EPiC Series in Computing

Volume 103, 2024, Pages 163–181

Proceedings of the 11th Int. Workshop on Applied Verification for Continuous and Hybrid Systems

Implementation of Polyhedral Operations in CORA 2024

Mark Wetzlinger, Viktor Kotsev, Adrian Kulmburg, and Matthias Althoff

Technische Universität München, Boltzmannstr. 3, 85748 Garching b. München, Germany
{m.wetzlinger,viktor.kotsev,adrian.kulmburg,althoff}@tum.de

Abstract

Tool presentation: Polyhedra are a common set representation with widespread applications in many areas of research, including convex geometry, reachability analysis, and invariant set computation. Their popularity stems from a tight relation to linear programming. Apart from some basic set operations with analytic formulae, the implementation of many polyhedral set operations in tools is not well documented. For this reason, we describe the evaluation of common set operations on polyhedra, including their runtime complexity, and provide an overview of their implementation as a dedicated class in the MATLAB tool Continuous Reachability Analyzer (CORA) for set-based computing. In particular, we highlight the handling of unbounded and degenerate sets. Finally, we compare our implementation to the popular multi-parametric toolbox.

1 Introduction

Polyhedra are elementary sets [1], in part owing to the intuitive nature of their two most popular representations: the intersection of halfspaces and the convex hull of points. They appear in many areas of numerical analysis, e.g., as constraint sets in convex optimization problems [2], as template polyhedra in reachability analysis [3, 4, 5], and as robust positively/control invariant sets [6, 7]. Many of these applications only require specific polyhedral operations, which can be implemented directly—yet, some libraries have been written to provide operations for general use. Examples include HyPro [8], written in C++, the Parma Polyhedra Library [9], written in C++, and the popular multi-parametric toolbox [10], written in MATLAB and C. Since the latter toolbox is no longer maintained, we have implemented a polytope class for the Continuous Reachability Analyzer (CORA) [11], which facilitates using polytopes among other set representations for reachability analysis [12], verification [13], set-based observers [14], and system identification [15].

In Section 2, we define the halfspace representation and the vertex representation. Then, we introduce the object class implemented in CORA in Section 3, which also contains several set properties to improve computational efficiency. Afterward in Section 4, we define all implemented operations, including conversions between representations, predicates, as well as unary and binary set operations. Our numerical evaluation in Section 5 compares our implementation with the multi-parametric toolbox [10].

2 Preliminaries

Let us first introduce some notation: Vectors are denoted by lowercase letters, matrices by uppercase letters, and sets by calligraphic letters. For a vector $v \in \mathbb{R}^n$, $v_{(i)}$ denotes its ith coordinate; for a matrix $M \in \mathbb{R}^{m \times n}$, we use $M_{(i,\cdot)}$ and $M_{(\cdot,j)}$ to denote the ith row and jth column, respectively. The transposes of a vector and matrix are written as $v^\top$ and $M^\top$, respectively. Comparisons between two vectors, such as $v_1 \leq v_2$, are evaluated element-wise. We use $\mathbf{0}$ and $\mathbf{1}$ for an all-zero or all-ones vector or matrix of proper size, and I_n to denote the n-dimensional identity matrix. The canonical basis vector in the ith dimension is written as $e_i \in \mathbb{R}^n$. The closed Euclidean ball around the origin with radius $\varepsilon > 0$ is denoted by $\mathcal{B}_\varepsilon$. For a matrix $M \in \mathbb{R}^{m \times n}$, $\text{span}(M) \subseteq \mathbb{R}^m$ is the set $\{Mx \,|\, x \in \mathbb{R}^n\}$, and $\text{span}(M)^\perp$ is its orthogonal complement, i.e., the set $\{y \in \mathbb{R}^m \,|\, \forall x \in \mathbb{R}^n \colon y^\top M x = 0\}$. The symbols $\top$ and $\bot$ denote true and false, respectively.

In general, the intersection of halfspaces results in a polyhedron, which is not necessarily bounded. If boundedness is ensured, we call the resulting set a polytope.

Definition 1 (H-polyhedra and H-polytopes [1, Sec. 1.1])**.** *A polyhedron $\mathcal{P} \subseteq \mathbb{R}^n$ in halfspace representation (or H-polyhedron) is described using $h \in \mathbb{N}$ linear inequalities defined by the matrix $A \in \mathbb{R}^{h \times n}$ and the vector $b \in \mathbb{R}^h$:*

$$\mathcal{P} \coloneqq \left\{ x \in \mathbb{R}^n \;\middle|\; Ax \leq b \right\}.$$

We use the shorthand $\mathcal{P} = \langle A, b \rangle_H$. If $\mathcal{P}$ is bounded, it is called a polytope in halfspace representation (or H-polytope). □

We use the convention that the absence of constraints implies $\mathcal{P} = \mathbb{R}^n$; this can occur, e.g., following the removal of redundant constraints if all constraints are redundant (for instance, when $A = 0$ and $b = 0$, the constraint may be removed, as it is trivially satisfied). Polytopes can equivalently be defined using points.

Definition 2 (V-polytopes [1, Sec. 1.1])**.** *A polytope $\mathcal{P} \subset \mathbb{R}^n$ in vertex representation is the convex combination of the points $\{v_1, \ldots, v_m\} \in \mathbb{R}^n$:*

$$\mathcal{P} \coloneqq \left\{ \sum_{i=1}^{m} \beta_{(i)} \, v_i \;\middle|\; \sum_{i=1}^{m} \beta_{(i)} = 1, \; \beta \geq 0 \right\}.$$

We use the shorthand $\mathcal{P} = \langle [v_1 \ldots v_m] \rangle_V$ and non-redundant points are called vertices. □

Please note that the set of points $\{v_1, \ldots, v_m\}$ may be empty, resulting in an empty polytope $\mathcal{P} = \emptyset$; this can occur, e.g., through the conversion of an H-polytope with infeasible constraints to a V-polytope.

Recently, novel representations of polytopes have been proposed [16, 17], which are related to the generator representation of polynomial zonotopes [18, Def. 1]. However, these representations are outside of the scope of this work.

We will also analyze degenerate polyhedra and polytopes.

Definition 3 (Degenerate and Non-Degenerate Sets)**.** *A set $\mathcal{S} \subseteq \mathbb{R}^n$ is non-degenerate if its topological interior is non-empty, otherwise it is degenerate. Equivalently, a set $\mathcal{S}$ is degenerate if and only if every point in $\mathcal{S}$ lies on the (topological) boundary:*

$$\forall s \in \mathcal{S}, \forall \varepsilon > 0 \colon s \oplus \mathcal{B}_\varepsilon \not\subseteq \mathcal{S}.$$

Please note that we use the convention that empty sets are degenerate. □

Table 1: Effect of binary set operations on the implemented set properties. E: empty, NE: non-empty, D: degenerate, ND: non-degenerate, B: bounded, NB: unbounded, –: unknown. For set operations that commute, we skip the second variant using '·' to avoid clutter.

$M\mathcal{S}$		$\mathcal{S}$ is ...					
		E	NE	D	ND	B	NB
M is ...	invertible	E	NE	D	ND	B	NB
	singular	E	NE, D	D	NE, D	B	–
$\mathcal{S}_1 \oplus \mathcal{S}_2$		$\mathcal{S}_2$ is ...					
		E	NE	D	ND	B	NB
$\mathcal{S}_1$ is ...	E	E	E	E	E	E	E
	NE	E	NE	–	ND	–	NB
	D	E	·	–	ND	–	NB
	ND	E	·	·	ND	ND	ND, NB
	B	E	·	·	·	B	NB
	NB	E	·	·	·	·	NB
$\mathcal{S}_1 \ominus \mathcal{S}_2$		$\mathcal{S}_2$ is ...					
		E	NE	D	ND	B	NB
$\mathcal{S}_1$ is ...	E	ND, NB	E	E	E	E	E
	NE	ND, NB	–	–	–	–	–
	D	ND, NB	–	–	E	–	–
	ND	ND, NB	–	–	–	–	–
	B	ND, NB	B	B	B	B	E
	NB	ND, NB	–	–	–	–	–
$\mathcal{S}_1 \times \mathcal{S}_2$		$\mathcal{S}_2$ is ...					
		E	NE	D	ND	B	NB
$\mathcal{S}_1$ is ...	E	E	E	E	E	E	E
	NE	E	NE	D	NE	–	NB
	D	E	·	D	D	D, B	D, NB
	ND	E	·	·	ND	–	NB
	B	E	·	·	·	B	–
	NB	E	·	·	·	·	NB
$\texttt{conv}(\mathcal{S}_1, \mathcal{S}_2)$		$\mathcal{S}_2$ is ...					
		E	NE	D	ND	B	NB
$\mathcal{S}_1$ is ...	E	E	E	E	E	E	E
	NE	E	NE	–	ND, NE	–	NB
	D	E	·	–	ND	–	NB
	ND	E	·	·	ND	ND	ND, NB
	B	E	·	·	·	B	NB
	NB	E	·	·	·	·	NB
$\mathcal{S}_1 \cap \mathcal{S}_2$		$\mathcal{S}_2$ is ...					
		E	NE	D	ND	B	NB
$\mathcal{S}_1$ is ...	E	E	E	E	E	E	E
	NE	E	–	D	–	B	–
	D	E	·	D	D	D, B	D
	ND	E	·	·	–	B	–
	B	E	·	·	·	B	B
	NB	E	·	·	·	·	–

