The Octahedron Abstract Domain ⋆

Robert Clarisó and Jordi Cortadella
Universitat Politècnica de Catalunya
Barcelona, Spain

Abstract. An interesting area in static analysis is the study of numeric properties. Complex properties can be analyzed using *abstract interpretation*, provided that an adequate abstract domain is defined. Each domain can represent and manipulate a family of properties, providing a different trade-off between the precision and complexity of the analysis. The contribution of this paper is a new numeric abstract domain called *octahedron* that represents constraints of the form $(\pm x_j \pm \ldots \pm x_k \geq c)$, where x_i are numerical variables such that $x_i \geq 0$. The implementation of octahedra is based on a new kind of decision diagrams called *Octahedron Decision Diagrams* (OhDD).

1 Introduction

Abstract interpretation [5] defines a generic framework for the static analysis of dynamic properties of a system. This framework can be used, for instance, to analyze termination or to discover invariants in programs automatically. However, each analysis requires the framework to be parametrized for the relevant domain of properties being studied, e.g. numerical properties.

There is a wide selection of numeric abstract domains that can be used to represent and manipulate properties. Some examples are intervals, octagons and convex polyhedra. Each domain provides a different trade-off between the precision of the properties that can be represented and the efficiency of the manipulation. An interesting problem in abstract interpretation is the study of new abstract domains that are sufficiently expressive to analyze relevant problems and allow an efficient implementation.

In this paper, a new numerical abstract domain called *octahedron* is described. This abstract domain can represent conjunctions of restricted linear inequalities of the form $(\pm x_j \pm \ldots \pm x_k \geq c)$, where x_i are numerical variables such that $x_i \geq 0$. A new kind of decision diagram called *Octahedron Decision Diagram* (OhDD) has been specifically designed to represent and manipulate this family of constraints efficiently. Several analysis problems can be solved using these constraints, such as the analysis of timed systems [1, 12], the analysis of string length in C programs [8] and the discovery of bounds on the size of asynchronous communication channels.

The remaining sections of the paper are organized as follows. Section 2 explains related work in the definition of numeric domains for abstract interpretation, and previous decision diagram techniques used to represent numerical constraints. Section 3 defines the numeric domain of octahedra, and section 4 describes the data structure and its operations. In section 5, some possible applications of the octahedron abstract domain are discussed, and some experimental results are provided. Finally, section 6 draws some conclusions and suggests some future work.

Abstraction	Cite	Properties	Example
Intervals	[5]	$k_1 \leq x \leq k_2$	$2 \leq x \leq 5$
Difference Bound Matrices (DBMs)	[7, 15]	$k_1 \leq x \leq k_2$ $x - y \leq k$	$1 \leq x \leq 3$ $x - y \leq 5$
Octagons	[16]	$\pm x \pm y \leq k$	$2 \leq x + y \leq 6$
Two variables per inequality	[22]	$c_1 \cdot x_1 + c_2 \cdot x_2 \geq k$	$2 \leq 3x - 2y \leq 5$
Octahedra	This paper	$\pm x_i \pm \ldots \pm x_k \geq k$	$x - y + z \geq 5$
Convex polyhedra	[6, 11]	$c_1 \cdot x_1 + \ldots + c_n \cdot x_n \geq k$	$x + 3y - 2z \geq 6$

Table 1. A comparison of numeric abstract domains based on inequality properties.

2 Related work

2.1 Numeric Abstract Domains

Abstract domain is a concept used to denote a computer representation for a family of constraints, together with the algorithms to perform the abstract operators such as union, intersection, widening or the transfer function. Several abstract domains have been defined for interesting families of numeric properties, such as *inequality* or *modulo* properties. The octahedron abstract domain belongs to the former category. Other abstract domains based on inequalities are *intervals*, *difference bound matrices*, *octagons*, *two-variables-per-inequality*, and *convex polyhedra*. An example of these abstract domains and their relation to octahedra can be seen in Table 1.

Intervals are a representation for constraints on the upper or lower bound of a single variable, e.g. ($k_1 \leq x \leq k_2$). Interval analysis is very popular due to its simplicity and efficiency: an interval abstraction for n variables requires $O(n)$ space, and all operations require $O(n)$ time in the worst case. *Octagons* are an efficient representation for a system of inequalities on the sum or difference of variable pairs, e.g. ($\pm x \pm y \leq k$) and ($x \leq k$). The implementation of octagons is a based on *difference bound matrices* (DBM), a data structure used to represent constraints on differences of variables, as in ($x - y \leq k$) and ($x \leq k$). Efficiency is an advantage of this representation: the spatial cost for representing constraints on n variables is $O(n^2)$, while the temporal cost is between $O(n^2)$ and $O(n^3)$, depending on the operation. *Convex polyhedra* are an efficient representation for conjunctions of linear inequality constraints. This abstraction is very popular due to its ability to express precise constraints. However, this precision comes with a very high complexity overhead. This complexity has motivated the definition of abstract domains such as *two-variables per inequality*, which try to retain the expressiveness of linear inequalities with a lower complexity.

The abstract domain presented in this paper, *octahedra*, also attempts to keep some of the flexibility of convex polyhedra with a lower complexity. Instead of limiting the number of variables per inequality, the coefficients of the variables are restricted to $\{-1, 0, +1\}$. From this point of view, octahedra provide a precision that is between octagons and convex polyhedra.

