20:     return NS*;
    procedure PROMOTE-SINGLETONS(⟨SK, NS⟩)
22:     let SK = ⟨L, R, C, SP⟩;
        for all ns ∈ NS such that ns = ⟨∅, ∅, {c}, ∅⟩ do
24:         NS ← NS \ {ns}; C ← C \ {c}; SP ← SP ∪ {c};
```

Pseudocode 3. 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);
        if is_equality(β) then SK ← SK^0;
6:      if is_strict_ineq(β) then STRICT-ON-EQ-POINTS(β, ⟨SK, NS⟩);
    procedure STRICT-ON-EQ-POINTS(β, ⟨SK, NS⟩)
8:      NS* ← ∅; let SK^0 = ⟨L^0, R^0, C^0, SP^0⟩;
        for all ns ∈ NS^0 ∪ {{p} | p ∈ SP^0} do
10:         NS* ← NS* ∪ ENUMERATE-FACES(β, ns, SK^+, SK);
        SK^0 ← points-become-closure-points(SK^0);
12:     ⟨SK, NS⟩ ← ⟨SK^+ ∪ SK^0, NS^+ ⊕ NS*⟩;
```



The partition for $\mathcal{SK}$ induced by the non-strict inequality is as follows:

$$\mathcal{SK}^+ = \langle \emptyset, \emptyset, \emptyset, \{p\} \rangle, \quad \mathcal{SK}^0 = \langle \emptyset, \emptyset, \{c_0, c_2\}, \emptyset \rangle, \quad \mathcal{SK}^- = \langle \emptyset, \emptyset, \{c_1\}, \emptyset \rangle.$$

There are no adjacent generators in $\mathcal{SK}^+$ and $\mathcal{SK}^-$, so that $\mathcal{SK}^\star$ is empty. When processing the non-skeleton component, the skeleton point in $\mathcal{SK}^+$ will be considered in line 15 of function CREATE-NS. The corresponding call to function ENUMERATE-FACES computes

$$ns^\star = \mathrm{proj}^\beta_{\mathcal{SK}}\big(\mathrm{supp.cl}_{\mathcal{SK}}(\{p\} \cup \{c_1\})\big) = \mathrm{proj}^\beta_{\mathcal{SK}}\big(\{c_0, c_1, c_2, p\}\big) = \{c_0, c_2\},$$

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

It is worth noting that, when handling Example 4 adopting an entirely geometric representation, closure point c_1 needs to be combined with point p even if the two generators are *not* adjacent: this leads to a significant efficiency penalty. Similarly, an implementation based on the ϵ-representation will have to combine closure point c_1 with point p (and/or with some other ϵ-redundant points), because the addition of the slack variable makes them adjacent. Therefore, an implementation based on the new approach obtains a twofold benefit: first, the distinction between skeleton and non-skeleton allows for restricting the handling of non-adjacent combinations to the non-skeleton phase; second, thanks to the combinatorial representation, the non-skeleton component can be processed by using set index operations only, i.e., computing no linear combination at all.

Preparing for Next Iteration. In lines 10 to 15 of CONVERSION the generator system is updated for the next iteration. The new supports in $NS^\star$ are merged (using '$\oplus$' to remove redundancies) into the appropriate portions of the non-skeleton component. In particular, when processing a strict inequality, in line 12 the helper function

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

is applied to $\mathcal{SK}^0$, making sure that all of the skeleton points saturating β are transformed into closure points having the same position. The final processing step (line 15) calls helper procedure PROMOTE-SINGLETONS (see Pseudocode 2), making sure that all singleton supports get promoted to skeleton points.

Note that line 5 of CONVERSION, by calling procedure VIOLATING-LINE (see Pseudocode 3) handles the special case of a line violating β. This is just an optimization: the helper procedure STRICT-ON-EQ-POINTS can be seen as a tailored version of CREATE-NS, also including the final updating of $\mathcal{SK}$ and NS.

4.3 Duality

The definitions given in Sect. 3 for a geometric generator system have their dual versions working on a geometric *constraint* system. We provide a brief overview of these correspondences, which are summarized in Table 2.