3 Implementation in CORA

The constructor of our object class is versatile, realizing an instantiation using vertices (V), inequality constraints (A, b), equality constraints (A_e, b_e), or a combination of inequality and equality constraints (A, b, A_e, b_e). This design choice ensures that the class can handle a wide range of definitions.
Our implementation stores a list of set properties to enhance computational efficiency, as they might otherwise be recalculated numerous times during the execution of complex operations:

- **Emptiness:** Describes whether the polytope is the empty set.
- **Non-Degeneracy:** Describes whether the polytope is non-degenerate or degenerate, see Definition 3.
- **Boundedness:** Describes whether the polyhedron is bounded.
- **Minimal Representation:** Describes whether the polytope is represented by a minimal amount of halfspaces or points for the halfspace and vertex representations, respectively.

Please note that these properties result from certain set operations, as shown in Table 1 for binary set operations. In addition, it holds in general that a) empty sets are bounded, degenerate, and in minimal representation, b) non-degenerate sets are non-empty, and c) unbounded sets are non-empty. Cases with more than two properties can be analyzed by using the conjunction over all pairs of properties. For example, the Cartesian product of a non-empty set $\mathcal{S}_1$ and a degenerate, unbounded set $\mathcal{S}_2$ is a degenerate and unbounded set $\mathcal{S} = \mathcal{S}_1 \times \mathcal{S}_2$.

4 Polyhedral Operations

This section presents an overview of all implemented operations on polytopes. In Section 4.1, we discuss the conversion from halfspace to vertex representation and vice versa, known as vertex and facet enumeration, respectively. Next, we show how to evaluate predicates on polytopes, e.g., set equality, in Section 4.2. Finally, in Section 4.3, we consider set operations, including the linear map, Minkowski sum, or support function evaluation, among others. Each operation is first defined for general sets $\mathcal{S} \subseteq \mathbb{R}^n$, see Tables 2 and 3, and then evaluated for H-polytopes and V-polytopes. Please note that we only consider binary operations for computational complexity analysis. The time complexity for evaluating a linear program with p variables and q constraints is denoted by $\texttt{LP}(p, q)$, which is polynomial in p and q [19]. Moreover, we formulate feasibility checks with linear constraints as linear programs.

4.1 Minimal Representation and Conversions

Redundancy in the context of polytopes refers to the presence of superfluous constraints or points whose inclusion does not alter the set. However, a large number of redundancies may heavily impact the performance of certain set operations.

Minimal halfspace representation Given an H-polytope, one can immediately see that its jth halfspace is redundant if and only if the support function—see Table 3 for a definition—of the polytope in the direction of the corresponding normal vector remains unchanged after removing the halfspace:

$$\langle A,b\rangle_H \equiv \langle A^*,b^*\rangle_H \quad \Leftrightarrow \quad \rho\left(\langle A,b\rangle_H, A_{(j,\cdot)}^\top\right) = \rho\left(\langle A^*,b^*\rangle_H, A_{(j,\cdot)}^\top\right)$$
$$\text{with } A^* = \begin{bmatrix} A_{(1,\cdot)} \\ \vdots \\ A_{(j-1,\cdot)} \\ A_{(j+1,\cdot)} \\ \vdots \\ A_{(h,\cdot)} \end{bmatrix}, \; b^* = \begin{bmatrix} b_{(1)} \\ \vdots \\ b_{(j-1)} \\ b_{(j+1)} \\ \vdots \\ b_{(h)} \end{bmatrix}. \tag{1}$$

To obtain a minimal representation, one has to check the above condition for each halfspace. If the jth halfspace is redundant, we can further speed up the computation using the support vector ν of $\langle A^*,b^*\rangle_H$ in the direction of the jth normal vector: Since ν is also a vertex of the polytope $\langle A,b\rangle_H$, all halfspaces $k \in \{1,\dots,h\}, k \neq j$ are non-redundant, for which $A_{(k,\cdot)}\nu = b_{(k)}$ holds, because the vertex ν lies on the boundary of those halfspaces. The computational complexity for (1) is bounded by $\mathcal{O}(h\,\texttt{LP}(n,h))$ since each support function evaluation corresponds to a linear program, see (33). Please note that (1) works regardless whether the polyhedron is bounded or degenerate.

For H-polytopes with many halfspaces, one can also use further heuristics to determine redundant halfspaces. For instance, one can remove all halfspaces that strictly contain the interval hull of the polytope, which is computed using $2n$ support function evaluations [10]. This can significantly reduce the computational effort if the number of removed halfspaces is larger than $2n$, since checking whether a halfspace contains the interval hull is much faster than evaluating the support function in the direction of the normal vector of that halfspace.

Minimal vertex representation For V-polytopes, one can determine whether the jth point is redundant by checking its containment in the V-polytope composed of all other points:

$$\begin{aligned} &\langle [v_1 \dots v_m]\rangle_V \equiv \langle [v_1 \dots v_{j-1} \;\; v_{j+1} \dots v_m]\rangle_V \\ &\Leftrightarrow \; v_j \in \langle [v_1 \dots v_{j-1} \;\; v_{j+1} \dots v_m]\rangle_V . \end{aligned} \tag{2}$$

The process is applied to each point until a minimal representation is obtained. Conveniently, we use the MATLAB function `convhulln`$(\cdot)$[1], which implements the Quickhull algorithm [20] to efficiently check the redundancy of each point, with a polynomial complexity[2] of $\mathcal{O}(\frac{m}{r}f_r)$ with respect to the number of processed points r, where f_r is the maximum number of facets for r vertices, which is $\mathcal{O}(r^{\lfloor \frac{n}{2} \rfloor}/\lfloor \frac{n}{2} \rfloor!)$ according to [20]. For degenerate V-polytopes, we first restrict the set to its affine hull, see (20) and Algorithm 2, and execute the Quickhull algorithm in that subspace, followed by an embedding into the original space. Specifically, suppose we are given a V-polytope $\mathcal{P} = \langle [v_1 \dots v_m]\rangle_V$ and a matrix $B \in \mathbb{R}^{n\times r}$ representing a basis of the affine hull of $\mathcal{P}$ with minimal rank r, and c a point in $\mathcal{P}$ (for instance, this can be the arithmetic mean over all vertices). Then we know that any $x \in \mathcal{P}$ can be represented as $x = By + c$ for some $y \in \mathbb{R}^r$. If $B = U\Sigma V^\top$ is a singular value decomposition of B, since $r < n$ and B has full rank by construction, it follows that Σ has the form

$$\Sigma = \begin{bmatrix} D \\ \mathbf{0} \end{bmatrix}$$

[1] See https://www.mathworks.com/help/matlab/ref/convhulln.html
[2] Please note that the conditions for balanced execution [20, Def. 3.1] must hold.

Table 2: Definitions of unary and binary predicates using convex sets $\mathcal{S}, \mathcal{S}_1, \mathcal{S}_2 \subset \mathbb{R}^n$.

