Efficient Symbolic Representation of Convex Polyhedra in High-Dimensional Spaces

Bernard Boigelot and Isabelle Mainz

Institut Montefiore, B28,
Université de Liège,
B-4000 Liège, Belgium
{Bernard.Boigelot, Isabelle.Mainz}@uliege.be

Abstract. This work is aimed at developing an efficient data structure for representing symbolically convex polyhedra. We introduce an original data structure, the *Decomposed Convex Polyhedron (DCP)*, that is closed under intersection and linear transformations, and allows to check inclusion, equality, and emptiness. The main feature of DCPs lies in their ability to represent concisely polyhedra that can be expressed as combinations of simpler sets, which can overcome combinatorial explosion in high dimensional spaces. DCPs also have the advantage of being reducible into a canonical form, which makes them efficient for representing simple sets constructed by long sequences of manipulations, such as those handled by state-space exploration tools. Their practical efficiency has been evaluated with the help of a prototype implementation, with promising results.

1 Introduction

Convex polyhedra, i.e., the subsets of $\mathbb{R}^n$ defined by finite conjunctions of linear constraints, are extensively used in many areas of computer science. Among their many applications, convex polyhedra are employed in optimization theory and in particular linear programming [23], constraint programming, Satisfiability Modulo Theories (SMT) solving [10], abstract interpretation, for which they are one of the most used numerical abstract domains [12, 13], and computer-aided verification [5, 19, 15].

Our motivation for studying convex polyhedra is to use them for representing the reachable sets produced during symbolic state-space exploration of linear hybrid systems and temporal automata [18, 9, 7, 1]. For this application, one needs a data structure that is closed under intersection and linear transformations, in order to be able to compute the image of sets by the transition relation of the system under analysis. Furthermore, it should be possible to decide inclusion, equality, and emptiness of represented sets, in order to detect that a fixed point has been reached, as well as for comparing the reachability set against the safety property of interest. Our choice is to aim for an exact symbolic representation, in the sense that it should both rely only on exact arithmetic, and not over- or under-approximate the represented sets.

Existing solutions to this problem have several drawbacks. Representations based on logical formulas are notoriously difficult to simplify. This makes them inefficient for handling simple sets constructed by long sequences of manipulations, such as those produced by state-space exploration procedures.

Another well known representation is the double description method [22, 11], used by most popular software libraries for handling convex polyhedra, such as cdd [16], PolyLib [21], NewPolka [20], and PPL [4]. This technique consists in jointly describing a polyhedron by two different geometric representations: a constraint system, expressing the polyhedron as the set of solutions of a finite conjunction of linear constraints, and a generator system, defining the polyhedron as the convex-conical combination of a finite set of vertices and extremal rays. These two representations are equivalent, in the sense that each of them can be reconstructed from the other. However, keeping both of them makes it possible to speed up some operations, such as removing their redundant elements. The major drawback of the double description method is that it suffers from combinatorial explosion in high dimensional spaces. For instance, the n-cube $[0, 1]^n$ is characterized by $2n$ constraints, but its generator system contains 2^n vertices, which leads to a representation that grows exponentially with n.

From a mathematical point of view, the geometrical structure of a convex polyhedron is precisely described by its face lattice, which corresponds to a partial ordering of its faces. The double description method can actually be seen as an explicit representation of the non trivial top and bottom layers of this face lattice. Another strategy is to keep a representation of the whole face lattice of polyhedra, which has the advantage of providing complete information about the adjacency relation between their faces. This information makes it possible, in particular, to remove redundant constraints and elements of the generator system in polynomial time [3].

A data structure that explicitly represents the face lattice is the Real Vector Automaton, whose expressive power goes beyond first-order additive arithmetic of mixed integer and real variables [8]. When it represents a convex polyhedron, an RVA is essentially a deterministic decision graph for determining which face contains a given point. RVA have the advantage of being easily reducible to a minimal canonical form, which makes the representation of a set independent from its construction history. Nevertheless, their size grows linearly with the coefficients of linear constraints, and they suffer from the same combinatorial explosion as the double description method. The former drawback is alleviated by the Implicit Real Vector Automaton (IRVA) [14] and the Convex Polyhedron Decision Diagram (CPDD)[7], in which parts of the decision graph are encoded by more efficient algebraic structures.

Our goal is to make CPDDs efficient in high dimensional spaces. In order to deal with the combinatorial explosion of the generator system, a decomposition mechanism for convex polyhedra has been proposed in [17]. The approach consists in partitioning syntactically the variables involved in the linear constraints into independent subsets. Roughly speaking, convex polyhedra are decomposed into Cartesian products of simpler ones defined over disjoint subsets of variables.

This procedure has the disadvantage of being unable to handle efficiently constraints that jointly involve many variables, which makes it ill-suited for our intended applications. During the reachability analysis of timed automata for instance, applying a time-step operation to a polyhedron will generally produce constraints linking together all clock variables, making decomposition unfeasible.

The contributions of this work are twofold. First, by keeping an explicit representation of the face lattice of polyhedra, we obtain a significant advantage over the double description method, leading in particular to a more efficient implementation of the projection operation. Second, we tackle the combinatorial explosion in high dimensional spaces by introducing a novel decomposition mechanism. As opposed to the purely syntactic approach of [17], this mechanism is not affected by non-singular linear transformations, which significantly broadens its applicability. The resulting data structure, the Decomposed Convex Polyhedron (DCP), admits an easily computable canonical form, which simplifies comparison operations and leads to concise representations of simple sets constructed in a complex way. DCPs share the same advantages as CPDDs, such as offering a simple decision procedure for checking which face of a convex polyhedron contains a given point.

The rest of this paper is organized as follows. Section 2 recalls basic concepts and the principles of the double description method. Section 3 introduces DCPs, starting from CPDDs and enhancing them with a decomposition mechanism. Section 4 discusses the implementation of operations over DCPs. Section 5 assesses the practical efficiency of our proposed data structure with the help of a prototype implementation.