2.2 Decision Diagrams

The implementation of octahedra is based on decision diagrams. Decision diagram techniques have been applied successfully to several problems in different application do-

mains. Binary Decision Diagrams (BDD) [3] provide an efficient mechanism to represent boolean functions. Zero Suppressed BDDs (ZDD) [14] are specially tuned to represent sparse functions more efficiently. Multi-Terminal Decision Diagrams (MTBDD) [10] represent functions from boolean variables to reals, $f : \mathbb{B}^n \rightarrow \mathbb{R}$

The paradigm of decision diagrams has also been applied to the analysis of numerical constraints. Most of this approaches compare the value of numeric variables with constants or intervals, or compare the value of pairs of variables. Some examples of these representations are *Difference Decision Diagrams* (DDD) [17], *Numeric Decision Diagrams* (NDD) [9], and *Clock Difference Diagrams* (CDD) [2]. These data structures encode contraints on *a maximum of two* variables at a time. In other representations, each node encodes one complex constraint like a linear inequality. Some examples of these representations are *Decision Diagrams with Constraints* (DDC) [13] and *Hybrid-Restriction Diagrams* (HRD) [24]. The Octahedron Decision Diagrams described in this paper use an innovative approach to encode linear inequalities. This approach is presented in Section 4.

3 Octahedra

3.1 Definitions

The octahedron abstract domain is now introduced. In the same way as convex polyhedra, an octahedron abstracts a set of vectors in $\mathbb{Q}^n$ as a system of linear inequalities satisfied by all these vectors. The difference between convex polyhedra and octahedra is the family of constraints that are supported.

Definition 1 (Unit linear inequality). *A linear inequality is a constraint of the form* $(c_1 \cdot x_1 + \ldots + c_n \cdot x_n \geq k)$ *where the constant term* k *and the coefficients* c_i *are in* $\mathbb{Q} \cup \{-\infty\}$*, e.g.* $(3x + 2y - z \geq -7)$*. A linear inequality will be called* unit *if all coefficients are in* $\{-1, 0, +1\}$*, such as* $(x + y - z \geq -7)$*.*

Definition 2 (Octahedron). *An octahedron* O *over* $\mathbb{Q}^n$ *is the set of solutions to the system of* m *unit inequalities* $O = \{X \mid AX \geq B \ \wedge \ X \geq 0^n\}$*, where* $B \in (\mathbb{Q} \cup \{-\infty\})^m$ *and* $A \in \{-1, 0, +1\}^{m \times n}$*. Octahedra satisfy the following properties:*

1. Convexity: *An octahedron is a convex set.*
2. Closed for intersection: *The intersection of two octahedra is also an octahedron.*
3. Non-closed for union: *The union of two octahedra might not be an octahedron.*

Figure 1(a) shows some examples of octahedra in two-dimensional space. In Fig. 1(b) there are several regions of space which are not octahedra, either because they contain a region with negative values (1), they are not convex (2), they cannot be represented by a finite system of linear inequalities (3), or because they can be represented as system of linear inequalities, but not unit linear inequalities (4). Notice that in two-dimensional space all octahedra are octagons; octahedra can only show a better precision than octagons in higher-dimensional spaces.

During the remaining of this paper, we will use C to denote a vector in $\{-1, 0, +1\}^n$ where n is the number of variables. Therefore, $(C^T X \geq k)$ denotes the unit linear inequality $(c_1 \cdot x_1 + \ldots + c_n \cdot x_n \geq k)$.

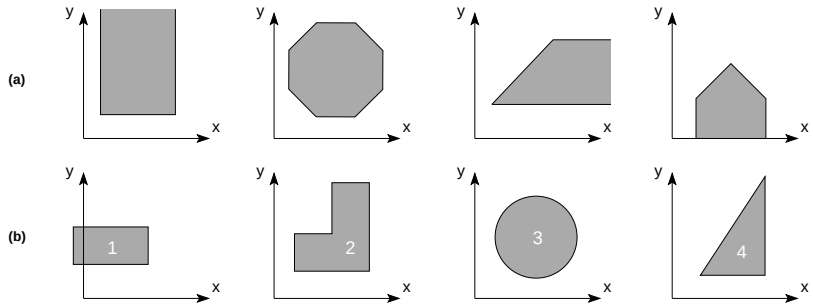


Fig. 1. Some examples of (a) octahedra and (b) non-octahedra in two-dimensional space.

Lemma 1. *An octahedron over n variables can be represented by at most 3^n non-redundant inequalities.*

Proof. Each variable can have at most three different coefficients in a unit linear inequality. These means that if an octahedron has more than 3^n unit inequalities, some of them will only differ in the constant term, e.g. $(C^T X \geq k_1)$ and $(C^T X \geq k_2)$. Only one of these inequalities is non-redundant, the one with the tightest bound (the largest constant), i.e. $(C^T X \geq \max(k_1, k_2))$. □

A problem when dealing with convex polyhedra and octahedra is the lack of canonicity of the systems of linear inequalities: the same polyhedron/octahedron can be represented with different inequalities. For example, both $(x = 3) \wedge (y \geq 5)$ and $(x = 3) \wedge (x + y \geq 8)$ define the same octahedron with different inequalities. Given a convex polyhedron, there are algorithms to minimize the number of constraints in a system of inequalities, i.e. removing all constraints that can be derived as linear combinations. However, in the previous example both representations are minimal and even then, they are different. Given that the number of possible linear inequalities in a convex polyhedron is infinite, the definition of a canonical form for convex polyhedra seems a difficult problem. However, a canonical form for octahedra can be defined using the result of lemma 1. Even though the number of inequalities of this canonical form makes an explicit representation impractical, symbolic representations based on decision diagrams can manipulate sets of unit inequalities efficiently.