Predicate	Definition
Containment	$\mathcal{S}_1 \subseteq \mathcal{S}_2 \leftrightarrow \forall s_1 \in \mathcal{S}_1 \colon s_1 \in \mathcal{S}_2$
Set equality	$\mathcal{S}_1 \equiv \mathcal{S}_2 \leftrightarrow \mathcal{S}_1 \subseteq \mathcal{S}_2 \wedge \mathcal{S}_2 \subseteq \mathcal{S}_1$
Emptiness	$\texttt{empty}\,(\mathcal{S}) \leftrightarrow \forall x \in \mathbb{R}^n \colon x \notin \mathcal{S}$
Intersection	$\texttt{intersects}\,(\mathcal{S}_1, \mathcal{S}_2) \leftrightarrow \exists s \in \mathbb{R}^n \colon s \in \mathcal{S}_1 \wedge s \in \mathcal{S}_2$
Boundedness	$\texttt{bounded}\,(\mathcal{S}) \leftrightarrow \forall \ell \in \mathbb{R}^n, \exists a \in \mathbb{R} \colon \max_{s \in \mathcal{S}} \ell^\top s \leq a$
Degeneracy	$\texttt{degenerate}\,(\mathcal{S}) \leftrightarrow \forall s \in \mathcal{S}, \forall \varepsilon > 0 \colon s \oplus \mathcal{B}_\varepsilon \not\subseteq \mathcal{S}$

for some invertible $D \in \mathbb{R}^{r \times r}$. Since U and V are also invertible, it follows that the polytope $\mathcal{P}' = \langle [M(v_1 - c) \dots M(v_m - c)] \rangle_V \subset \mathbb{R}^r$ with $M = \begin{bmatrix} I_r & \mathbf{0} \end{bmatrix} U^\top$ is non-degenerate, and $\mathcal{P} = M^\top \mathcal{P}' + c$. Consequently, it suffices to use the Quickhull algorithm on $\mathcal{P}'$, and then embed any vertex v' of $\mathcal{P}'$ into $\mathbb{R}^n$ by means of $v = M^\top v' + c$.

Conversion between halfspace and vertex representations For the conversions between the halfspace and the vertex representation, we refer to [21]. The aforementioned Quickhull algorithm $\texttt{convhulln}\,(\cdot)$ in MATLAB also returns a list of sets of vertices making up each facet, thus, providing us with the halfspace representation, through, e.g., the well-known Gram-Schmidt algorithm.

For the solution of the vertex enumeration problem, we use a primal-dual approach where halfspaces in the primal space are mapped to vertices in the dual space [1, Ch. 2.3]. By applying the Quickhull algorithm $\texttt{convhulln}\,(\cdot)$ to these points, the facets of the resulting hull in the dual space directly yield the vertices of the original primal polytope. Degenerate polytopes are first restricted to their affine hull, see (20) and Algorithm 2, which facilitates the use of primal-dual methods for the conversion. Concretely, given a matrix $B \in \mathbb{R}^{n \times r}$ representing a basis of the affine hull of the polytope $\mathcal{P} = \langle A, b \rangle_H$ with minimal rank r, and a point $c \in \mathcal{P}$, we know that any point $x \in \mathcal{P}$ can be represented as $x = By + c$ for some $y \in \mathbb{R}^r$. Therefore, to compute the vertices of a degenerate polytope $\mathcal{P}$, it suffices to compute the vertices $v'_1, \dots, v'_m$ of $\mathcal{P}' = \langle AB, b - Ac \rangle_H$, which is non-degenerate by definition of B. The vertices of $\mathcal{P}$ are then given by $v_i = Bv'_i + c$ with $i = 1, \dots, m$, since $\mathcal{P} = B\mathcal{P}' + c$. In the unbounded case, we return an error when the vertex enumeration is called.

4.2 Predicates

We have implemented the predicates defined in Table 2.

Containment Whether a V-polytope contains another V-polytope can be formalized as

$$\left\langle \begin{bmatrix} v_1^{(1)} \dots v_{m_1}^{(1)} \end{bmatrix} \right\rangle_V \subseteq \left\langle \begin{bmatrix} v_1^{(2)} \dots v_{m_2}^{(2)} \end{bmatrix} \right\rangle_V \leftarrow \begin{cases} \top & \text{if } \forall i \in \{1, \dots, m_1\} \colon v_i^{(1)} \in \left\langle \begin{bmatrix} v_1^{(2)} \dots v_{m_2}^{(2)} \end{bmatrix} \right\rangle_V \\ \bot & \text{otherwise,} \end{cases} \tag{3}$$

where a single point v is contained if and only if the following linear program is feasible:

$$\min_{\beta \in \mathbb{R}^{m_2}} 1 \quad \text{s.t.} \quad v = \begin{bmatrix} v_1^{(2)} \dots v_{m_2}^{(2)} \end{bmatrix} \beta, \ \sum_{k=1}^{m_2} \beta_{(k)} = 1, \ \beta \geq 0. \tag{4}$$

This results in a total time complexity of $\mathcal{O}(\texttt{LP}(m_2, n))$ for the point containment, and $\mathcal{O}(m_1 \cdot \texttt{LP}(m_2, n))$ for the V-polytope in V-polytope containment.

The containment of an H-polytope in another H-polytope can be determined using support function evaluations:

$$\langle A_1, b_1 \rangle_H \subseteq \langle A_2, b_2 \rangle_H \leftarrow \begin{cases} \top & \text{if } \forall j \in \{1, \dots, h_1\} \colon \rho\left(\langle A_1, b_1 \rangle_H, A_{2(j,\cdot)}\right) \leq b_{2(j)} \\ \bot & \text{otherwise.} \end{cases} \tag{5}$$

The complexity of this operation is bounded by $\mathcal{O}(h_2 \, \texttt{LP}(n, h_1))$ since each support function evaluation is a linear program. The containment of a V-polytope in an H-polytope simplifies to a matrix-vector product, which can be evaluated in $\mathcal{O}(mhn)$:

$$\langle [v_1 \dots v_m] \rangle_V \subseteq \langle A, b \rangle_H \leftarrow \begin{cases} \top & \text{if } \forall j \in \{1, \dots, m\} \colon Av_j \leq b \\ \bot & \text{otherwise,} \end{cases} \tag{6}$$

where the inequality must hold element-wise.

Checking the containment of an H-polyhedron in a V-polytope is NP-hard [22]. Therefore, either representation is converted into the other so that (3) or (5) can be used. Note that one can immediately disprove containment if the H-polyhedron is known to be unbounded.

Set equality A V-polytope is equal to another V-polytope if and only if each point is either also contained in the set of points of the other polytope or that point is redundant, see (2):

$$\left\langle \left[v_1^{(1)} \dots v_{m_1}^{(1)} \right] \right\rangle_V \equiv \left\langle \left[v_1^{(2)} \dots v_{m_2}^{(2)} \right] \right\rangle_V \leftarrow \begin{cases} \top & \text{if } \forall j \in \{1, \dots, m_1\} \, \exists k \in \{1, \dots, m_2\} \colon v_j^{(1)} = v_k^{(2)} \\ & \quad \vee v_j^{(1)} \in \left\langle \left[v_1^{(1)} \dots v_{j-1}^{(1)} \;\; v_{j+1}^{(1)} \dots v_{m_1}^{(1)} \right] \right\rangle_V \\ & \wedge \forall k \in \{1, \dots, m_2\} \, \exists j \in \{1, \dots, m_1\} \colon v_k^{(2)} = v_j^{(1)} \\ & \quad \vee v_k^{(2)} \in \left\langle \left[v_1^{(2)} \dots v_{k-1}^{(2)} \;\; v_{k+1}^{(2)} \dots v_{m_2}^{(2)} \right] \right\rangle_V \\ \bot & \text{otherwise,} \end{cases} \tag{7}$$

whose evaluation is (coarsely) upper bounded by $\mathcal{O}(m_1 m_2 n)$.

We determine set equality between two non-degenerate H-polytopes by Algorithm 1: For each normalized constraint in A_1 of $\mathcal{P}_1$, we check whether there exists an equivalent constraint in $\mathcal{P}_2$ (condition in line 3). If this is not the case, the constraint A_1 must be redundant for set equality to hold, which can be checked using the support function (condition in line 4), as shown in (1). We then perform the same checks looping over all constraints in $\mathcal{P}_2$ (line 8). If the conditions in lines 3 and 4 are never violated, it holds that $\mathcal{P}_1 \equiv \mathcal{P}_2$. The complexity of Algorithm 1 is bounded by $\mathcal{O}(h_2 \texttt{LP}(n, h_1) + h_1 \texttt{LP}(n, h_2))$, which follows from evaluating the maximum number of possible support functions in line 4.

For degenerate H-polytopes, we evaluate the mutual containment using (5), yielding a complexity of $\mathcal{O}(h_2 \texttt{LP}(n, h_1) + h_1 \texttt{LP}(n, h_2))$:

$$\langle A_1, b_1 \rangle_H \equiv \langle A_2, b_2 \rangle_H \leftarrow \begin{cases} \top & \text{if } \langle A_1, b_1 \rangle_H \subseteq \langle A_2, b_2 \rangle_H \wedge \langle A_2, b_2 \rangle_H \subseteq \langle A_1, b_1 \rangle_H \\ \bot & \text{otherwise,} \end{cases} \tag{8}$$

which is faster than restricting the polytopes to their affine hull using Algorithm 2 and then checking for set equality with Algorithm 1.