2 Preliminaries

2.1 Basics

A *convex polyhedron* P is defined as the set of solutions of a finite conjunction of linear constraints, i.e., $P = \{\boldsymbol{x} \in \mathbb{R}^n \mid \bigwedge_{i=1}^{k} \boldsymbol{a_i}.\boldsymbol{x} \#_i b_i\}$ where, for all i, $\boldsymbol{a_i} \in \mathbb{Z}^n$, $b_i \in \mathbb{Z}$, and $\#_i \in \{\leq, <\}$. Such polyhedra can either be bounded or unbounded, as well as topologically closed or not[1]. We denote by $\overline{P}$ the topological closure of P, that is, the set $\overline{P} = \{\boldsymbol{x} \in \mathbb{R}^n \mid \bigwedge_{i=1}^{k} \boldsymbol{a_i}.\boldsymbol{x} \leq b_i\}$.

Given a constraint $\boldsymbol{a_i}.\boldsymbol{x} \#_i b_i$, a point $\boldsymbol{v} \in \mathbb{R}^n$ *satisfies* this constraint if $\boldsymbol{a_i}.\boldsymbol{v} \#_i b_i$, and *saturates* it if $\boldsymbol{a_i}.\boldsymbol{v} = b_i$. Constraints of the form $\boldsymbol{a_i}.\boldsymbol{x} \leq b_i$ are called *closed*, or *non-strict*, and constraints of the form $\boldsymbol{a_i}.\boldsymbol{x} < b_i$ are called *open*, or *strict*. The *dimension* of a convex polyhedron P, noted $\dim P$, is the dimension of its affine hull, i.e the smallest affine space that contains P. The *lineality space* $\operatorname{lin} P$ of P is the largest vector space L such that $P + L = P$, where $+$ denotes the Minkowski sum.

A closed convex polyhedron $\overline{P}$ can be represented as a finite intersection of halfspaces by its *constraint system* $\mathcal{H} = \{\boldsymbol{a_i}.\boldsymbol{x} \leq b_i\}$. Alternatively, $\overline{P}$ can be expressed in terms of a *generator set* $\mathcal{G} = (V, R)$, where V, $R \in \mathbb{Q}^n$ are finite sets

[1] They are also known as NNC (Not Necessarily Closed) polyhedra, or copolyhedra.

of (respectively) *vertices* and *extremal rays.* One then has $P = \{\sum_{i=1}^{p} \lambda_i \boldsymbol{v_i} + \sum_{i=1}^{q} \mu_i \boldsymbol{r_i}\}$, where $V = \{\boldsymbol{v_1}, \dots, \boldsymbol{v_p}\}$, $R = \{\boldsymbol{r_1}, \dots, \boldsymbol{r_q}\}$, λ_i, $\mu_i \geq 0$ for all i, and $\sum_{i=1}^{p} \lambda_i = 1$. The pair $(\mathcal{H}, \mathcal{G})$ forms the *double description* of $\overline{P}$ [22, 11].

2.2 Face Lattice of a Polyhedron

With respect to a polyhedron P, a linear inequality $\boldsymbol{c}.\boldsymbol{x} \leq \delta$ is said to be *valid* if it is satisfied by all $\boldsymbol{x} \in \overline{P}$. A *face* of P is any set F such that $F = \overline{P} \cap \{\boldsymbol{x} \mid \boldsymbol{c}.\boldsymbol{x} = \delta\}$, where $\boldsymbol{c}.\boldsymbol{x} \leq \delta$ is valid. Note that from the valid inequalities $\boldsymbol{0}.\boldsymbol{x} \leq 0$ and $\boldsymbol{0}.\boldsymbol{x} \leq 1$, we get that $\overline{P}$ and the empty set $\emptyset$ are both faces of P. These two faces are said to be *trivial.* Note that a face is itself a polyhedron; the *dimension* of a face is its dimension as a polyhedron. The faces of dimension 0, 1, and $\dim P - 1$ are respectively called *vertices*, *edges*, and *facets.* Remark that the intersection of any set of faces of P is itself a face of P.

A *partial order* $\preceq$ over a set S is a binary relation that is reflexive, antisymmetric and transitive. We then say that $(S, \preceq)$, or simply S if the partial order is clear from the context, is a *partially ordered set.* A partially ordered set S is a *lattice* if every two elements $x, y \in S$ admit a unique minimal upper bound in S, called the *join* $x \sqcup y$, and a unique maximal lower bound in S, called the *meet* $x \sqcap y$.

The set $\mathcal{F}'(P)$ of nonempty faces of P is partially ordered by set inclusion. However, this set does not necessarily contain a minimum element, hence we define the smallest face of P as the intersection $F_0 = \cap_{F \in \mathcal{F}'(P)} F$ of all its nonempty faces. The set $\mathcal{F}(P) = \{F_0\} \cup \mathcal{F}'(P)$ is a finite lattice under set inclusion, called the *face lattice* of P.

For F, $G \in \mathcal{F}(P)$, the face $F \sqcup G = \cap\{H \in \mathcal{F}(P) \mid F \cup G \subseteq H\}$, is the smallest one containing both F and G. Similarly, the face $F \sqcap G = F \cap G$ is the largest one contained in both F and G. Furthermore, we say that F is an *ascendant* of G, or equivalently that G is a *descendant* of F, if $F \subset G$. We use the terms *direct ascendant* and *direct descendant* if there does not exist $H \in \mathcal{F}(P)$ such that $F \subset H \subset G$.

2.3 Canonical Representation of Convex Polyhedra

In the double description $(\mathcal{H}, \mathcal{G})$ of a closed convex polyhedron $\overline{P}$, the constraint system $\mathcal{H}$ and the generator system $\mathcal{G}$ admit minimal forms, meaning that no element can be removed from them without affecting $\overline{P}$.

Under two hypotheses, the minimal forms of $\mathcal{H}$ and $\mathcal{G}$ are unique for a given $\overline{P}$, which implies that $(\mathcal{H}, \mathcal{G})$ can then provide a canonical representation of $\overline{P}$. The first hypothesis is to have a *fully dimensional* polyhedron, meaning that $\overline{P} \subseteq \mathbb{R}^n$ is such that $\dim \overline{P} = n$. If $\overline{P}$ is not fully dimensional, then it can be expressed as the image $\overline{P} = A\,\overline{Q} + \boldsymbol{b}$ of a fully dimensional polyhedron $\overline{Q} \subset \mathbb{R}^m$ of smaller dimension $m < n$ by a linear transformation $(A, \boldsymbol{b})$, with $A \in \mathbb{Z}^{n \times m}$ and $\boldsymbol{b} \in \mathbb{Q}^n$. This transformation can be made canonical by Gaussian elimination.