Definition 3 (Canonical form of octahedra). *The canonical form of an octahedron $O \subseteq \mathbb{Q}^n$ is either (i) the empty octahedron or (ii) a system of 3^n unit linear inequalities, where in each inequality $(C^T X \geq k)$, k is the tightest bound satisfied by O.*

Theorem 1. *Two octahedra O_1 and O_2 represent the same subset of $\mathbb{Q}^n$ if and only if they both have the same canonical form.*

Proof. ($\rightarrow$) Given a constraint $(C^T X \geq k)$, there is a single tightest bound to that constraint. So if two octahedra are equal, they will have the same bound for each possible linear constraint, and therefore, the same canonical form. □

($\leftarrow$): From its definition, an octahedron is completely characterized by its system of inequalities. If two octahedra O_1 and O_2 have the same canonical form, then they satisfy exactly the same system of inequalities and therefore are equal. □

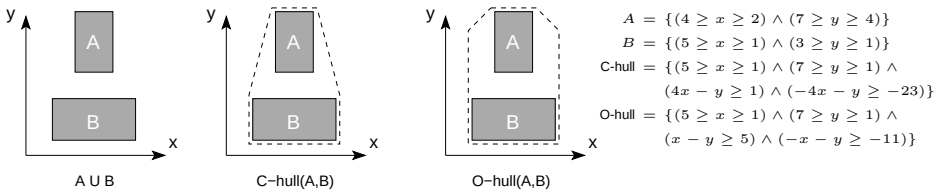


Fig. 2. Two upper approximations of the union: convex hull (C-hull) and octahedral hull (O-hull)

Theorem 2. *Let A and B be two non-empty octahedra represented by systems of inequalities of the form $(C^T X \geq k_a)$ and $(C^T X \geq k_b)$ for all $C \in \{-1, 0, +1\}^n$. The intersection $A \cap B$ is defined by the system of inequalities $(C^T X \geq \max(k_a, k_b))$, which might be in non-canonical form even if the input systems were canonical.*

Proof. Any point $P \in \mathbb{Q}^n$ that satisfies $(C^T P \geq \max(k_a, k_b))$ will also satisfy $(C^T P \geq k_a)$ and $(C^T P \geq k_b)$. Therefore, any point P satisfying the new system of inequalities will also appear in both A and B. □

Lemma 2. *An octahedron B is an upper approximation of an octahedron A, noted $A \subseteq B$, iff (i) A is empty or (ii) for any constraint $(C^T X \geq k_a)$ in the canonical form of A, the equivalent constraint $(C^T X \geq k_b)$ in the canonical form of B has a constant term k_b such that $(k_a \geq k_b)$.*

Proof. By definition, $A \subseteq B$ iff $A = A \cap B$. This lemma is a direct consequence of this property and Theorem 2. □

Definition 4 (Convex and octahedral hull). *The* convex hull (C-hull) *of two convex polyhedra A and B is the intersection of all convex polyhedra that include both A and B. The* octahedral hull (O-hull) *of two octahedra A and B is the intersection of all octahedra that include both A and B.*

Figure 2 shows an example of the convex and octahedral hulls of two octahedra A and B. Notice that the convex hull is always an upper approximation of the union, and the octahedral hull is always an upper approximation of the convex hull, i.e. $A \cup B \subseteq$ C-hull$(A, B) \subseteq$ O-hull(A, B).

Theorem 3. *Let A and B be two non-empty octahedra whose canonical form are respectively $(C^T X \geq k_a)$ and $(C^T X \geq k_b)$ for all $C \in \{-1, 0, +1\}^n$. Then, the octahedral hull O-hull(A, B) is defined by the system of inequalities $(C^T X \geq \min(k_a, k_b))$*

Proof. Given a bound k for one inequality $(C^T X \geq k)$ of O-hull(A, B), the proof can be split into two parts: proving that $k \leq \min(k_a, k_b)$ and proving that $k \geq \min(k_a, k_b)$.

As the octahedral hull includes A and B, all points $P \in A$ and $P \in B$ should also be in O-hull(A, B). Therefore, any point in A or B should satisfy the constraints of O-hull(A, B). Given a constraint $(C^T X \geq k)$, it is known that points in A satisfy $(C^T X \geq k_a)$ and points in B satisfy $(C^T X \geq k_b)$. If both sets of points must satisfy the constraint in O-hull(A, B), then k must satisfy $k \leq \min(k_a, k_b)$.

On the other side, the octahedral hull is the least octahedron that includes A and B. Therefore, the bounds of each constraint should be as tight as possible, i.e. as large

as possible. If we know that $k \leq \min(k_a, k_b)$ should hold for a given unit inequality, the tightest bound for that inequality is precisely $k = \min(k_a, k_b)$. As a **corollary**, the octahedral hull computed in this way is in canonical form. □

3.2 Abstractions of Octahedra

As it was shown in the previous section, the canonical form of an octahedron provides a useful mechanism to define operations such as the test for inclusion, the intersection or the octahedral hull. However, finding an *efficient* algorithm that can compute the canonical form of an octahedron from a non-canonical system of inequalities is an open problem at the time of writing this paper.

On the other hand, octahedra are defined in the context of abstract interpretation of numeric properties. In this context, the problem is the abstraction of a set of values in $\mathbb{Q}^n$, and the main concern is ensuring that our abstraction is an *upper approximation* of the concrete set of values. Thus, as long as an upper approximation can be guaranteed, an exact representation of octahedra is not required, as octahedra are already abstractions of more complex sets. Keeping this fact in mind, efficient algorithms that operate with upper approximations of the canonical form can be designed.

The first step is the definition of a relaxed version of the canonical form, which is called *saturated* form. While the canonical form has the tightest bound in each of its inequalities, the bounds in the saturated form may be more relaxed. A system of unit inequalities is in saturated form as long as the bounds imposed by the sum of any pair of constraints appear explicitly. For example, a saturated form of the octahedron $(a \geq 3) \wedge (b \geq 0) \wedge (c \geq 0) \wedge (b - c \geq 7) \wedge (a + b \geq 8) \wedge (a + c \geq 6)$ can be defined by the following system of inequalities:

$$(a \geq 3) \wedge (b \geq 7) \wedge (c \geq 0) \wedge (a + b \geq 10) \wedge (a + c \geq 6) \wedge (b + c \geq 7)$$
$$\wedge\, (b - c \geq 7) \wedge (a + b - c \geq 10) \wedge (a + b + c \geq 13)$$

where the constraints with a bound of $-\infty$ have been removed for brevity. In this example, saturation has exposed explicitly that $(a + b \geq 10)$. This inequality is the linear combination of $(a \geq 3)$, $(b - c \geq 7)$ and $(c \geq 0)$.