To check whether a V-polytope and an H-polytope represent the same set, we convert either set into the representation of the other and then execute either (7), Algorithm 1, or (8) accordingly.

Algorithm 1 Set equality check for two non-degenerate H-polytopes

Require: H-polytopes $\mathcal{P}_1 = \langle A_1, b_1 \rangle_H, \mathcal{P}_2 = \langle A_2, b_2 \rangle_H$ in minimal halfspace representation
Ensure: $\mathcal{P}_1 \equiv \mathcal{P}_2$
1: $\forall i \in \{1,2\} \forall j \in \{1,...,h_i\}$: $A_{i(j,\cdot)} \leftarrow \frac{A_{i(j,\cdot)}}{\|A_{i(j,\cdot)}\|_2}, b_{i(j)} \leftarrow \frac{b_{i(j)}}{\|A_{i(j,\cdot)}\|_2}$ ▷ Row-wise normalization
2: **for** $j \leftarrow 1$ to h_1 **do**
3: **if** $\left(\nexists k \in \{1,...,h_2\}\colon A_{1(j,\cdot)} = A_{2(k,\cdot)} \wedge b_{1(j)} = b_{2(k)}\right)$
4: $\wedge \left(\rho\left(\langle A_1^*, b_1^* \rangle_H, A_{1(j,\cdot)}^\top\right) = b_{1(j)}\right)$ **then** ▷ A_1^*, b_1^* as in (1)
5: **return** $\bot$
6: **end if**
7: **end for**
8: *Repeat lines 2-7 with switched indices 1 and 2*
9: **return** $\top$

Emptiness V-polytopes are empty if and only if the set of points $\{v_1, ..., v_m\}$ is empty. For H-polytopes, we use linear programming to check whether the inequality constraints $Ax \leq b$ contain a point, which can be rewritten using Farkas' Lemma:

$$\texttt{empty}\left(\langle A, b \rangle_H\right) \leftarrow \begin{cases} \top & \text{if (10) is feasible with } y^* < 0 \\ \bot & \text{otherwise,} \end{cases} \tag{9}$$

with

$$y^* = \min_{y \in \mathbb{R}^h} b^\top y \quad \text{s.t.} \quad A^\top y = 0,\ y \geq 0. \tag{10}$$

The complexity is $\mathcal{O}(\texttt{LP}(h, n))$.

Intersection The intersection of two V-polytopes is checked as follows:

$$\texttt{intersects}\left(\left\langle \begin{bmatrix} v_1^{(1)} \dots v_{m_1}^{(1)} \end{bmatrix} \right\rangle_V, \left\langle \begin{bmatrix} v_1^{(2)} \dots v_{m_2}^{(2)} \end{bmatrix} \right\rangle_V\right) \leftarrow \begin{cases} \top & \text{if (12) is feasible} \\ \bot & \text{otherwise,} \end{cases} \tag{11}$$

where the linear program in (12) checks whether there exist two vectors $\beta_1 \in \mathbb{R}^{m_1}, \beta_2 \in \mathbb{R}^{m_2}$ fulfilling the conditions in Definition 2 and resulting in the same point:

$$\begin{aligned} &\min_{\beta_1 \in \mathbb{R}^{m_1},\, \beta_2 \in \mathbb{R}^{m_2}} 1 \\ &\text{s.t.} \quad \begin{bmatrix} v_1^{(1)} \dots v_{m_1}^{(1)} \end{bmatrix} \beta_1 = \begin{bmatrix} v_1^{(2)} \dots v_{m_2}^{(2)} \end{bmatrix} \beta_2,\ \sum_{j=1}^{m_1} \beta_{1(j)} = 1,\ \beta_1 \geq 0,\ \sum_{k=1}^{m_2} \beta_{2(k)} = 1,\ \beta_2 \geq 0. \end{aligned} \tag{12}$$

The complexity is $\mathcal{O}(\texttt{LP}(m_1 + m_2, n + m_1 + m_2))$.

Since H-polytopes are defined as the intersection of halfspaces, see Definition 1, we can directly use (9) to check for intersection:

$$\texttt{intersects}\left(\langle A_1, b_1 \rangle_H, \langle A_2, b_2 \rangle_H\right) \leftarrow \begin{cases} \bot & \text{if } \texttt{empty}\left(\langle A_1, b_1 \rangle_H \cap \langle A_2, b_2 \rangle_H\right) \\ \top & \text{otherwise,} \end{cases} \tag{13}$$

which is a single linear program that can be evaluated in $\mathcal{O}(\texttt{LP}(h_1 + h_2, n))$.

For the intersection check of a V-polytope and an H-polytope, we have

$$\texttt{intersects}\left(\langle [v_1 \dots v_m] \rangle_V, \langle A, b \rangle_H\right) \leftarrow \begin{cases} \top & \text{if (15) is feasible} \\ \bot & \text{otherwise,} \end{cases} \tag{14}$$

where

$$\min_{x \in \mathbb{R}^n,\, \beta \in \mathbb{R}^m} 1 \quad \text{s.t.} \quad Ax \leq b,\ [v_1 \dots v_m]\,\beta = x,\ \sum_{j=1}^{m} \beta_{(j)} = 1,\ \beta \geq 0 \tag{15}$$

checks in $\mathcal{O}(\texttt{LP}(n+m, h+n+m))$ whether a linear combination of vertices yields a point that fulfills the inequality constraints.

Boundedness V-polytopes are bounded by definition. For polyhedra in halfspace representation, we check whether a simplex[3] can be scaled to enclose the polytope. Consequently, we evaluate the support function along the $n+1$ normal vectors in $A_{\text{simplex}} = [I_n \ -\mathbf{1}]^\top$ bounding the simplex:

$$\texttt{bounded}\left(\langle A, b \rangle_H\right) \leftarrow \begin{cases} \top & \text{if } \forall i \in \{1, \dots, n+1\}\, \exists a \in \mathbb{R} \colon \rho\left(\langle A, b \rangle_H, A_{\text{simplex}(i,\cdot)}\right) \leq a \\ \bot & \text{otherwise.} \end{cases} \tag{16}$$

The complexity of (16) is $\mathcal{O}(n\,\texttt{LP}(n, h))$.

Degeneracy For V-polytopes, degeneracy can be determined in $\mathcal{O}(nm \min\{n, m\})$ via analyzing the rank of the matrix that contains all points shifted by their arithmetic mean $\bar{v} \in \mathbb{R}^n$ so that they enclose the origin, which is required to check whether the polytope lies in a strict subspace:

$$\texttt{degenerate}\left(\langle [v_1 \dots v_m] \rangle_V\right) \leftarrow \begin{cases} \top & \text{if } \operatorname{rank}([v_1 - \bar{v} \dots v_m - \bar{v}]) < n \\ \bot & \text{otherwise.} \end{cases} \tag{17}$$

We determine degeneracy of H-polytopes in $\mathcal{O}(\texttt{LP}(n, h))$ via the Chebyshev center, see (19):

$$\texttt{degenerate}\left(\langle A, b \rangle_H\right) \leftarrow \begin{cases} \top & \text{if } r > 0, \text{ with } r \text{ as in (19)} \\ \bot & \text{otherwise.} \end{cases} \tag{18}$$

If the radius of the Chebyshev ball is 0, then the polytope must be degenerate. Note that not only the Chebyshev center, but any interior point suffices to disprove degeneracy if a ball around that point with non-zero radius is contained in the polytope.

4.3 Set Operations

Next, we show how we have implemented the set operations in Table 3 for V-polytopes and H-polytopes.

[3] A simplex has only $n+1$ halfspaces, the smallest number for any bounded, non-degenerate H-polytope. This follows from duality, as a non-degenerate V-polytope requires at least $n+1$ vertices.

Table 3: Definitions of set operations using convex sets $\mathcal{S}_i \subseteq \mathbb{R}^n$, the matrix $M \in \mathbb{R}^{m \times n}$, and the vector $\ell \in \mathbb{R}^n$.