The second hypothesis is to have a polyhedron $\overline{P}$ with a lineality space of dimension 0. If this condition is not satisfied, then $\overline{P}$ can be expressed as a sum

$\overline{P} = \overline{Q} + L$, where $\operatorname{lin} \overline{Q} = 0$, $L = \operatorname{lin} \overline{P}$, and $\dim \overline{P} = \dim \overline{Q} + \dim L$. The vector space L can be described canonically by applying Gaussian elimination to one of its bases.

Consider a polyhedron $\overline{P}$ that satisfies both hypotheses, for which the double description $(\mathcal{H}, \mathcal{G})$ has been made minimal. This means that all redundant constraints have been removed from $\mathcal{H}$, hence the saturated form of each constraint in $\mathcal{H}$ is a facet. More generally, each face of $\overline{P}$ corresponds to a subset of saturated constraints in $\mathcal{H}$. Similarly, the vertices of $\mathcal{G}$ correspond to the minimal non-trivial faces of $\overline{P}$. If $\overline{P}$ is bounded, then $\mathcal{G}$ does not contain extremal rays, and the double description $(\mathcal{H}, \mathcal{G})$ exactly contains the minimal and maximal non-trivial elements of the face lattice of $\overline{P}$. If $\overline{P}$ is unbounded, the extremal rays of $\mathcal{G}$ can be computed from the direct descendants of the vertices.

3 Decomposed Convex Polyhedra

We now present our proposed data structure, by first introducing the principles of CPDDs, and then enhancing them with a decomposition mechanism.

3.1 Convex Polyhedron Decision Diagram

A *Convex Polyhedron Decision Diagram* (CPDD) [7] representing a convex polyhedron P is a directed acyclic graph (Q, T, q_0) such that:

- Q is a finite set of *nodes*. Each node $q \in Q$ corresponds to a face of P, and is labeled by the constraints of P that are saturated by that face. (In the special case where q represents the empty face, all constraints are considered to be saturated.) Moreover, q is associated with a binary *polarity* that is true if each constraint that is saturated by q is an open constraint of P, and false otherwise. This polarity is used for representing the strictness of constraints; the representations of P and $\overline{P}$ only differ in the polarity of their nodes.
- $q_0 \in Q$ is an *initial node*, representing the unique minimal element of the face lattice of P.
- $T \subseteq Q \times Q$ is a transition relation corresponding to the inclusion relation between faces, removing the edges that are redundant by transitivity. An edge $(q_1, q_2) \in T$ is labeled by the constraints that are saturated in q_1 but not in q_2.

An example of a CPDD is given in Figure 1. This data structure can be seen as a deterministic decision graph for determining which face of P contains a given point $\boldsymbol{v} \in \mathbb{R}^n$. This operation consists in starting from the initial node, and then following edges labeled by constraints satisfied by $\boldsymbol{v}$. The procedure ends either upon reaching a node q labeled by constraints saturated by $\boldsymbol{v}$, in which case q represents the face of P containing $\boldsymbol{v}$, and the polarity of q indicates whether $\boldsymbol{v}$ belongs to P, or when no outgoing edge can be followed from the current node, corresponding to $\boldsymbol{v} \notin P$. Note that if several paths can be followed

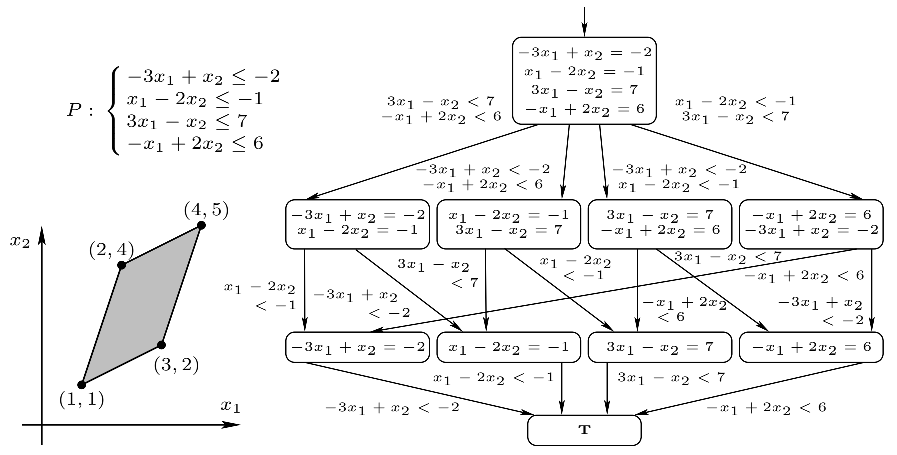

Fig. 1: Example of CPDD.

from a given node, one of them can be chosen arbitrarily without the need for backtracking, since for every pair of nodes q_1, $q_2 \in Q$, all paths linking q_1 to q_2 are labeled with the same constraints. Intuitively, a CPDD can be understood as a compact representation of a deterministic finite automaton accepting the points of a convex polyhedron [8, 7].

3.2 Decomposition of Convex Polyhedra

Like the double description method, CPDDs suffer from combinatorial explosion in high dimensional spaces. For instance, a simple polyhedron such as the n-cube $[0, 1]^n$ has 2^n vertices, which makes its representation grow exponentially with the dimension n.

In this example, each constraint involves a single variable. In order to check whether a given point $\boldsymbol{p} = (p_1, \ldots, p_n)$ belongs to the cube, one can separately check that each p_i is inside $[0, 1]$. This essentially amounts to decomposing the n-cube into a Cartesian product of intervals, that can be processed individually. This idea is developed in [17], which shows how to determine syntactically blocks of variables that can be considered independently from each other.