A saturated form O^* of an octahedron $O = \{X \mid AX \geq B \ \wedge \ X \geq 0^n\}$ can be computed using the following *saturation* procedure:

1. Initialize the system of 3^n unit inequalities for all possible values of the coefficients $C \in \{-1, 0, +1\}^n$. The bound k of a given inequality $(C^T X \geq k)$ is chosen as:

$$k = \begin{cases} \max(0, b) & \text{if } C^T X \geq b \text{ appears in } AX \geq B \text{ and } C \geq 0^n \\ b & \text{if } C^T X \geq b \text{ appears in } AX \geq B \text{ and } C \not\geq 0^n. \\ 0 & \text{if } C^T X \geq b \text{ does not appear in } AX \geq B \text{ but } C \geq 0^n \\ -\infty & \text{otherwise} \end{cases}$$

2. Select two inequalities $C_1^T X \geq k_1$ and $C_2^T X \geq k_2$ such that $k_1 > -\infty$ and $k_2 > -\infty$. Let us define $C_* = C_1 + C_2$ and $k_* = k_1 + k_2$.
3. If $C_* \notin \{-1, 0, +1\}^n$ return to step 2.
4. If $C_*^T X \geq k$ appears in the system of inequalities with $k \geq k_*$, return to step 2.

5. Replace the inequality $C_*^T X \geq k$ by $C_*^T X \geq k_*$.
6. Repeat steps 2-5 until:
   - A fixpoint is reached *or*
   - An inequality $C_*^T X \geq k$ with $C = 0^n$ and $k > 0$ is found. In this case, the octahedron is empty.

Theorem 4. *Let $O = \{X \mid AX \geq B \ \wedge \ X \geq 0^n\}$ be a non-empty octahedron. The* saturation *algorithm applied to O terminates.*

Proof. Each step of the saturation algorithm defines a tighter bound for an inequality of the octahedron. The new inequality $(C_3^T X \geq k'_3)$ is obtained from two previously known inequalities $(C_1^T X \geq k_1)$ and $(C_2^T X \geq k_2)$, so that $C_3 = C_1 + C_2$ and $k'_3 = k_1 + k_2$, and $k'_3 > k_3$, where k_3 is the previously known bound for the inequality. If inequalities 1 and 2 were computed in previous rounds of the saturation algorithm, this dependency chain can be expanded, e.g. if inequality 2 comes from inequalities 4 and 5, then $C_3 = C_1 + C_4 + C_5$ and $k'_3 = k_1 + k_4 + k_5$. Non-termination of the saturation algorithm implies that there will be infinitely many sums of pairs of inequalities. Ignoring the bound k, there are only finitely many inequalities over n variables. Therefore, it is always possible to find a step that computes a bound k'_j that depends on a previously known bound k_j, i.e. $C_j = C_j + \sum C_l$ and $k'_j = k_j + \sum k_l$. As $C_j - C_j = \sum C_l = 0^n$ and $k'_j - k_j = \sum k_l > 0$, the linear combination $((\sum C_l)^T X \geq (\sum k_l))$ is equivalent to $(0 > 0)$, which implies that O is empty. □

At each step, the saturation algorithm computes a new linear combination between two unit inequalities. If this linear combination has a tighter bound than the one already known, the bound is updated, and so on until a fixpoint is reached. Notice that this fixpoint *may not be reached* if the octahedron is empty. For example, the octahedron in Fig. 3(a) is empty because the sum of the last four inequalities is $(0 \geq 4)$. The saturation algorithm applied to this octahedron does not terminate. Adding the constraints in bottom-down order allows the saturation algorithm to produce $(x_2 - x_4 \geq 5)$, which can again be used to produce $(x_2 - x_4 \geq 9)$ and so on. Even then, the saturation algorithm is used to perform the emptiness test because of two reasons. First, there are special kinds of octahedra where termination is guaranteed. For instance, if all inequalities describe constraints between symbols (all constant term is zero), saturation is guaranteeed to terminate. Second, the conditions required to build an octahedron for which the saturation algorithm does not terminate are complex and artificial, and therefore they will rarely occur.

Even if the saturation algorithm terminates, in some cases it might fail to discover the tightest bound for an inequality. For example, in the octahedron in Fig. 3(b), saturation will fail to discover the constraint $(x_1 - x_2 + x_3 + x_4 + x_5 + x_6 \geq 6)$, as any sum of two inequalities will yield a non-unit linear inequality. Therefore, given a constraint $(C^T X \geq k_s)$ in the saturated form, the bound k_c for the same inequality in the canonical form may be different, $k_c \not\leq k_s$. But $k_c \geq k_s$ always holds, as k_c is the tightest bound for that inequality. Using this property, operations like the union or intersection that have been defined for the canonical form can also be used for the saturated form. The result will always be an upper approximation of the exact canonical result, as $k_c \geq k_s$ is the exact definition for upper approximation of octahedra (Lemma 2).

$$
\begin{array}{rcl}
+x_2 \quad -x_4 & \geq & 1 \\
-x_1 - x_2 + x_3 + x_4 + x_5 - x_6 & \geq & 1 \\
+x_1 - x_2 - x_3 + x_4 - x_5 + x_6 & \geq & 1 \\
+x_1 + x_2 + x_3 - x_4 - x_5 - x_6 & \geq & 1 \\
-x_1 + x_2 - x_3 - x_4 + x_5 + x_6 & \geq & 1
\end{array}
$$

(a)

$$
\begin{array}{l}
+x_1 - x_2 - x_3 + x_4 \geq 1 \\
-x_1 - x_2 + x_3 + x_5 \geq 2 \\
+x_1 + x_2 + x_3 + x_6 \geq 3
\end{array}
$$

(b)

Fig. 3. (a) Empty octahedron where the saturation algorithm does not terminate and (b) Non-empty octahedron where the saturated form is different from the canonical form.