Table 2. Correspondences between generator and constraint concepts.

	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

For a non-empty $\mathcal{P} = \mathrm{con}(\mathcal{C}) \in \mathbb{P}_n$, the skeleton of $\mathcal{C} = \langle C_=, C_\geq, C_> \rangle$ includes the non-redundant constraints defining $\mathcal{Q} = \mathrm{cl}(\mathcal{P})$. Denoting by $SC_>$ the *skeleton strict inequalities* (i.e., those whose corresponding non-strict inequality is not redundant for $\mathcal{Q}$), we have $\mathcal{SK}_\mathcal{Q} \stackrel{\text{def}}{=} \langle C_=, C_\geq \cup SC_>, \emptyset \rangle$, so that $\mathcal{Q} = \mathrm{con}(\mathcal{SK}_\mathcal{Q})$. The *ghost* faces of $\mathcal{P}$ are the faces of the closure $\mathcal{Q}$ that do not intersect $\mathcal{P}$: $gFaces_\mathcal{P} \stackrel{\text{def}}{=} \{ F \in cFaces_\mathcal{Q} \mid F \cap \mathcal{P} = \emptyset \}$; thus, $\mathcal{P} = \mathrm{con}(\mathcal{SK}_\mathcal{Q}) \setminus \bigcup gFaces_\mathcal{P}$. The set $gFaces' \stackrel{\text{def}}{=} gFaces \cup \{\mathcal{Q}\}$ is a meet sublattice of $cFaces$; also, $gFaces$ is downward closed and can be represented by its *maximal* elements.

The skeleton support $\mathcal{SK}_F$ of a face $F \in cFaces_\mathcal{Q}$ is defined as the set of all the skeleton constraints that are saturated by all the points in 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. A constraint system non-skeleton component $NS \subseteq \mathbb{NS}$ is thus 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 that the non-redundant cutters in $gFaces$ are 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 $ns \in NS$ stands for a *skeleton strict inequality* constraint, which is better represented in the skeleton component, 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 $ns \cap SC_> \neq \emptyset$.

When the concepts underlying the skeleton and non-skeleton representation are reinterpreted as discussed above, it is possible to define a conversion procedure mapping a generator representation into a constraint representation which is very similar to the one from constraints to generators.

5 Experimental Evaluation

The new representation and conversion algorithms for NNC polyhedra have been implemented and tested in the context of the PPL (Parma Polyhedra Library). A full integration in the PPL domain of NNC polyhedra is not possible, since the latter assumes the presence of the slack variable ϵ. The approach, summarized by the diagram in Fig. 1, is to intercept each call to the PPL's conversion (working on ϵ-representations in $\mathbb{CP}_{n+1}$) and pair it with a corresponding call to the new algorithm (working on the new representations in $\mathbb{P}_n$).

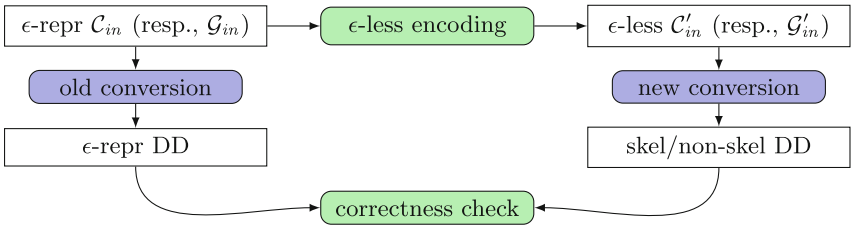


Fig. 1. High level diagram for the experimental evaluation (non-incremental case).

On the left hand side of the diagram we see the application of the standard PPL conversion procedure: the input ϵ-representation is processed by 'old conversion' so as to produce the output ϵ-representation DD pair. The 'ϵ-less encoding' phase produces a copy of the input without the slack variable; this is processed by 'new conversion' to produce the output DD pair, based on the new skeleton/non-skeleton representation. After the two conversions are completed, the outputs are checked for both semantic equivalence and non-redundancy. This final checking phase was successful on all the experiments performed, which include all of the tests in the PPL. In order to assess efficiency, additional code was added to measure the time spent inside the old and new conversion procedures, disregarding the input encoding and output checking phases. It is worth stressing that several experimental evaluations, including recent ones [2], confirm that the PPL is a state-of-the-art implementation of the DD method for a wide spectrum of application contexts.