This approach is however not sufficient for handling the reachable sets computed by state-space exploration tools. In particular, the analysis of timed automata often produces constraints that involve all variables, expressing that they share an identical rate of variation with time. Another example is given by the polyhedron in Figure 1, which depicts a typical region obtained during the state-space exploration of a linear hybrid system.

In this latter example, one notices however that the polyhedron can become decomposable into a Cartesian product of two intervals by expressing it in a different coordinate system, for instance the one defined by the basis $\{(2, 1), (1, 3)\}$.

The idea behind our improved decomposition scheme is to detect whether a suitable coordinate system exists, that makes the polyhedron decomposable into a Cartesian product of simpler ones. The main advantage of this strategy over a purely syntactic one is that the decomposability property of polyhedra remains unaffected by changes of coordinate system, or equivalently, by non-singular linear transformations (cf. Section 4.2).

We define a *decomposition* of a finite set of vectors $S \subset \mathbb{R}^n$ as a partition of S into blocks, such that:

– If a block B contains at least two elements, then each of them can be written as a linear combination of the other ones. Formally, if $B = \{\boldsymbol{b_1}, \ldots, \boldsymbol{b_k}\}$ with $k \geq 2$, then

$$\forall i \in [1,k] : \exists \beta_1 \ldots, \beta_k \in \mathbb{R}^n : \boldsymbol{b_i} = \sum_{j \in [1,k],\, j \neq i} \beta_j \boldsymbol{b_j}.$$

– For each block B, there does not exist a non-zero linear combination of the elements of B that can be written as a linear combination of the elements of the other blocks. Formally, if $B = \{\boldsymbol{b_1}, \ldots, \boldsymbol{b_k}\}$, then

$$\sum_{\boldsymbol{b_i} \in B} \beta_i \boldsymbol{b_i} = \sum_{\boldsymbol{b_i}' \in S \setminus B} \beta_i' \boldsymbol{b_i}' \;\Rightarrow\; \sum_{\boldsymbol{b_i} \in B} \beta_i \boldsymbol{b_i} = \mathbf{0}.$$

Intuitively, a decomposition of a set of vectors partitions this set into blocks that are linearly independent from each other. For example, the set $\{(1,1,1), (1,1,2), (-2,-2,-2), (1,-1,0), (0,1,1)\}$ admits the decomposition $\{\{(1,1,1), (-2,-2,-2)\}, \{(1,1,2), (1,-1,0), (0,1,1)\}\}$.

If P_1 and P_2 are two partitions of a set S, then P_1 is *finer* than P_2 (or, equivalently, P_2 is *coarser* than P_1) if every block of P_1 is a subset of some block of P_2. This notion generalizes to decompositions as follows.

Proposition 1. *If D_1 and D_2 are decompositions of a set S, then the partition*

$$D = D_1 \cap D_2 = \{B_i \cap B_j' \mid B_i \cap B_j' \neq \emptyset \;\wedge\; B_i \in D_1,\, B_j' \in D_2\}$$

is itself a decomposition.

This property naturally leads to a notion of *finest decomposition* of a set, obtained by computing the intersection of all its decompositions. This finest decomposition is, by definition, unique.

The finest decomposition of a given set S can be computed by an incremental procedure that considers successively all vectors $\boldsymbol{v}$ in S. At each step, one checks whether $\boldsymbol{v}$ can be expressed as a linear combination of the vectors that have already been dealt with. In the positive case, the blocks containing these vectors have to be merged into a single one, to which the vector $\boldsymbol{v}$ is added. Otherwise, a new block is created, containing only $\boldsymbol{v}$.

We are now ready to apply our notion of decomposition to polyhedra. The *canonical decomposition* of a convex polyhedron $P \subseteq \mathbb{R}^n$ is defined as the finest decomposition of the set of normal vectors of its bounding hyperplanes, that is, of the set $\{\boldsymbol{a_1}, \ldots, \boldsymbol{a_k}\}$ where $P = \{\boldsymbol{x} \in \mathbb{R}^n \mid \bigwedge_{i=1}^{k} \boldsymbol{a_i}.\boldsymbol{x} \,\#_i\, b_i\}$.

Let $D = \{B_1, \ldots, B_k\}$ be the canonical decomposition of P. In order to express P in a coordinate system in which it can be decomposed into a Cartesian product of simpler polyhedra, one builds a new basis of $\mathbb{R}^n$ by computing individual bases for the blocks $B_1, \ldots, B_k$, and then taking their union. The next step is to perform a coordinate change by expressing the constraints of P in terms of this new basis. This operation will turn a constraint $\boldsymbol{a_i}.\boldsymbol{x} \,\#_i\, b_i$ into $\boldsymbol{a_i}'.\boldsymbol{x} \,\#_i\, b_i'$, in which the components of $\boldsymbol{a_i}'$ are all zero, except for the ones provided by the basis of the block B_j containing $\boldsymbol{a_i}$. In other words, if B_j contains up to d linearly independent vectors, then the constraint $\boldsymbol{a_i}'.\boldsymbol{x} \,\#_i\, b_i'$ will only involve d variables. The change of coordinates induced by the canonical decomposition of P is the one that maximizes the possibility of separating syntactically the variables.

3.3 Decomposed Convex Polyhedron

A *Decomposed Convex Polyhedron* representing a polyhedron $P \subseteq \mathbb{R}^n$ is a tuple $(A, \boldsymbol{b}, q_0, C)$, where

- $(A, \boldsymbol{b})$ with $A \in \mathbb{Z}^{n \times m}$ and $\boldsymbol{b} \in \mathbb{Q}^n$ is a linear transformation such that $P = A\,Q + \boldsymbol{b}$, where $Q \in \mathbb{R}^m$ is a fully dimensional polyhedron (cf. Section 2.3).
- q_0 is an initial node labeled with the canonical decomposition D of Q, and its associated change of coordinates.
- C is a finite set of CPPDs, each of them being associated to an element of D. One thus has $|C| = |D|$. The transition from q_0 to an element of C is called a *decomposition branch*. Each decomposition branch is labeled by its corresponding variables in the new coordinate system.