3.3 Abstract Semantics of the Operators

In order to characterize the octahedron abstract domain, the abstract semantics of the abstract interpretation operators must be defined. Intuitively, this abstract semantics is defined as simple manipulations of the saturated form of octahedra. All operations are guaranteed to produce upper approximations of the exact result, as it was justified in section 3.2. Some operations like the intersection can deal with non-saturated forms without any loss of precision, while others like the union can only do so at the cost of additional over-approximation.

In the definition of the semantics, A and B will denote octahedra, whose saturated forms contain inequalities of the form $(C^T X \geq k_a)$ and $(C^T X \geq k_b)$, respectively.

- **Intersection** $A \cap B$ is represented by system of inequalities $(C^T X \geq \max(k_a, k_b))$, which might be in non-saturated form.
- **Union** $A \cup B$ is approximated by the saturated form $(C^T X \geq \min(k_a, k_b))$.
- **Inclusion** Let A and B be two octahedra. If $k_a \geq k_b$ for all inequalities in their saturated form, then $A \subseteq B$. Notice that the implication does not work in the other direction, i.e. if $k_a \not\geq k_b$ then we don't know whether $A \subseteq B$ or $A \not\subseteq B$.
- **Widening** $A \nabla B$ is defined as the octahedron with inequalities $(C^T X \geq k)$ such that k:

$$
k = \begin{cases} -\infty & \text{if } \ k_a > k_b \\ k_a & \text{otherwise} \end{cases}
$$

  As established in [16], the result should *not* be saturated in order to guarantee convergence in a finite number of steps.
- **Extension** An octahedron O can be extended with a new variable $y \geq 0$ by modifying the constraints of its saturated form O^*. Let $(c_1 \cdot x_1 + \ldots + c_n \cdot x_n \geq k)$ be a constraint of O^*, the inequalities that will appear in the saturated form of the extension are:
  - $c_1 \cdot x_1 + \ldots + c_n \cdot x_n - 1 \cdot y \geq -\infty$
  - $c_1 \cdot x_1 + \ldots + c_n \cdot x_n + 0 \cdot y \geq k$
  - $c_1 \cdot x_1 + \ldots + c_n \cdot x_n + 1 \cdot y \geq k$
- **Projection** A projection of an octahedron O removing a dimension x_i can be performed by removing from its saturated form O^* all inequalities where x_i has a coefficient that is not zero.

– **Unit linear assignment** A unit linear assignment $[x_i := \sum_{j=1}^{m} c_j \cdot x_j]$ with coefficients $c_i \in \{-1, 0, +1\}$ can be defined using the following steps:
  - Extend the octahedron with a new variable t.
  - Intersect the octahedron with the octahedron $(t = \sum_{j=1}^{m} c_j \cdot x_j)$
  - Project the variable x_i.
  - Rename t as x_i.

Impact of the conservative inclusion test on abstract interpretation: Using these operations, upper approximations of the concrete values will be computed in abstract interpretation. A special mention is the case of test of inclusion, where the result is only definite if the answer is true. Intuitively, this lack of accuracy appears from the impossiblity to discover the tightest bound with saturation. In abstract interpretation, the analysis is performed until a fixpoint is reached, and the fixpoint is detected using the test for inclusion. The inaccurate test of inclusion might lead to additional iterations in the abstract interpretation loop. Each iteration will add new constraints to our octahedra that were not being discovered by saturation, until the test for inclusion is able to detect the fixpoint. However, in practical examples, this theoretical scenario does not seem to arise, as constraints tend to be generated in a structured way that allows saturation to obtain good approximations of the exact canonical form.

4 Octahedra Decision Diagrams

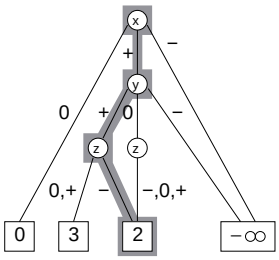
4.1 Overview

The constraints of an octahedron can be represented compactly using a specially devised decision diagram representation. This representation is called *Octahedron Decision Diagram* (OhDD). Intuitively, it can be described as a Multi-Terminal Zero-Suppressed Ternary Decision Diagram:

– *Ternary*: Each non-terminal node represents a variable x_i and has three output arcs, labelled as $\{-1, 0, +1\}$. Each arc represents a coefficient of x_i in a linear constraint.
– *Multi-Terminal* [10]: Terminal nodes can be constants in $\mathbb{R} \cup \{-\infty\}$. The semantics of a path σ from the root to a terminal node k is the linear constraint $(c_1 \cdot x_1 + c_2 \cdot x_2 + \ldots + c_n \cdot x_n \geq k)$, where c_i is the coefficient of the arc taken from the variable x_i in the path σ.
– *Zero-Suppressed* [14]: If a variable does not appear in any linear constraint, it also does not appear in the OhDD. This is achieved by using special reduction rules as it is done in Zero-Suppressed Decision Diagrams.

Figure 4 shows an example of a OhDD and the octahedron it represents on the right. The shadowed path highlights one constraint of the octahedron, $(x + y - z \geq 2)$. All constraints that end in a terminal node with $-\infty$ represent constraints with an unknown bound, such as $(x - y \geq -\infty)$. As the OhDD represents the saturated form of the octahedron, some redundant constraints such as $(x + y + z \geq 3)$ appear explicitly.

This representation based on decision diagrams provides three main advantages. First, decision diagrams provide many opportunities for reuse. For example, nodes in a



$$
\begin{aligned}
x &\geq 2 \\
y &\geq 0 \\
z &\geq 0 \\
x + y &\geq 3 \\
x - z &\geq 2 \\
x + y - z &\geq 2 \\
x + y + z &\geq 3
\end{aligned}
$$

Fig. 4. An example of a OhDD. On the right, the constraints of the octahedron.