Set operation	Definition
Chebyshev center	See (19)
Basis of affine hull	Matrix B with minimal rank s.t. $\mathcal{S} \subseteq \text{span}(B) + c$ for some $c \in \mathbb{R}^n$
Linear map	$M\mathcal{S} \coloneqq \{Ms \,|\, s \in \mathcal{S}\}$
Minkowski sum	$\mathcal{S}_1 \oplus \mathcal{S}_2 \coloneqq \{s_1 + s_2 \,|\, s_1 \in \mathcal{S}_1, s_2 \in \mathcal{S}_2\}$
Minkowski difference	$\mathcal{S}_1 \ominus \mathcal{S}_2 \coloneqq \{s \,|\, s \oplus \mathcal{S}_2 \subseteq \mathcal{S}_1\}$
Cartesian product	$\mathcal{S}_1 \times \mathcal{S}_2 \coloneqq \{[s_1^\top \; s_2^\top]^\top \,|\, s_1 \in \mathcal{S}_1, s_2 \in \mathcal{S}_2\}$
Convex hull	$\texttt{conv}(\mathcal{S}_1, \mathcal{S}_2) \coloneqq \{\lambda s_1 + (1-\lambda)s_2 \,|\, s_1 \in \mathcal{S}_1, s_2 \in \mathcal{S}_2, \lambda \in [0,1]\}$
Support function	$\rho(\mathcal{S}, \ell) \coloneqq \max_{s \in \mathcal{S}} \ell^\top s$
Support vector	$\nu(\mathcal{S}, \ell) \coloneqq \arg\max_{s \in \mathcal{S}} \ell^\top s$
Intersection	$\mathcal{S}_1 \cap \mathcal{S}_2 \coloneqq \{s \,|\, s \in \mathcal{S}_1 \wedge s \in \mathcal{S}_2\}$
Union	$\mathcal{S}_1 \cup \mathcal{S}_2 \coloneqq \{s \,|\, s \in \mathcal{S}_1 \vee s \in \mathcal{S}_2\}$

Chebyshev center Given a V-polytope, we first convert it to an H-polytope. The Chebyshev center is defined as the center of the largest ball contained inside the polytope, which can be determined for H-polytopes through the following linear program in $\mathcal{O}(\texttt{LP}(n,h))$ [2, Sec. 8.5.1]:

$$\max_{r \in \mathbb{R},\, c \in \mathbb{R}^n} \; r \quad \text{s.t.} \quad \forall j \in \{1, \dots, h\} \colon A_{(j,\cdot)} c + r \|A_{(j,\cdot)}\|_2 \leq b_{(j)}, \; r \geq 0, \tag{19}$$

where $r \in \mathbb{R}$ is the radius of the ball and $c \in \mathbb{R}^n$ its center, i.e., the Chebyshev center. If the constraints are infeasible, the polytope is empty, and we cannot return a result; if $r = 0$, the polytope is degenerate. Please note that the Chebyshev center c is not necessarily unique—an example for this is a rectangle in $\mathbb{R}^2$.

Basis of affine hull For the computation of a minimal vertex representation of degenerate sets using the Quickhull algorithm, we require to map the set into the space of its affine hull. To obtain the basis $B \in \mathbb{R}^{n \times r}$ of the affine hull, one first has to shift the polytope by a vector $c \in \mathcal{P}$, which yields the polytope $\mathcal{P}_{\text{shift}} = \mathcal{P} - c$ that contains the origin. This allows one to analyze in which subspace $\mathcal{P}_{\text{shift}}$ lies, i.e., one can find a basis $B \in \mathbb{R}^{n \times r}$ of minimal rank such that $\mathcal{P}_{\text{shift}} \subseteq \text{span}(B)$, which yields a basis of the affine hull $\mathcal{P} \subseteq \text{span}(B) + c$.

For V-polytopes, we compute the QR decomposition of the matrix containing all points shifted by the arithmetic mean $\bar{v} \in \mathbb{R}^n$ over all points, which has a complexity of $\mathcal{O}(nm^2)$:

$$B = [Q_{(\cdot,1)} \dots Q_{(\cdot,r)}] \quad \text{with} \quad QR = \bar{V},\, r = \text{rank}\, \bar{V},\, \bar{V} = [v_1 - \bar{v} \dots v_m - \bar{v}]. \tag{20}$$

For an H-polytope, we propose a novel method to determine the basis of the affine hull, where the main idea is to iteratively append orthogonal vectors, to form an orthonormal basis that generates the affine hull of the polytope. Algorithm 2 summarizes the procedure: Assuming $\mathcal{P}$ is not empty, we first compute the Chebyshev center c (line 1) and shift the polytope so that the origin $\mathbf{0}$ is included in the shifted polytope $\mathcal{P}_{\text{shift}}$ (line 2). In each iteration, we search for a new non-zero vector x_{iter} that is perpendicular to all previously obtained vectors (line 4) stored in the matrix B. That way, we can construct the linear hull $\text{span}(B)$ that fully contains the shifted polytope. Concretely, such an x_{iter} can be computed as the maximizer for x of the

Algorithm 2 Basis of the affine hull

Require: H-polytope $\mathcal{P} = \langle A, b \rangle_H$, tolerance $\varepsilon \in \mathbb{R}, 0 < \varepsilon \ll 1$, value $\varphi \in \mathbb{R}, \varphi > 0$
Ensure: Basis $B \in \mathbb{R}^{n \times r}$ of the affine hull

```
B ← [ ], c ← (19)                          ▷ Exit if (19) is infeasible
P_shift ← P − c                            ▷ Shift so that 0 ∈ P
for r ← 1 to n do
    x_iter ← (22)
    if ‖x_iter‖_∞ ≤ ε then
        break                              ▷ No more non-trivial perpendicular directions
    end if
    B ← [B x_iter]                         ▷ Append to orthonormal basis
end for
return B
```

optimization problem

$$\max_{\substack{w \in \mathbb{R}^k \\ x + Bw \in \mathcal{P}_{\text{shift}}}} \|x\|_\infty \quad \text{s.t.} \quad \forall i \in \{1, \dots, r\} \colon B_{(\cdot,i)}^\top x = 0, \tag{21}$$

which is equivalent to

$$\max_{y \in \{e_1, -e_1, \dots, e_n, -e_n\}} \max_{\substack{w \in \mathbb{R}^r \\ x + Bw \in \mathcal{P}_{\text{shift}}}} y^\top x \quad \text{s.t.} \quad \forall i \in \{1, \dots, r\} \colon B_{(\cdot,i)}^\top x = 0. \tag{22}$$

The problem in (22) can be solved using $2n$ linear programs, one for each basis vector $\pm e_i$. Note that (22) may be unbounded if $\mathcal{P}$ is unbounded, in which case one can add the constraint $y^\top x \leq \varphi$ for some arbitrary $\varphi > 0$. If the vector x_{iter} is $\mathbf{0}$, we can conclude that $\mathcal{P}_{\text{shift}} \subseteq \text{span}(B)$. To prove this, we argue by contradiction: Suppose $\mathcal{P}_{\text{shift}} \not\subseteq \text{span}(B)$, so that there exists a point $z \in \mathcal{P}_{\text{shift}}$ with $z \notin \text{span}(B)$. Such a point can always be decomposed as $z = z_B + z_{B^\perp}$, with $z_B \in \text{span}(B)$ and $z_{B^\perp} \in \text{span}(B)^\perp$. The vector $z_{B^\perp}$ corresponds to x in (21), and z_B can, by construction, always be written as $z_B = Bw$ for some $w \in \mathbb{R}^r$, where r is the current number of columns of B. Since in (21) we maximize with respect to the length of x, the case $x_{\text{iter}} = \mathbf{0}$ can only happen if there does not exist any non-zero vector $z_{B^\perp}$ in the decomposition of z, hence z must be entirely contained in $\text{span}(B)$, which proves $\mathcal{P}_{\text{shift}} \subseteq \text{span}(B)$. The overall time complexity of Algorithm 2 is $\mathcal{O}(nr\,\texttt{LP}(n + r_{\max}, h + r_{\max}))$, where $r_{\max}$ is the number of required basis vectors, due to $2n$ support function evaluations required for solving (22).