$$P : \begin{cases} -3x_1 + x_2 \leq -2 \\ x_1 - 2x_2 \leq -1 \\ 3x_1 - x_2 \leq 7 \\ -x_1 + 2x_2 \leq 6 \end{cases} = \begin{cases} -y_1 \leq -2 \\ y_2 \leq -1 \\ y_1 \leq 7 \\ -y_2 \leq 6 \end{cases}$$

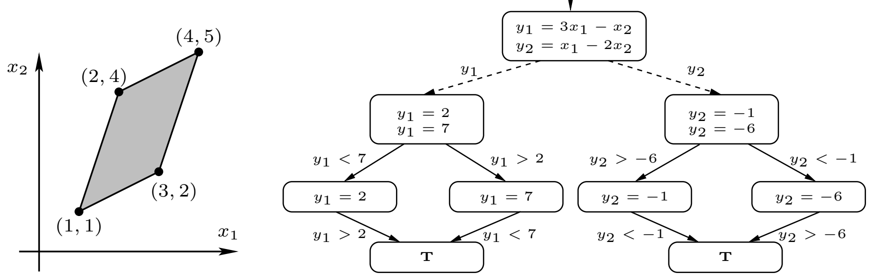

Fig. 2: Example of DCP.

An example of DCP is given in Figure 2. In this example, the represented polyhedron is fully dimensional, thus the transformation $(A, \boldsymbol{b})$ can be chosen as the identity relation, which we do not depict for clarity sake. Decomposition branches are denoted by dashed edges. The set of normal vectors of the constraints is $\{(-3, 1), (1, -2), (3, -1), (-1, 2)\}$, the canonical decomposition of which is $\{\{(-3, 1), (3, -1)\}, \{(1, -2), (-1, 2)\}\}$. The basis of $\mathbb{R}^2$ induced by this decomposition is $\{(3, -1), (1, -2)\}$.

In order to determine whether a given point $\boldsymbol{v} \in \mathbb{R}^n$ belongs to a polyhedron P represented by a DCP $(A, \boldsymbol{b}, q_0, C)$, the first step consists in computing $\boldsymbol{v}' \in \mathbb{R}^m$ such that $\boldsymbol{v} = A\boldsymbol{v}' + \boldsymbol{b}$. If no such vector exists, then the answer is negative. Otherwise, the coordinate change associated to q_0 is applied to $\boldsymbol{v}'$, yielding vectors $\boldsymbol{y_1}, \ldots \boldsymbol{y_k}$ such that $k = |C|$ and $\dim \boldsymbol{v'} = \dim \boldsymbol{y_1} + \cdots + \dim \boldsymbol{y_k}$. One then runs the point location procedure described in Section 3.1 for one $\boldsymbol{y_i}$ in each of the k decomposition branches, all of which have to succeed in order to conclude that $\boldsymbol{v}$ belongs to P. Determining which face of P contains $\boldsymbol{v}$ amounts to combining together the faces reached in each decomposition branch. For example, in Figure 2, the point $\boldsymbol{v} = (2, 1.5)$ is found to belong to the universal (bottom) node in the branch labeled by y_1, and to the node $y_2 = -1$ in the one labeled by y_2. The corresponding face of P is thus $x_1 - 2x_2 = -1$.

Finally, it is worth mentioning that in the case of a polyhedron with a lineality space of non-zero dimension d, our decomposition strategy will produce d trivial decomposition branches, associated to the universal set. Such branches do not have to be explicitly constructed and can be omitted in an actual implementation of the data structure.

4 Operations

4.1 Intersection

We now discuss the computation of operations over convex polyhedra represented by DCPs, starting with the intersection $P_1 \cap P_2$ of two given polyhedra P_1 and P_2. These polyhedra may define different decompositions. We go around this problem by proceeding incrementally, starting from P_1 and successively intersecting the polyhedron with each constraint of P_2.

Dealing with Decompositions In order to intersect a polyhedron P with a constraint $\boldsymbol{c}.\boldsymbol{x} \# \delta$, the first step consists in inserting $\boldsymbol{c}$ in the current decomposition of P, following the procedure outlined in Section 3.2. If $\boldsymbol{c}$ is placed in a single existing branch, or in a newly created one, then the intersection can be computed locally over the CPDD associated to this branch. Otherwise, if several decomposition blocks become merged, then a single CPDD corresponding to the Cartesian product of their associated branches first needs to be constructed. The intersection operation is then computed over this CPDD, leaving the other decomposition branches untouched. Then, after having intersected a CPDD with a constraint, the result is inspected in order to detect whether it is further decomposable. This is achieved by applying the procedure of Section 3.2 to its system of constraints. A final step is to check whether the resulting polyhedron is fully dimensional, which amounts to inspecting the bottom component of the CPDD of the branch affected by the intersection. Depending on the outcome of this operation, it may be needed to adapt the linear transformation of the DCP.

CPDD Intersection The intersection of a polyhedron P represented by a CPDD with a constraint $\boldsymbol{c}.\boldsymbol{x} \,\#\, \delta$ is computed by means of a coloring procedure, consisting in labeling the nodes of the CPDD with a color that indicates how they are affected by the operation.

Recall that the CPDD nodes correspond to the faces of P. The coloring scheme uses the colors Green, Red, Blue, and Yellow. A face F is colored Red if $\forall \boldsymbol{x} \in F : \boldsymbol{c}.\boldsymbol{x} \geq \delta \,\wedge\, \exists \boldsymbol{x} \in F : \boldsymbol{c}.\boldsymbol{x} > \delta$ (no point in F satisfies the open form of the constraint), Green if $\forall \boldsymbol{x} \in F : \boldsymbol{c}.\boldsymbol{x} \leq \delta \,\wedge\, \exists \boldsymbol{x} \in F : \boldsymbol{c}.\boldsymbol{x} < \delta$ (all points in F satisfy the closed form of the constraint), Blue if $\forall \boldsymbol{x} \in F : \boldsymbol{c}.\boldsymbol{x} = \delta$ (all points in F saturate the constraint), and Yellow if $\exists \boldsymbol{x}, \boldsymbol{y} \in F : \boldsymbol{c}.\boldsymbol{x} < \delta \,\wedge\, \boldsymbol{c}.\boldsymbol{y} > \delta$ (some points in F satisfy the constraint, and some others do not).