The first experiment[5] on efficiency is meant to evaluate the *overhead* incurred by the new representation and algorithm for NNC polyhedra when processing topologically closed polyhedra, so as to compare it with the corresponding overhead incurred by the ϵ-representation. To this end, we considered the `ppl_lcdd` demo application of the PPL, which solves the *vertex/facet enumeration problem*. In Table 3 we report the results obtained on a selection of the test benchmarks[6] when using: the conversion algorithm for closed polyhedra (columns 2–3); the conversion algorithm for the ϵ-representation of NNC polyhedra (columns 4–5); and the new conversion algorithm for the new representation of NNC polyhedra (columns 6–7). Columns 'time' report the number of milliseconds spent; columns 'sat' report the number of saturation (i.e., bit vector) operations, in millions.

The results in Table 3 show that the use of the ϵ-representation for closed polyhedra incurs a significant overhead. In contrast, the new representation and algorithm go beyond all expectations: in almost all of the tests there is no overhead at all (that is, any overhead incurred is so small to be masked by the improvements obtained in other parts of the algorithm).

Table 3. Overhead of conversion for C polyhedra. Units: time (ms), sat (M).

test	closed poly		ϵ-repr		$\langle \mathcal{SK}, NS \rangle$	
	time	sat	time	sat	time	sat
cp6.ext	21	1.1	47	5.3	13	1.1
cross12.ine	157	17.1	215	18.1	180	17.2
in7.ine	47	1.7	149	6.1	27	0.9
kkd38_6.ine	498	28.3	1870	113.2	218	14.2
kq20_11_m.ine	42	1.7	153	6.1	27	0.9
metric80_16.ine	39	2.3	76	5.4	25	2.0
mit31-20.ine	1109	88.7	35629	702.2	816	60.1
mp6.ine	86	6.4	215	17.9	72	8.0
reg600-5_m.ext	906	24.7	3062	119.1	723	14.0
sampleh8.ine	5916	307.4	42339	1420.7	3309	154.1
trunc10.ine	1274	91.7	5212	396.6	803	89.9

The second experiment is meant to evaluate the efficiency gains obtained in a more appropriate context, i.e., when processing polyhedra that are *not* topologically closed. To this end, we consider the same benchmark discussed in [3, Table 2],[7] which highlights the efficiency improvement resulting from the adoption of an *enhanced* evaluation strategy (where a knowledgeable user of the

[5] All experiments have been performed on a laptop with an Intel Core i7-3632QM CPU, 16 GB of RAM and running GNU/Linux 4.13.0-25.

[6] We only show the tests where PPL time on closed polyhedra is above 20 ms.

[7] The test `dualhypercubes.cc` is distributed with the source code of the PPL.

library explicitly invokes, when appropriate, the strong minimization procedures for ϵ-representations) with respect to the *standard* evaluation strategy (where the user simply performs the required computation, leaving the burden of optimization to the library developers). In Table 4 we report the results obtained for the most expensive test among those described in [3, Table 2], comparing the standard and enhanced evaluation strategies for the ϵ-representation (rows 1 and 2) with the new algorithm (row 3). For each algorithm we show in column 2 the total number of iterations of the conversion procedures and, in the next two columns, the median and maximum sizes of the representations computed at each iteration (i.e., the size of the intermediate results); in columns from 5 to 8 we show the numbers of incremental and non-incremental calls to the conversion procedures, together with the corresponding time spent (in milliseconds); in column 9 we show the time spent in strong minimization of ϵ-representations; in the final column, we show the overall time ratio, computed with respect to the time spent by the new algorithm.