Linear map For V-polytopes, the linear map with a matrix $M \in \mathbb{R}^{r \times n}$ follows directly from Definition 2 and Table 3, which can be evaluated in $\mathcal{O}(nmr)$:

$$M \langle [v_1 \dots v_m] \rangle_V = \langle [Mv_1 \dots Mv_m] \rangle_V . \tag{23}$$

For H-polytopes, we use the singular value decomposition

$$\begin{aligned} &M = U \Sigma V^\top, \\ &\text{where } \Sigma = \begin{bmatrix} D & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \in \mathbb{R}^{m \times n}, D \in \mathbb{R}^{(m-r) \times n}, r = \text{rank}\, M, \\ &\qquad \forall i = j \colon D_{(i,j)} > 0, \ \forall i \neq j \colon D_{(i,j)} = 0, \end{aligned}$$

in the general case, and a simple matrix inverse if M is square and invertible:

$$M \langle A, b \rangle_H = \begin{cases} \langle AM^{-1}, b \rangle_H & \text{if } M \text{ is square and invertible,} \\ U \begin{bmatrix} I_r & \mathbf{0} \end{bmatrix} \left\langle AV \begin{bmatrix} D^{-1} & \mathbf{0} \\ \mathbf{0} & I_n \end{bmatrix}, b \right\rangle_H & \text{otherwise.} \end{cases} \tag{24}$$

The simple case is $\mathcal{O}(hn^2)$ and follows directly from the definition. In contrast, the general case contains a projection operation that we solve using the Fourier-Motzkin elimination [23]: This algorithm takes a system of linear inequalities $Ax \leq b$ and projects it onto the hyperplane $\{x \in \mathbb{R}^n \,|\, x_{(j)} = 0\}$. Successive application yields a projection onto a lower-dimensional subspace. In general, the time complexity of the projection is not polynomial [1, Ch. 1.3].

Minkowski sum The Minkowski sum of two V-polytopes follows directly from Definition 2 and Table 3: We add all pairs of points in $\mathcal{O}(nm_1m_2)$,

$$\begin{aligned} &\left\langle \begin{bmatrix} v_1^{(1)} \dots v_{m_1}^{(1)} \end{bmatrix} \right\rangle_V \oplus \left\langle \begin{bmatrix} v_1^{(2)} \dots v_{m_2}^{(2)} \end{bmatrix} \right\rangle_V \\ &\quad = \left\langle \begin{bmatrix} v_1^{(1)} + v_1^{(2)} \dots v_1^{(1)} + v_{m_2}^{(2)} & \dots & v_{m_1}^{(1)} + v_1^{(2)} \dots v_{m_1}^{(1)} + v_{m_2}^{(2)} \end{bmatrix} \right\rangle_V, \end{aligned} \tag{25}$$

which generally yields redundant points that can be removed as described in Section 4.1. For H-polytopes, it follows directly from Definition 1 and Table 3 that

$$\langle A_1, b_1 \rangle_H \oplus \langle A_2, b_2 \rangle_H = \begin{bmatrix} I_n & I_n \end{bmatrix} \left\langle \begin{bmatrix} A_1 & \mathbf{0} \\ \mathbf{0} & A_2 \end{bmatrix}, \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \right\rangle_H, \tag{26}$$

which requires the projection from $2n$ to n dimensions. The Minkowski sum of two polytopes in halfspace representation is known to be NP-hard [24].

For the Minkowski sum of a V-polytope and an H-polytope, one has to convert one summand into the desired representation of the resulting set and use (25) or (26) accordingly. For CORA, we chose to always transform both summands to H-polytopes.

Minkowski difference The Minkowski difference of two V-polytopes is computed by an intermediate conversion of the minuend to the halfspace representation. For an H-polytope as a minuend, we only require to evaluate the support function of the subtrahend to obtain [25, Thm. 2.2]

$$\begin{aligned} &\langle A_1, b_1 \rangle_H \ominus \mathcal{P}_2 = \langle A_1, b^* \rangle_H \\ &\text{where } \forall j \in \{1, \dots, h_1\} \colon b^*_{(j)} = b_{1(j)} - \rho\left(\mathcal{P}_2, A_{1(j,\cdot)}^\top\right), \end{aligned} \tag{27}$$

which is $\mathcal{O}(h_1 n m_2)$ if the subtrahend $\mathcal{P}_2$ is a V-polytope and $\mathcal{O}(h_1 \, \texttt{LP}(n, h_2))$ if the subtrahend is an H-polytope. Please note that (27) also extends to arbitrary subtrahends. For example, if $\langle A_2, b_2 \rangle_H = \emptyset$, the support function evaluates to $-\infty$, see (33), and the resulting set becomes $\mathbb{R}^n$. Moreover, the result of (27) can be the empty set, which has to be checked using (9).

Cartesian product For V-polytopes, it follows directly from Definition 2 and Table 3 that

$$\left\langle \begin{bmatrix} v_1^{(1)} \dots v_{m_1}^{(1)} \end{bmatrix} \right\rangle_V \times \left\langle \begin{bmatrix} v_1^{(2)} \dots v_{m_2}^{(2)} \end{bmatrix} \right\rangle_V = \left\langle \begin{bmatrix} v_1^{(1)} & \dots & v_1^{(1)} & \dots & v_{m_1}^{(1)} & \dots & v_{m_1}^{(1)} \\ v_1^{(2)} & \dots & v_{m_2}^{(2)} & \dots & v_1^{(2)} & \dots & v_{m_2}^{(2)} \end{bmatrix} \right\rangle_V \tag{28}$$

and for H-polytopes, Definition 1 and Table 3 directly lead to the formula

$$\langle A_1, b_1 \rangle_H \times \langle A_2, b_2 \rangle_H = \left\langle \begin{bmatrix} A_1 & \mathbf{0} \\ \mathbf{0} & A_2 \end{bmatrix}, \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \right\rangle_H . \tag{29}$$

Both operations are in $\mathcal{O}(1)$. For the Cartesian product of a V-polytope and an H-polytope (in any order), one has to convert one of two operands into the desired representation of the resulting set and then apply (28) or (29) accordingly. For CORA, we chose to always transform both factors to H-polytopes.

Convex hull Since V-polytopes are defined via the convex hull of a set of points, the evaluation of the convex hull operation is trivially obtained in $\mathcal{O}(1)$:

$$\texttt{conv}\left(\left\langle \begin{bmatrix} v_1^{(1)} \dots v_{m_1}^{(1)} \end{bmatrix} \right\rangle_V, \left\langle \begin{bmatrix} v_1^{(2)} \dots v_{m_2}^{(2)} \end{bmatrix} \right\rangle_V\right) = \left\langle \begin{bmatrix} v_1^{(1)} \dots v_{m_1}^{(1)} & v_1^{(2)} \dots v_{m_2}^{(2)} \end{bmatrix} \right\rangle_V . \tag{30}$$

To compute the exact convex hull of two H-polytopes, we convert them into V-polytopes, compute the convex hull using (30), and convert the result back to H-polytopes. This procedure requires substantial computational effort, so that we instead propose a simpler method to compute an enclosure of the convex hull given a number $h_* \in \mathbb{N}$ of template directions as row vectors in $A^* \in \mathbb{R}^{h_* \times n}$ [4, Sec. 2]:

$$\begin{aligned} &\texttt{conv}\left(\langle A_1, b_1 \rangle_H, \langle A_2, b_2 \rangle_H\right) \subseteq \langle A^*, b^* \rangle_H , \\ &\forall j \in \{1, \dots, h_*\} \colon b^*_{(j)} = \max\left\{ \rho\left(\langle A_1, b_1 \rangle_H, (A^*_{(j,\cdot)})^\top\right), \rho\left(\langle A_2, b_2 \rangle_H, (A^*_{(j,\cdot)})^\top\right) \right\}, \end{aligned} \tag{31}$$

which may be sufficiently tight in case one can exploit information about the two H-polytopes to select good directions for A^*. The computational complexity is $\mathcal{O}(h_*(\texttt{LP}(n, h_1) + \texttt{LP}(n, h_2)))$.

For the convex hull of a V-polytope and an H-polytope, we convert one of the two operands to the desired representation of the resulting set and then evaluate (30) or (31) accordingly.

Support function Using the definition from Table 3, the support function of a V-polytope is obtained by simply returning the maximum value over the dot product of the direction $\ell \in \mathbb{R}^n$ and all points:

$$\rho\left(\langle [v_1 \dots v_m] \rangle_V, \ell\right) = \max_{i \in \{1,\dots,m\}} \ell^\top v_i, \tag{32}$$

which is $\mathcal{O}(nm)$. In contrast, the support function of an H-polytope requires the linear program

$$\rho\left(\langle A, b \rangle_H, \ell\right) = \max_{x \in \mathbb{R}^n} \ell^\top x \quad \text{s.t.} \quad Ax \leq b, \tag{33}$$

which can be solved in $\mathcal{O}(\texttt{LP}(n, h))$ and follows directly from inserting Definition 1 into the definition of the support function in Table 3. If the H-polytope is unbounded in the direction ℓ, the result is ∞. For an empty polytope, the support function evaluates to $-\infty$ by convention.