The color of all nodes can be computed by first coloring the minimal non-trivial faces of P, and then propagating this information through its face lattice. Consider for instance the case of a face F that has a direct ascendant F_1 labeled Green, and another one F_2 labeled Red. Thus, there exist $\boldsymbol{x_1} \in F_1$ such that $\boldsymbol{c}.\boldsymbol{x_1} < \delta$ and $\boldsymbol{x_2} \in F_2$ such that $\boldsymbol{c}.\boldsymbol{x_2} > \delta$. Since $F_1 \cup F_2 \subseteq F$, the face F must be colored Yellow.

Similar propagation rules are easily obtained for all cases, except for a technical difficulty arising when P is unbounded. In such a case it is possible for a face G to have a single direct ascendant F. In order to determine the color of G from the color of F, one then needs to take into account a direction $\boldsymbol{d}$ from F to G, defined as a vector satisfying $\forall \boldsymbol{x'} \in G : \exists \boldsymbol{x} \in F, \lambda \geq 0 : \boldsymbol{x'} = \boldsymbol{x} + \lambda \boldsymbol{d}$. This direction will be colored Green if it is compatible with the constraint ($\boldsymbol{c}.\boldsymbol{d} < 0$), Red if it is not ($\boldsymbol{c}.\boldsymbol{d} > 0$), and Blue if it saturates it ($\boldsymbol{c}.\boldsymbol{d} = 0$). It will then be considered as an additional ascendant of G. Intuitively, this direction simulates a face F' with the same dimension as F, located infinitely far away from F in the same direction as G.

After all nodes have been colored, a CPDD representing the result of the intersection is obtained as follows. All Green and Blue nodes remain unchanged, since they represent faces that satisfy $\boldsymbol{c}.\boldsymbol{x} \leq \delta$. Similarly, Red nodes disappear, since all points of their associated face violate the constraint. Yellow faces F are split into two new faces: A first one $F_1 = F \cap \{\boldsymbol{x} \mid \boldsymbol{c}.\boldsymbol{x} \leq \delta\}$ with the same dimension as F, and another one $F_2 = F \cap \{\boldsymbol{x} \mid \boldsymbol{c}.\boldsymbol{x} = \delta\}$ of smaller dimension. Note that F_1 is associated with the same set of saturated constraints as F, and can thus be considered as being a modified copy of F.

After having computed all the faces of the resulting polyhedron, it remains to restore the inclusion relation between them. For the nodes left untouched by the intersection operation, such as Green and Blue ones, this information can simply be copied from the original CPDD. For Yellow nodes, an additional step needs to be performed. Consider two Yellow nodes F and G such that F is a direct ascendant of G. The nodes F and G will respectively be split into F_1, F_2, and G_1, G_2, where F_2 (resp. G_2) is a direct ascendant of F_1 (resp. G_1). In this situation, one has $F_2 \subset G_2$, hence an edge needs to be added linking F_2 to G_2. A similar phenomenon occurs when a Blue face F has a Green direct descendant G, that has a Yellow direct descendant H. The node H is split into H_1 and H_2 with $\dim H_2 < \dim H_1$. In this case, one has $F \subset H_2$, hence an edge must be added between those nodes. These two situations are illustrated in Figure 3 (added edges are in bold).

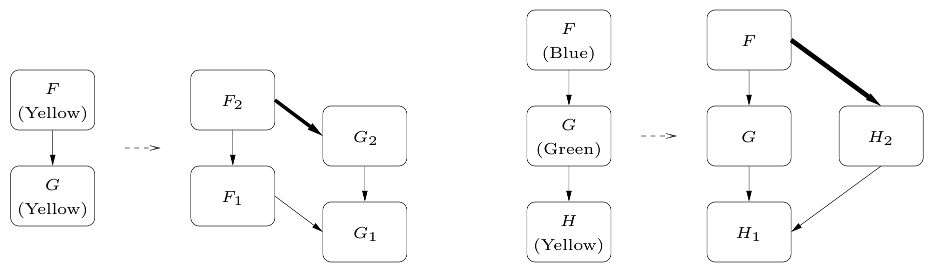

Fig. 3: Restoring the inclusion relation between faces.

A pseudocode version of the CPDD intersection algorithm is sketched in Figure 4.

Implementation Issues In our implementation, the coloring procedure is implemented lazily, meaning that it only considers the nodes that will potentially be present in the resulting CPDD. In particular, the descendants of a Red node will not be explored, except if they are reached by another path. When several decomposition branches need to be merged upon processing a new constraint, the CPDD representing their Cartesian product is not explicitly constructed but computed on-the-fly, which helps keeping the memory used by the procedure under control. Finally, we keep a canonical double representation of the maximal and non-trivial minimal faces of polyhedra within their respective nodes of their face lattice. This information makes it possible to speed up the computation of the color of minimal faces, as well as the check for full dimensionality.

4.2 Linear Transformations

We now address the problem of computing the image of a convex polyhedron $P \subseteq \mathbb{R}^n$ represented by a DCP by an affine transformation $\pi : \boldsymbol{x} \mapsto A\boldsymbol{x} + \boldsymbol{b}$, with $A \in \mathbb{Q}^{n \times n}$ and $\boldsymbol{b} \in \mathbb{Q}^n$.