OhDD can be shared. Furthermore, different OhDD can share internal nodes, leading to a greater reduction in the memory usage. Second, the reduction rules avoid representing the zero coefficients of the linear inequalities. Finally, symbolic algorithms on OhDD can deal with sets of inequalities instead of one inequality at a time. All these factors combined improve the efficiency of operations with octahedra.

4.2 Definitions

Definition 5 (Octahedron Decision Diagram - OhDD). *An Octahedron Decision Diagram is a tuple (V, G) where V is a finite set of positive real-valued variables, and $G = (N \cup K, E)$ is a labeled single rooted directed acyclic graph with the following properties. Each node in K, the set of* terminal *nodes, is labeled with a constant in $\mathbb{R} \cup \{-\infty\}$, and has an outdegree of zero. Each node $n \in N$ is labeled with a variable $v(n) \in V$, and it has three outgoing arcs, labeled $-$, 0 and $+$.*

By establishing an order among the variables of the OhDD, the notion of *ordered* OhDD can be defined. The intuitive meaning of ordered is the same as in BDDs, that is, in every path from the root to the terminal nodes, the variables of the decision diagram always appear in the same order. For example, the OhDD in Fig. 4 is an ordered OhDD.

Definition 6 (Ordered OhDD). *Let $\succ$ be a total order on the variables V of a OhDD. The OhDD is ordered if, for any node $n \in N$, all of its descendants $d \in N$ satisfy $v(d) \succ v(n)$.*

In the same way, the notion of a *reduced* OhDD can be introduced. However, the reduction rules will be different in order to take advantage of the structure of the constraints. In an octahedron, most variables will not appear in all the constraints. Avoiding the representation of these variables with a zero coefficient would improve the efficiency of OhDD. This can be achieved as in ZDDs by using a special reduction rule: whenever the target of the $-$ arc of a node n is $-\infty$, and the 0 and $+$ arcs have the same target m, n is reduced as m. The rationale behind this rule is the following: if a constraint $(c_1 \cdot x_1 + \ldots + c_i \cdot x_i + \ldots + c_n \cdot x_n \geq k)$ holds for $c_i = 0$, it will also hold for $c_i = +1$ as $x_i \geq 0$. However, it is not known if it will hold for $c_i = -1$. This means that in the OhDD, if a variable has coefficient zero in a constraint, it is very likely that it will end up creating a node where the 0 and $+$ arcs have the same target, and the target of

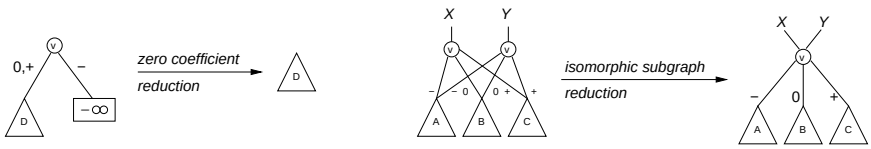


Fig. 5. Reduction rules for OhDD.

the $-$ arc is $-\infty$. By reducing these nodes, the zero coefficient is not represented in the OhDD. Remarkably, using this reduction rule, the set of constraints stating that "any sum of variables is greater or equal to zero" is represented only as the terminal node 0.

Figure 5 shows an example of the two reduction rules. Notice that contrary to BDDs, nodes where all arcs have the same target will not be reduced.

Definition 7 (Reduced OhDD). *A reduced OhDD is an ordered OhDD where none of the following rules can be applied:*

- Reduction of zero coefficients*: Let $n \in N$ be a node with the $-$ arc going to the terminal $-\infty$, and with the arcs 0 and $+$ point to a node m. Replace n by m.*
- Reduction of isomorphic subgraphs*: Let D_1 and D_2 be two isomorphic subgraphs of the OhDD. Merge D_1 and D_2.*

4.3 Implementation of the Operations

The octahedra abstract domain and its operations have been implemented as OhDD on top of the CUDD decision diagram package [23]. Each operation on octahedra performs simple manipulations such as computing the maximum or the minimum between two systems of inequalities, where each inequality is encoded as a path in a OhDD. These operations can be implemented as recursive procedures on the decision diagram. The algorithm may take as arguments one or more decision diagrams, depending of the operation. All these recursive algorithms share the same overall structure:

1. Check if the call is a base case, e.g. all arguments are constant decision diagrams. In that case, the result can be computed directly.
2. Look up the cache to see if the result of this call was computed previously and is available. In that case, return the precomputed result.
3. Select the top variable t in all the arguments according to the ordering. The algorithm will only consider this variable during this call, leaving the rest of the variables to be handled by the subsequent recursive calls.
4. Obtain the cofactors of t in each of the arguments of the call. In our case, each cofactor represents the set of inequalities for each coefficient of the top variable.
5. Perform recursive calls on the cofactors of t.
6. Combine the results of the different calls into the new top node for variable t.
7. Store the result of this recursive call in the cache.
8. Return the result to the caller.

The saturation algorithm is a special case: all sums of pairs of constraints are computed by a single traversal; but if new inequalities have been discovered, the traversal must be

repeated. The process continues until a fixpoint is reached. Even though this fixpoint might not be reached, as seen in Fig. 3, the number of iterations required to saturate an octahedron tends to be very low (1-4 iterations) if it is derived from saturated octahedra, e.g. the intersection of two saturated octahedra.

These traversals might have to visit 3^n inequalities/paths in the OhDD in the worst case. However, as OhDD are directed graphs, many paths share nodes so many recursive calls will have been computed previously, and the results will be reused without the need to recompute. The efficiency of the operations on decision diagrams depends upon on two very important factors. The first one is the *order of the variables* in the decision diagram. Intuitively, each call should perform as much work as possible. Therefore, the variables that appear early in the decision diagram should discriminate the result as much as possible. Currently there is no *dynamic reordering* [21] in our implementation of OhDD, but we plan to add it in the near future. A second factor in the performance of these algorithms is the *effectivity of the cache* to reuse previously computed results.