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

algorithm	# iter	iter sizes		full conv		incr conv		ϵ-min	time
		median	max	num	time	num	time	time	ratio
ϵ-repr standard	1142	3706	7259	4	11	3	30336	27	1460.9
ϵ-repr enhanced	525	109	1661	7	204	0	—	29	11.2
$\langle \mathcal{SK}, NS \rangle$ standard	314	62	180	4	6	3	15	—	1.0

Even though adopting the standard computation strategy (requiring no clever guess by the end user), the new algorithm obtains impressive time improvements, outperforming not only the standard, but also the enhanced computation strategy for the ϵ-representation. The reason for the latter efficiency improvement is that the enhanced computation strategy, when invoking the strong minimization procedures, interferes with incrementality: the figures in Table 4 confirm that the new algorithm performs three of the seven required conversions in an incremental way, while in the enhanced case they are all non-incremental. Moreover, a comparison of the iteration counts and the sizes of the intermediate results provides further evidence that the new algorithm is able to maintain a non-redundant description even *during* the iterations of a conversion.

6 Conclusion

We have presented a new approach for the representation of NNC polyhedra in the Double Description framework, avoiding the use of slack variables and distinguishing between the skeleton component, encoded geometrically, and the non-skeleton component, provided with a combinatorial encoding. We have proposed

and implemented a variant of the Chernikova conversion procedure achieving significant efficiency improvements with respect to a state-of-the-art implementation of the domain of NNC polyhedra, thereby providing a solution to all the issues affecting the ϵ-representation approach. As future work, we plan to develop a full implementation of the domain of NNC polyhedra based on this new representation. To this end, we will have to reconsider each semantic operator already implemented by the existing libraries (which are based on the addition of a slack variable), so as to propose, implement and experimentally evaluate a corresponding correct specification based on the new approach.

References

1. 4ti2 team: 4ti2—a software package for algebraic, geometric and combinatorial problems on linear spaces. www.4ti2.de
2. Assarf, B., Gawrilow, E., Herr, K., Joswig, M., Lorenz, B., Paffenholz, A., Rehn, T.: Computing convex hulls and counting integer points with polymake. Math. Program. Comput. **9**(1), 1–38 (2017)
3. Bagnara, R., Hill, P.M., Zaffanella, E.: Not necessarily closed convex polyhedra and the double description method. Form. Asp. Comput. **17**(2), 222–257 (2005)
4. Bagnara, R., Hill, P.M., Zaffanella, E.: Applications of polyhedral computations to the analysis and verification of hardware and software systems. Theor. Comput. Sci. **410**(46), 4672–4691 (2009)
5. Bagnara, R., Ricci, E., Zaffanella, E., Hill, P.M.: Possibly not closed convex polyhedra and the Parma polyhedra library. In: Hermenegildo, M.V., Puebla, G. (eds.) SAS 2002. LNCS, vol. 2477, pp. 213–229. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45789-5_17
6. Bastoul, C.: 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), Antibes Juan-les-Pins, France, pp. 7–16. IEEE Computer Society (2004)
7. Becchi, A., Zaffanella, E.: A conversion procedure for NNC polyhedra. CoRR, abs/1711.09593 (2017)
8. Benerecetti, M., Faella, M., Minopoli, S.: Automatic synthesis of switching controllers for linear hybrid systems: safety control. Theor. Comput. Sci. **493**, 116–138 (2013)
9. Birkhoff, G.: Lattice Theory. 3rd edn. Volume XXV of Colloquium Publications. American Mathematical Society, Providence (1967)
10. Chernikova, N.V.: Algorithm for finding a general formula for the non-negative solutions of system of linear equations. U.S.S.R. Comput. Math. Math. Phys. **4**(4), 151–158 (1964)
11. Chernikova, N.V.: Algorithm for finding a general formula for the non-negative solutions of system of linear inequalities. U.S.S.R. Comput. Math. Math. Phys. **5**(2), 228–233 (1965)
12. Chernikova, N.V.: Algorithm for discovering the set of all solutions of a linear programming problem. U.S.S.R. Comput. Math. Math. Phys. **8**(6), 282–293 (1968)
13. Colón, M.A., Sipma, H.B.: Synthesis of linear ranking functions. In: Margaria, T., Yi, W. (eds.) TACAS 2001. LNCS, vol. 2031, pp. 67–81. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45319-9_6