There are two cases to consider. First, if A is a non-singular matrix, then applying the transformation amounts to expressing P in a new coordinate system. The decomposition of P and the structure of its face lattice are thus left

```
Color(node N, constraint C):
   if N is minimal:
      compute N.color from the saturated constraints of N
   else:
      S = set of colors of the direct ascendants of N
      if S == { Green } or S == { Green, Blue }: N.color = Green
      if S == { Red } or S == { Red, Blue }: N.color = Red
      if S == { Blue}: N.color = Blue
      if Yellow in S: N.color = Yellow
      if Green in S and Red in S: N.color = Yellow

SplitYellowNodes(node N, constraint C):
   if N.color == Yellow:
      add new node N2 and new edge from N2 to N
   for all direct ascendants M of N:
      if M.color == Yellow:
         add new edge from M to N2
      if M.color == Green:
         for all direct ascendants L of M:
            if L.color == Blue:
               add new edge from L to N2

Intersect(CPDD P, constraint C):
   for all N:
     N.color = undefined
   FIFO Queue Q = { }
   Q.put(P.initialNode)
   while not IsEmpty(Q):
      Node N = Q.get()
      if N.color == undefined:
        Color(N, C)
        if N.color != Red or MinimalNonTrivial(N):
           for all direct descendants M of N:
              Q.put(M)
   Q.put(P.initialNode)
   while not IsEmpty(Q)
      Node N = Q.get()
      SplitYellowNodes(N, C)
      for all direct descendants M of N:
         Q.put(M)
   CheckFullDimensionality(P)
```

Fig. 4: CPDD intersection algorithm.

unchanged, and the operation can be implemented by translating the constraints of P in the new coordinate system, and then updating the labels of the nodes and edges of the DCP accordingly.

If on the other hand A is singular, then the transformation represents a *projection*, mapping P into a polyhedron $P' = \pi(P)$ such that $\dim P' = \mathrm{rank}(A)$, hence $\dim P' < \dim P$. It is well known that P' is itself a convex polyhedron, and that for each face F' of P', there exists a face F of P such that $\pi(F) = F'$. Moreover, the face lattice of P' shares the same structure as the one of P.

W.l.o.g., we assume $\mathrm{rank}(A) = n - 1$, since any projection can easily be expressed as a sequence of projections that satisfy this hypothesis. The first step of the computation consists in checking whether the decomposition of the DCP can be preserved. This is done by computing a *direction* for the projection, defined as a vector $\boldsymbol{d}$ that satisfies $A\boldsymbol{d} = 0$. This intuitively means that two points that only differ in a multiple of this direction are projected identically. In the current decomposition, all the branches that are not orthogonal to $\boldsymbol{d}$ (i.e., containing a vector $\boldsymbol{a}$ such that $\boldsymbol{a}.\boldsymbol{d} \neq 0$) must be merged together. The projection is then applied separately to the CPDD associated to each branch.

Consider a CPDD representing a polyhedron $P \subseteq \mathbb{R}^n$. The computation of its projection by π proceeds bottom-up in its face lattice, as opposed to the intersection operation that was carried out in top-down order. We start by projecting the trivial face of dimension n (corresponding to the whole polyhedron P). This projection yields the trivial face $P' = \pi(P)$, of dimension $n - 1$.

The next step consists in projecting the following two layers, that is, the facets of P (of dimension $n - 1$), and their direct ascendants (of dimension $n - 2$). The projection $\pi(F)$ of a facet F of P may either be of dimension $n - 1$ or $n - 2$. In the former case, it corresponds to the unique trivial face of dimension $n - 1$ of P'. In the latter, the set $\pi(F)$ needs to be explicitly computed.

The projection $\pi(F)$ of a face F of P such that $\dim F = n - 2$ can either be a face of P' (of dimension $n - 2$ or $n - 3$), or it will not be a face of P'. These situations are distinguished by performing Fourier-Motzkin elimination. This is illustrated in Figure 5, the two parts of which show a vertex (of dimension 0) that respectively remains as a face of P', or vanishes after projecting out x_2.

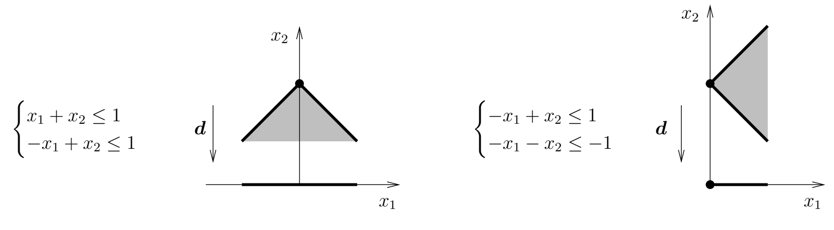
$\begin{cases} x_1 + x_2 \leq 1 \\ -x_1 + x_2 \leq 1 \end{cases}$
$\boldsymbol{d}$
x_2
x_1
$\begin{cases} -x_1 + x_2 \leq 1 \\ -x_1 - x_2 \leq -1 \end{cases}$
$\boldsymbol{d}$
x_2
x_1

Fig. 5: Effect of Fourier-Motzkin projection.

The major difference with the double description method [22, 11] is that we only apply Fourier-Motzkin elimination to the constraints that intersect at the face of interest, which are readily determined using the adjacency relation represented in the face lattice. This is the key to the efficiency of our procedure.

After having projected the first two non-trivial layers, it remains to compute the projection of the other faces. Since the face lattice of P' matches the one of P, this is simply done by following the structure of this lattice in bottom up order, computing each face as the meet (thus, the intersection), of its direct descendants. The resulting representation of P' is, by construction, free from redundancy in its constraint and generator systems.

Finally, the computation of the projection of a DCP is followed by a cleanup step aimed at detecting further decompositions and checking full dimensionality. This step is identical to the last operation of the insersection algorithm presented in Figure 4.

5 Experimental Results

In order to assess the advantages of DCPs against other solutions for dealing with convex polyhedra, we have implemented a prototype tool that builds the minimal DCP representing a polyhedron given by its set of constraints. Unsurprisingly, other tools based on the double description method do not come with benchmarks containing problems expressed in high dimensional spaces. Our first idea was to construct a set of examples composed of polyhedra that are decomposable by design. Our implementation handles them in an exponentially faster way than the other tools that we have considered, but this was expected since these examples were specifically tailored to our decomposition mechanism.

Obtaining instances of realistic problems related to the state-space exploration of hybrid systems, which was the main motivation for this work, is not easy since to the best of our knowledge, no existing tool can handle high-dimensional problems. We therefore turned to the domain of SMT solving, for which extensive benchmarks of problems involving a large number of variables are available. Our approach consisted in running the SMT prover veriT [10] on the verification problem `uart-9.base`[2] from the QF_LRA benchmark of the SMT-LIB library [6]. During its operation, the SMT prover generates systems of linear inequalities that are checked for satisfiability by an external simplex procedure. We replaced this procedure by an explicit construction of a DCP representing the corresponding convex polyhedron.