5 Applications of the Octahedron Abstract Domain

5.1 Motivating Application

Asynchronous circuits are a kind of circuits where there is no global clock to synchronize its different components. Asynchronous circuits replace the global clock by a local hand-shake between components, gaining several advantages such as lower power usage. However, the absence of a clock makes the verification of asynchronous circuits more complex. The lack of clock makes the circuit more dependent on *timing constraints* that ensure the correctness of the synchronization within the circuit. This means that the correctness of the circuit depends on the delays of its gates and wires.

In many asynchronous circuits implementing control logic, the timing constraints that arise are unit inequalities. Intuitively, they correspond to constraints of the type

$$\underbrace{(\delta_1 + \cdots + \delta_i)}_{\text{delay(path}_1\text{)}} - \underbrace{(\delta_{i+1} + \cdots + \delta_n)}_{\text{delay(path}_2\text{)}} \geq k$$

hinting that certain paths in the circuit must be longer than other paths. In very rare occasions, coefficients different from ± 1 are necessary. A typical counterexample would be a circuit where one path must be c times longer than another one, e.g. a fast counter.

Example. Figure 6(a) depicts a D flip-flop [20]. Briefly stated, a D flip-flop is a 1-bit register. It stores the data value in signal D whenever there is a rising edge in the clock signal CK. The output Q of the circuit is the value which was stored in the last clock rising edge. We would like to characterize the behavior of this circuit in terms of the internal gate delays. The flip-flop has to be characterized with respect to three parameters (see Figure 6(b)):

- *Setup time*, noted as T_{setup}, is the amount of time that D should remain stable before a clock rising edge.
- *Hold time*, noted as T_{hold}, is the amount of time that D should remain stable after a clock rising edge.
- *Delay* or *clock-to-output time*, noted as $T_{CK \to Q}$, is the amount of time required by the latch to propagate a change in the input D to the output Q.

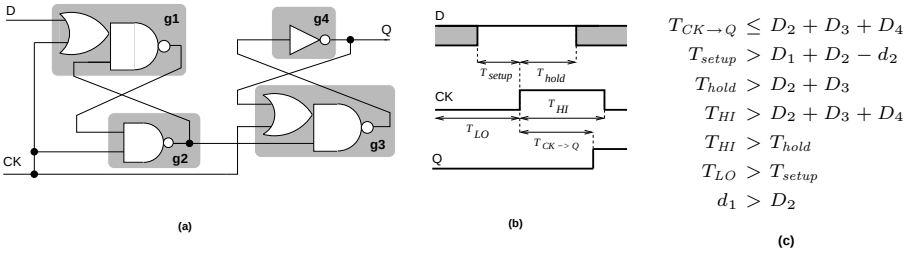


Fig. 6. (a) Implementation of a D flip-flop [20], (b) description of variables that characterize any D flip-flop and (c) sufficient constraints for correctness for any delay of the gates.

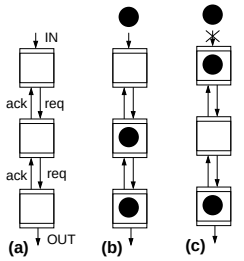
Example	States	Variables	Time - Poly (sec)	Time - Oct (sec)
nowick	60	30	0.5	0.1
sbuf-read-ctl	74	31	1.2	1.4
rcv-setup	72	27	2.1	8.3
alloc-outbound	82	39	1.3	0.2
ebergen	83	27	1.3	1.7
mp-forward-pkt	194	29	1.9	3.8
chu133	288	26	1.3	1.0

Table 2. Experimental results using convex polyhedra and octahedra.

The timing analysis algorithm is capable of deriving a set of sufficient linear contraints that guarantee the correctness of the circuit's behavior. This behavior will be correct if the output Q matches the value of D in the last clock rising edge. Any behavior not fulfilling this property is considered to be a failure. Fig. 6(c) reports the set of sufficient timing constraints derived by the algorithm. Each gate g_i has a symbolic delay in the interval $[d_i, D_i]$. Notice that the timing constraints are unit inequalities.

Experimental Results. Timing verification has been performed on several asynchronous circuits from the literature. This verification can be seen as the analysis of a set of clock variables, and the underlying timing behavior can be modeled as assignments and guards on these variables [4]. The analysis of clock variables has been performed using two different numeric abstractions: convex polyhedra and octahedra. The implementation of polyhedra uses the New Polka polyhedra library [19], while the library of OhDD is implemented on top of the CUDD package [23]. Table 2 shows a comparison of the experimental results for some examples. All these examples were verified successfully using both octahedra and polyhedra, as all relevant constraints were unit linear inequalities. For all these cases, the execution time of convex polyhedra and octahedra is comparable, while the memory usage for octahedra is lower. For each example, we provide the number of different states (configurations) of the circuit, the number of clock and delay variables of the abstractions and the execution time required by the analysis with each abstraction.

The difference in memory usage is quantified in the next example, an asynchronous pipeline with different number of stages and an environment running at a fixed frequency. The processing time required by each stage i has a processing time bounded by an interval, with unknown upper and lower bound $[d_i, D_i]$. Whenever a stage finishes its



# of stages	# of States	# of variables	Polyhedra CPU Time	Polyhedra Memory	OhDD CPU Time	OhDD Memory
2	36	20	0.6s	64Mb	1s	5Mb
3	108	24	2s	67Mb	17s	8Mb
4	324	28	13.5s	79Mb	249s	39Mb
5	972	32	259.2s	147Mb	3974s	57Mb
6	2916	36	–	–	143058s	83Mb

Fig. 7. (a) Asynchronous pipeline with N=3 stages, (b) correct behavior of the pipeline and (c) incorrect behavior. Dots represent data elements. On the right, the CPU time and memory required to verify pipelines with different number of stages.