Support vector The support vector is given by the arguments that maximize (32) and (33),

$$\nu\left(\langle [v_1 \dots v_m] \rangle_V, \ell\right) = v_{i_*}, \text{ with } i_* = \operatorname*{arg\,max}_{i \in \{1,\dots,m\}} \ell^\top v_i \tag{34}$$

$$\nu\left(\langle A, b \rangle_H, \ell\right) = \operatorname*{arg\,max}_{x \in \mathbb{R}^n} \ell^\top x \quad \text{s.t.} \quad Ax \leq b, \tag{35}$$

for V-polytopes and H-polytopes, respectively, with the same computational complexity as for computing the support function value. Note that the support vector is in general not unique. For unbounded and empty sets, we return an error message, as the support vector is not well-defined.

Intersection Since the intersection of two V-polytopes is NP-hard to compute [24], we first convert both V-polytopes to H-polytopes, then apply the intersection operation (36) below, after which we convert the resulting H-polytope back to a V-polytope.

For H-polytopes, the intersection follows directly from Definition 1 in $\mathcal{O}(1)$:

$$\langle A_1, b_1 \rangle_H \cap \langle A_2, b_2 \rangle_H = \left\langle \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}, \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \right\rangle_H . \tag{36}$$

Note that the intersection may yield an empty set $\emptyset$.

For the intersection of a V-polytope and an H-polytope, we convert the V-polytope to an H-polytope. If the resulting set is requested in vertex representation, we apply the vertex enumeration to the result of (36).

Union Since the union of two polytopes is in general not a polytope, we return an outer approximation using the convex hull, see (30)-(31):

$$\mathcal{P}_1 \cup \mathcal{P}_2 \subseteq \texttt{conv}(\mathcal{P}_1, \mathcal{P}_2) . \tag{37}$$

5 Numerical Evaluation

In this section, we evaluate all operations from Sections 4.2 and 4.3 using our implementation in CORA, with MOSEK [26] for solving linear programs. Additionally, we provide a comparison to the multi-parametric toolbox in terms of the computation times for an increasing dimension. All computations are carried out on an AMD Ryzen 5 5600H processor with a 3.30 GHz Radeon Graphics unit and 16GB memory.

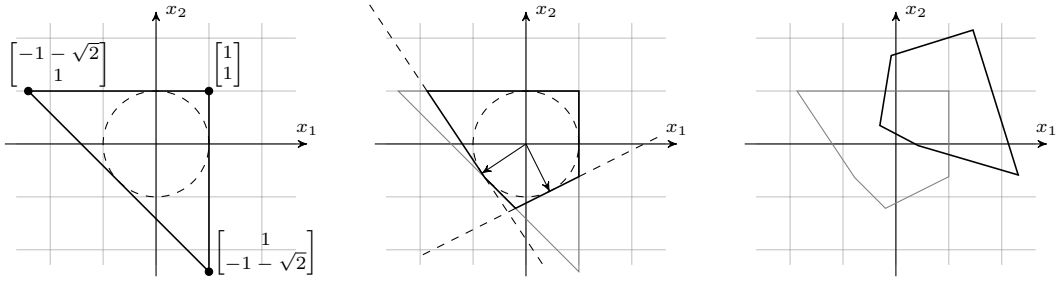
5.1 Generation of Random Polytopes

Algorithm 3 Random generation of H-polytopes

Require: Dimension $n \in \mathbb{N}$, number of constraints $h \in \mathbb{N}, h \geq n+1$
Ensure: Bounded, non-degenerate polytope $\mathcal{P} \subset \mathbb{R}^n$
1: $A \leftarrow [I_n \ \ -\frac{1}{\sqrt{n}}\mathbf{1}]^\top, b \leftarrow \mathbf{1}$ ▷ Initialize simplex
2: **for** $j \leftarrow n+2$ to h **do**
3: Sample direction $\ell \in \mathbb{R}^n$ from unit ball
4: $A \leftarrow [A^\top \ \ell]^\top$, $b \leftarrow [b^\top \ 1]^\top$ ▷ Append new constraint
5: **end for**
6: Sample invertible matrix $M \in \mathbb{R}^{n \times n}$, sample vector $z \in \mathbb{R}^n$
7: $\mathcal{P} \leftarrow M \langle A, b \rangle_H + z$ ▷ Affine map
8: **return** $\mathcal{P}$

One can easily generate random polytopes by sampling points in $\mathbb{R}^n$ and computing the convex hull. Since the convex hull as well as the facet enumeration for the conversion to H-polytopes are computationally demanding operations for large dimensions n, we propose a more direct method. The procedure summarized in Algorithm 3 randomly generates bounded, non-degenerate H-polytopes, which requires at least $n+1$ constraints. The three main steps are illustrated in Figure 1 for the generation of a two-dimensional polytope with five constraints:

Step 1: (Figure 1a) We instantiate a simplex (line 1), which has $n+1$ constraints.



(a) Step 1: 2D-Simplex (line 1). (b) Step 2: Intersection with additional constraints (line 4). (c) Step 3: Affine map (line 7).

Figure 1: Illustration of Algorithm 3 for $n = 2$ and $h = 5$.

Step 2: (Figure 1b) For the remaining number of constraints (if $h = n+1$, we directly proceed to Step 3), we sample a random direction $\ell \in \mathbb{R}^n$ from the unit ball and append the inequality constraint $\ell x \leq 1$ to the constraint matrix A and offset b (line 4). Since the simplex is chosen such that a unit ball fits tightly inside, any direction ℓ of unit length and with an associated offset of 1 will be non-redundant.

Step 3: (Figure 1c) We compute the affine map of the polytope $\langle A, b \rangle_H$ (line 7), where we have to ensure that the matrix M is invertible, as the resulting polytope would otherwise become degenerate. This can easily be ensured by choosing, e.g., the matrix U from the singular value decomposition $N = U\Sigma V^\top$ of a random matrix $N \in \mathbb{R}^{n \times n}$.

5.2 Comparison to the Multi-Parametric Toolbox

Finally, we compare our implementation in CORA to the popular multi-parametric toolbox using the average computation time for each operation described in Sections 4.2 and 4.3. The random polytopes of increasing dimension n required for the comparison are generated by Algorithm 3 with the number of constraints chosen randomly between $2n + \lfloor n/2 \rfloor$ and $3n$.

Table 4 shows the computation times averaged over 50 different polytopes per dimension. Both tools are similarly fast at removing redundant points from the vertex representation as well as the vertex enumeration. In contrast, our implementation is much faster in removing redundant halfspaces and in converting from the vertex to the halfspace representation. CORA outperforms the multi-parametric toolbox in the evaluation of all predicates introduced in Section 4.2, and the difference becomes more apparent as the set dimension increases. Moreover, our implementations for checking containment, boundedness, and set equality are orders of magnitude faster. Please note that we omitted the intersection check, as it amounts to checking emptiness. CORA is also faster for almost all set operations defined in Section 4.3. The time gap for computing the Chebyshev center as well as the linear map with an invertible matrix widens over an increasing dimension. For the linear map with a singular matrix and the Minkowski sum, both of which cause computationally demanding projections onto a lower-dimensional subspaces, and the evaluation of the Minkowski difference, CORA is seconds quicker than the multi-parametric toolbox in large dimensions. The evaluation of the support function and the

support vector is similarly fast for both tools. Finally, our novel algorithm for computing the basis of the affine hull also scales well. Please note that the convex hull and union operations are not included, as they mainly amount to enumerating the vertices; the Cartesian product and intersection are left out since their complexity is $\mathcal{O}(1)$.

6 Conclusion

This work presents our implementation of a polytope class in the Continuous Reachability Analyzer (CORA) for set-based computing. Moreover, this is the first paper that exhaustively documents the most relevant polytope set operations. In particular, we explicitly handle degenerate and unbounded sets for all operations. Our numerical evaluation on randomly generated polytopes of varying dimensions shows a significant speed up for computationally demanding operations in comparison to the popular multi-parametric toolbox.

7 Acknowledgments

The authors gratefully acknowledge financial support by the project justITSELF funded by the European Research Council (ERC) under grant agreement No 817629.

References

[1] G. M. Ziegler. *Lectures on polytopes*. Springer Science & Business Media, 2012.

[2] S. P. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004. DOI: 0.1017/CBO9780511804441.

[3] A. Girard and C. Le Guernic. Efficient reachability analysis for linear systems using support functions. *IFAC Proceedings Volumes*, 41(2), 2008. DOI: 10.3182/20080706-5-KR-1001.01514.

[4] C. Le Guernic and A. Girard. Reachability analysis of linear systems using support functions. *Nonlinear Analysis: Hybrid Systems*, 4(2):250–262, 2010. DOI: 10.1016/j.nahs.2009.03.002.