14. Cousot, P., Cousot, R.: Systematic design of program analysis frameworks. In: Proceedings of the Sixth Annual ACM Symposium on Principles of Programming Languages, San Antonio, TX, USA, pp. 269–282 (1979)
15. Cousot, P., Halbwachs, N.: Automatic discovery of linear restraints among variables of a program. In: Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages, Tucson, Arizona, pp. 84–96 (1978)
16. Doose, D., Mammeri, Z.: Polyhedra-based approach for incremental validation of real-time systems. In: Yang, L.T., Amamiya, M., Liu, Z., Guo, M., Rammig, F.J. (eds.) EUC 2005. LNCS, vol. 3824, pp. 184–193. Springer, Heidelberg (2005). https://doi.org/10.1007/11596356_21
17. Ellenbogen, R.: Fully automatic verification of absence of errors via interprocedural integer analysis. Master's thesis, School of Computer Science, Tel-Aviv University, Tel-Aviv, Israel, December 2004
18. Frehse, G.: PHAVer: algorithmic verification of hybrid systems past HyTech. Softw. Tools Technol. Transf. **10**(3), 263–279 (2008)
19. Fukuda, K., Prodon, A.: Double description method revisited. In: Deza, M., Euler, R., Manoussakis, I. (eds.) CCS 1995. LNCS, vol. 1120, pp. 91–111. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-61576-8_77
20. Genov, B.: The Convex Hull Problem in Practice: Improving the Running Time of the Double Description Method. Ph.D. thesis, University of Bremen, Germany (2014)
21. Gopan, D.: Numeric Program Analysis Techniques with Applications to Array Analysis and Library Summarization. Ph.D. thesis, University of Wisconsin, Madison, Wisconsin, USA, August 2007
22. Halbwachs, N., Proy, Y.-E., Raymond, P.: Verification of linear hybrid systems by means of convex approximations. In: Le Charlier, B. (ed.) SAS 1994. LNCS, vol. 864, pp. 223–237. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-58485-4_43
23. Halbwachs, N., Proy, Y.-E., Roumanoff, P.: Verification of real-time systems using linear relation analysis. Form. Methods Syst. Des. **11**(2), 157–185 (1997)
24. Henry, J., Monniaux, D., Moy, M.: PAGAI: a path sensitive static analyser. Electr. Notes Theor. Comput. Sci. **289**, 15–25 (2012)
25. Jeannet, B., Miné, A.: Apron: a library of numerical abstract domains for static analysis. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 661–667. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02658-4_52
26. Kaibel, V., Pfetsch, M.E.: Computing the face lattice of a polytope from its vertex-facet incidences. Comput. Geom. **23**(3), 281–290 (2002)
27. Loechner, V.: PolyLib: a library for manipulating parameterized polyhedra (1999). http://icps.u-strasbg.fr/PolyLib/
28. Motzkin, T.S., Raiffa, H., Thompson, G.L., Thrall, R.M.: The double description method. In: Contributions to the Theory of Games - Volume II, number 28 in Annals of Mathematics Studies, pp. 51–73. Princeton University Press, Princeton (1953)
29. Pop, S., Silber, G.-A., Cohen, A., Bastoul, C., Girbal, S., Vasilache, N.: GRAPHITE: Polyhedral analyses and optimizations for GCC. Technical Report A/378/CRI, Centre de Recherche en Informatique, École des Mines de Paris, Fontainebleau, France (2006)
30. Singh, G., Püschel, M., Vechev, M.T.: Fast polyhedra abstract domain. In: Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, pp. 46–59 (2017)

31. Terzer, M., Stelling, J.: Large-scale computation of elementary flux modes with bit pattern trees. Bioinformatics **24**(19), 2229–2235 (2008)
32. Terzer, M., Stelling, J.: Parallel extreme ray and pathway computation. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) PPAM 2009. LNCS, vol. 6068, pp. 300–309. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14403-5_32
33. Zolotykh, N.Y.: New modification of the double description method for constructing the skeleton of a polyhedral cone. Comput. Math. Math. Phys. **52**(1), 146–156 (2012)