The results of this experimental evaluation are summarized in Figure 6. We compare the execution time (in seconds) of our prototype implementation against cdd [16] and PPL [4], which are based on the double description method with some clever optimizations, as well as lrs [2], which implements the reverse search algorithm for computing the vertices of polyhedra. The experiments were carried out on a computer equipped with a i7-970 processor running at 3.2 GHz, with

[2] The test cases are available at `http://www.montefiore.ulg.ac.be/~boigelot/research/atva2018-case-study.tgz`.

turbo boost disabled. Timeout was set at one hour. The indices 1, 50, 100, ... of the instances correspond to the steps at which these problems were produced by veriT (selected arbitrarily), and not to the increasing value of some parameter. In this setting, the results show that our approach (DCP) compares quite favorably against the other tools.

Problem	#Var.	#Constr.	DCP	PPL	cdd	lrs
dump-1	80	171	0.036	0.019	0.157	0.041
dump-50	129	402	0.091	0.158	1.306	0.240
dump-100	141	482	0.111	1006.717	2.014	0.350
dump-150	153	561	0.123	timeout	2.066	0.490
dump-200	166	639	0.148	timeout	4.458	0.650
dump-250	171	732	0.163	timeout	5.401	0.899

Fig. 6: Experimental results (times in s).

6 Conclusions

This paper introduces a new data structure, the Decomposed Convex Polyhedron (DCP), for representing symbolically convex polyhedra in $\mathbb{R}^n$. This data structure is based on an explicit representation of the whole face lattice of polyhedra, including complete adjacency information between its faces, which makes some operations (such as projection) more efficient. It is able to scale up to high dimensional spaces thanks to a novel decomposition mechanism that is not affected by changes of coordinates. DCPs have been evaluated experimentally with a prototype implementation. On an SMT solving case study related to software verification, they perform better than other existing tools for handling convex polyhedra.

Future work will focus on implementing additional operations on DCPs, such as the time-elapse operator needed for exploring the state-space of linear hybrid systems, and on improving our prototype with some optimization mechanisms borrowed from other tools. The practical cost of operations performed over DCPs also needs to be thoroughly evaluated in the scope of a more detailed case study.

Acknowledgment

The authors wish to thank Pascal Fontaine and Laurent Poirrier for their precious help in obtaining relevant benchmarks.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science **126**(2), 183–235 (1994)

2. Avis, D.: A revised implementation of the reverse search vertex enumeration algorithm. Polytopes — Combinatorics and Computation pp. 177–198 (2000)
3. Bachem, A., Grötschel, M.: Characterizations of adjacency of faces of polyhedra. Mathematical Programming at Oberwolfach pp. 1–22 (1981)
4. Bagnara, R., Hill, P.M., Zaffanella, E.: The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. Science of Computer Programming **72**(1–2), 3–21 (2008)
5. Bagnara, R., Hill, P.M., Zaffanella, E.: Applications of polyhedral computations to the analysis and verification of hardware and software systems. Theoretical Computer Science **410**(46), 4672 – 4691 (2009)
6. Barrett, C., Stump, A., Tinelli, C.: The SMT-LIB Standard: Version 2.0. In: Proc. SMT'10 (2010)
7. Boigelot, B., Herbreteau, F., Mainz, I.: Acceleration of affine hybrid transformations. In: Proc. ATVA'14. LNCS, vol. 8837, pp. 31–46. Springer (2014)
8. Boigelot, B., Jodogne, S., Wolper, P.: An effective decision procedure for linear arithmetic over the integers and reals. ACM Transactions on Computational Logic **6**(3), 614–633 (2005)
9. Bournez, O., Maler, O., Pnueli, A.: Orthogonal polyhedra: Representation and computation. In: Proc. HSCC'99. LNCS, vol. 1569, pp. 46–60. Springer (1999)
10. Bouton, T., de Oliveira, D.C.B., Déharbe, D., Fontaine, P.: veriT: An open, trustable and efficient SMT-solver. In: Proc. CADE'09. pp. 151–156. LNCS, Springer (2009)
11. Chernikova, N.: Algorithm for finding a general formula for the non-negative solutions of a system of linear inequalities. USSR Computational Mathematics and Mathematical Physics **5**(2), 228 – 233 (1965)
12. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proc. POPL'77. pp. 238–252. ACM Press (1977)
13. Cousot, P., Halbwachs, N.: Automatic discovery of linear restraints among variables of a program. In: Proc. POPL'78. pp. 84–96. ACM (1978)
14. Degbomont, J.F.: Implicit Real-Vector Automata. Ph.D. thesis, Université de Liège (2013)
15. Frehse, G.: PHAVer: algorithmic verification of hybrid systems past HyTech. International Journal on Software Tools for Technology Transfer **10**(3), 263–279 (2008)
16. Fukuda, K.: cdd. `https://www.inf.ethz.ch/personal/fukudak/cdd_home/`
17. G. Singh, M.P., Vechev, M.: Fast polyhedra abstract domain. In: Proc. POPL'17. pp. 46–59. ACM (2017)
18. Halbwachs, N., Proy, Y., Raymond, P.: Verification of linear hybrid systems by means of convex approximations. In: Proc. SAS'94. LNCS, vol. 864, pp. 223–237. Springer (1994)
19. Halbwachs, N., Proy, Y.E., Roumanoff, P.: Verification of real-time systems using linear relation analysis. Formal Methods in System Design **11**(2), 157–185 (1997)
20. Jeannet, B., Miné, A.: Apron: A library of numerical abstract domains for static analysis. In: Proc. CAV'09. pp. 661–667. LNCS, Springer (2009)
21. Leverge, H., Wilde, D.: PolyLib. `http://www.irisa.fr/polylib/`
22. Motzkin, T.S., Raiffa, H., Thompson, G.L., Thrall, R.M.: The double description method, pp. 51–74. Princeton University Press (1953)
23. Schrijver, A.: Theory of linear and integer programming. Wiley (1999)