[5] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler. SpaceEx: Scalable verification of hybrid systems. In *Proc. of the 23rd International Conference on Computer Aided Verification*, LNCS 6806, pages 379–395. Springer, 2011. DOI: 10.1007/978-3-642-22110-1_30.

[6] P. Trodden. A one-step approach to computing a polytopic robust positively invariant set. *IEEE Transactions on Automatic Control*, 61(12):4100–4105, 2016. DOI: 10.1109/TAC.2016.2541300.

[7] M. Rungger and P. Tabuada. Computing robust controlled invariant sets of linear systems. *IEEE Transactions on Automatic Control*, 62(7):3665–3670, 2017. DOI: 10.1109/TAC.2017.2672859.

[8] S. Schupp, E. Ábrahám, I. Ben Makhlouf, and S. Kowalewski. HyPro: A C++ library of state set representations for hybrid systems reachability analysis. In *Proc. of the 9th NASA Formal Methods Symposium*, pages 288–294, 2017. DOI: 10.1007/978-3-319-57288-8_20.

Table 4: Average computation time (in s) over 50 randomly generated polytopes of increasing set dimension, faster time in bold. ✻: Computation time $\geq 10\,\text{s}$, ✗: operation not supported.

Operation	**Tool**	**Dimension**						
		2	3	4	10	20	30	50
Redundancy removal (V-polytopes)	CORA	0.001	0.001	0.001	**0.793**	✻	✻	✻
	mpt	0.001	0.001	0.001	0.797	✻	✻	✻
Redundancy removal (H-polytopes)	CORA	**0.002**	0.007	0.009	0.020	**0.049**	**0.113**	**0.387**
	mpt	0.004	**0.004**	**0.005**	**0.014**	0.076	0.356	3.319
Facet enumeration	CORA	0.001	0.001	0.001	**0.301**	✻	✻	✻
	mpt	0.001	0.001	0.001	1.515	✻	✻	✻
Vertex enumeration	CORA	**0.003**	**0.003**	**0.004**	**0.695**	✻	✻	✻
	mpt	0.005	0.006	0.007	0.793	✻	✻	✻
Containment	CORA	**0.002**	**0.003**	**0.004**	**0.009**	**0.023**	**0.052**	**0.194**
	mpt	0.004	0.005	0.006	0.010	0.088	0.493	4.961
Emptiness	CORA	0.002	0.002	0.002	0.002	0.003	**0.004**	**0.006**
	mpt	**0.001**	**0.001**	**0.001**	0.002	0.003	0.012	0.108
Boundedness	CORA	**0.001**	**0.002**	**0.002**	**0.006**	**0.019**	**0.040**	**0.152**
	mpt	0.007	0.008	0.012	0.037	0.201	0.963	6.640
Degeneracy	CORA	0.001	0.001	0.001	0.001	**0.002**	**0.003**	**0.005**
	mpt	0.001	0.001	0.001	0.001	0.003	0.009	0.064
Set equality	CORA	**0.002**	**0.003**	**0.003**	**0.003**	**0.004**	**0.006**	**0.011**
	mpt	0.005	0.006	0.008	0.025	0.134	0.802	7.351
Chebyshev center	CORA	0.001	0.001	0.001	0.001	**0.002**	**0.002**	**0.004**
	mpt	0.001	0.001	0.001	0.001	0.003	0.010	0.109
Basis of affine hull	CORA	**0.036**	**0.063**	**0.112**	**0.618**	**2.831**	✻	✻
	mpt	✗	✗	✗	✗	✗	✗	✗
Linear map (invertible matrix)	CORA	0.001	0.001	0.001	**0.001**	**0.001**	**0.001**	**0.001**
	mpt	0.001	0.001	0.001	0.003	0.005	0.015	0.116
Linear map (singular matrix)	CORA	**0.006**	0.011	0.012	0.025	**0.063**	**0.169**	**0.999**
	mpt	0.008	**0.009**	**0.010**	0.025	0.117	0.586	6.101
Minkowski sum	CORA	0.019	0.023	**0.097**	✻	✻	✻	✻
	mpt	**0.009**	**0.012**	4.961	✻	✻	✻	✻
Minkowski difference	CORA	**0.004**	**0.005**	**0.006**	**0.017**	**0.048**	**0.110**	**0.407**
	mpt	0.007	0.008	0.010	0.028	0.140	0.587	5.199
Support function, support vector	CORA	0.001	0.001	0.001	0.001	0.002	**0.002**	**0.005**
	mpt	0.001	0.001	0.001	0.001	0.002	0.004	0.012

[9] R. Bagnara, P. M. Hill, and E. Zaffanella. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Science of Computer Programming*, 72(1):3–21, 2008. DOI: https://doi.org/10.1016/j.scico.2007.08.001.

[10] M. Herceg, M. Kvasnica, C. N. Jones, and M. Morari. Multi-Parametric Toolbox 3.0. In *Proc. of the European Control Conference*, pages 502–510, 2013. http://control.ee.ethz.ch/~mpt.

[11] M. Althoff. An introduction to CORA 2015. In *Proc. of the Workshop on Applied Verification for Continuous and Hybrid Systems*, pages 120–151, 2015. DOI: 10.29007/zbkv.

[12] N. Kochdumper and M. Althoff. Computing non-convex inner-approximations of reachable sets for nonlinear continuous systems. In *Proc. of the 59th Conference on Decision and Control*, pages 2130–2137. IEEE, 2020. DOI: 10.1109/CDC42340.2020.9304022.

[13] M. Wetzlinger, N. Kochdumper, S. Bak, and M. Althoff. Fully automated verification of linear systems using inner and outer approximations of reachable sets. *IEEE Transactions on Automatic Control*, 68(12):7771–7786, 2023. DOI: 10.1109/TAC.2023.3292008.

[14] M. Althoff. Guaranteed state estimation in CORA 2021. In *Proc. of the 8th Workshop on Applied Verification for Continuous and Hybrid Systems*, pages 161–175, 2021. DOI: 10.29007/7m2k.

[15] M. Althoff. Checking and establishing reachset conformance in CORA 2023. In *Proc. of the 10th Workshop on Applied Verification for Continuous and Hybrid Systems*, pages 9–33, 2023. DOI: 10.29007/5v1g.

[16] N. Kochdumper and M. Althoff. Representation of polytopes as polynomial zonotopes. *arXiv preprint arXiv:1910.07271*, 2019. DOI: 10.48550/arXiv.1910.07271.

[17] S. Sigl and M. Althoff. M-representation of polytopes. *arXiv preprint arXiv:2303.05173*, 2023. DOI: 10.48550/arXiv.2303.05173.

[18] N. Kochdumper and M. Althoff. Sparse polynomial zonotopes: A novel set representation for reachability analysis. *IEEE Transactions on Automatic Control*, 66(2):4043–4058, 2021. DOI: 10.1109/TAC.2020.3024348.

[19] P. M. Vaidya. An algorithm for linear programming which requires $\mathcal{O}(((M+n)n^2+(M+n)^{1.5}n)L)$ arithmetic operations. In *Proc. of the 19th Annual Symposium on Theory of Computing*, pages 29–38. ACM, 1987. DOI: 10.1145/28395.28399.

[20] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. The Quickhull algorithm for convex hulls. *ACM Transactions Mathematical Software*, 22(4):469–483, 1996. DOI: 10.1145/235815.235821.

[21] D. Bremner, K. Fukuda, and A. Marzetta. Primal-dual methods for vertex and facet enumeration. *Discrete & Computational Geometry*, 20:333–357, 1998. DOI: 10.1007/PL00009389.

[22] K. Kellner, T. Theobald, and C. Trabandt. Containment problems for polytopes and spectrahedra. *SIAM Journal on Optimization*, 23(2):1000–1020, 2013. DOI: 10.1137/120874898.

[23] G. B. Dantzig and B. C. Eaves. Fourier-Motzkin elimination and its dual. *Journal of Combinatorial Theory, Series A*, 14(3):288–297, 1973. DOI: https://doi.org/10.1016/0097-3165(73)90004-6.

[24] H. R. Tiwary. On the hardness of computing intersection, union and Minkowski sum of polytopes. *Discrete & Computational Geometry*, 40(3):469–479, 2008. DOI: 10.1007/s00454-008-9097-3.

[25] I. Kolmanovsky and E. G. Gilbert. Theory and computation of disturbance invariant sets for discrete-time linear systems. *Mathematical Problems in Engineering*, 4, 1998. DOI: 10.1155/S1024123X98000866.

[26] Mosek ApS. *The MOSEK optimization toolbox for MATLAB 10.0.47*. 2024.