computation, it sends the result to the next stage if it is empty. The safety property being verified in this case was *"the environment will never have to wait before sending new data to the pipeline"*, i.e. whenever the environment sends new data to the pipeline, the first stage is empty. Fig.7 shows the pipeline, with an example of a correct and incorrect behavior. The tool discovers that correct behavior can be ensured if the following holds:

$$d_{IN} > D_1 \wedge \ldots \wedge d_{IN} > D_N \wedge d_{IN} > D_{OUT}$$

where D_i is the delay of stage i, and d_{IN} and D_{OUT} refer to environment delays. This property is equivalent to:

$$d_{IN} > max(D_1, \ldots, D_N, D_{OUT})$$

Therefore, the pipeline is correct if the environment is slower than the slowest stage of the pipeline. Both the polyhedra and octahedra abstract domain are able to discover this property. This example is interesting because it exhibits a very high degree of concurrency. The verification times and memory usage for different lengths of the pipeline can be found in Fig.7. Notice that the memory consumption of OhDD is lower than that of convex polyhedra. This reduction in memory usage is sufficient to verify larger pipelines (n = 6 stages) not verifiable with our convex polyhedra implementation. However, this memory reduction comes at the expense of an increase in the execution time.

5.2 Other Applications

In general, the octahedron abstract domain may be interesting in any analysis problem where convex polyhedra can be used. Many times, the precision obtained with convex polyhedra is very good, but the efficiency of the analysis limits the applicability. In these scenarios, using octahedra might be adequate as long as the variables involved in the analysis are positive and unit linear inequalities provide sufficient information for the specific problem. Some examples of areas of applications are the following:

- *Analysis of program invariants involving unsigned variables*.
- *Static discovery of bounds in the size of asynchronous communication channels*: Many systems communicate using a non-blocking semantics, where the sender does not wait until the receiver is ready to read the message. In these systems, each channel requires a buffer to store the pending messages. Allocating these buffers statically would improve performance but it is not possible, as the amount of pending messages during execution is not known in advance. Analysis with octahedra

could discover these bounds statically. This problem is related to the problem of structural boundedness of a Petri Net [18], where an upper bound on the number of tokens that can be in each place of the Petri Net must be found.
– *Analysis of timed systems*: Clocks and delays are restricted to positive values in many types of models. Octahedra can be used to analyze these values and discover complex properties such as timing constraints or worst-case execution time(WCET).
– *Analysis of string length in C programs* [8]: Checking the absence of buffer overflows is important in many scenarios, specially in the applications where security is critical, e.g an operating system. C programs are prone to errors related to the manipulation of strings. Several useful constraints on the length of strings can be represented with octahedra. For instance, a constraint on the concatenation of two strings can be $\texttt{strlen}(\texttt{strcat}(s_1, s_2)) = \texttt{strlen}(s_1) + \texttt{strlen}(s_2)$.

6 Conclusions and future work

A new numeric abstract domain called octahedron has been presented. This domain can represent and manipulate constraints on the sum or difference of an arbitrary number of variables. In terms of precision, this abstraction is between octagons and convex polyhedra. Regarding complexity, the worst case complexity of octahedra operations over n variables is $O(3^n)$ in memory, and $O(3^n)$ in execution time in addition to the cost of saturation. However, worst-case performance is misleading due to the use of a decision diagram approach. For instance, BDDs have a worst-case complexity of $O(2^n)$, but they have a very good behavior in many real examples. Performance in this case depends on factors such as the ordering of the variables in the decision diagram and the effectiveness of the cache. In the experimental results of OhDD, memory consumption was shown to be smaller than that of our convex polyhedra implementation. Running time was comparable to that of convex polyhedra in small and medium-sized examples, while in more complex examples the execution time was worse. This shows that OhDD trade speed for a reduction in memory usage.

Future work in this area will try to improve the execution time of octahedra operations. For example, dynamic reordering [21] would improve efficiency if proper heuristics to find good variable orders can be developed. Another area where there is room for improvement is the current bottleneck of the representation, the saturation procedure.

Acknowledgements. This work has been partially funded by CICYT TIC2001-2476 and the FPU grant AP2002-3862 from the Spanish Ministry of Education, Culture and Sports. The authors would like to thank the referees for their valuable comments.

References

1. R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
2. G. Behrmann, K. G. Larsen, J. Pearson, C. Weise, and W. Yi. Efficient timed reachability analysis using clock difference diagrams. In *Computer Aided Verification*, pages 341–353, 1999.

3. R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986.
4. R. Clarisó and J. Cortadella. Verification of timed circuits with symbolic delays. In *Proc. of Asia and South Pacific Design Automation Conference*, pages 628–633, 2004.
5. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. of the ACM Symposium on Principles of Programming Languages*, pages 238–252. ACM Press, 1977.
6. P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proc. of the ACM Symposium on Principles of Programming Languages*, pages 84–97. ACM Press, New York, 1978.
7. D. L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Automatic Verification Methods for Finite State Systems*, LNCS 407, pages 197–212. Springer-Verlag, 1989.
8. N. Dor, M. Rodeh, and M. Sagiv. CSSV: towards a realistic tool for statically detecting all buffer overflows in C. In *Proceedings of the ACM SIGPLAN 2003 conference on Programming lan guage design and implementation*, pages 155–167. ACM Press, 2003.
9. E. Asarin, M. Bozga, A. Kerbrat, O. Maler, M. Pnueli, and A. Rasse. Data structures for the verification of timed automata. In O. Maler, editor, *Hybrid and Real-Time Systems*, pages 346–360, Grenoble, France, 1997. Springer Verlag, LNCS 1201.
10. M. Fujita, P. C. McGeer, and J. C.-Y. Yang. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. *Formal Methods in System Design*, 10(2/3):149–169, 1997.
11. N. Halbwachs, Y.-E. Proy, and P. Roumanoff. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design*, 11(2):157–185, 1997.
12. T. A. Henzinger. *The Temporal Specification and Verification of Real-Time Systems*. PhD thesis, Stanford University, Aug. 1991.
13. C. Mauras. Symbolic simulation of interpreted automata. In *3rd Workshop on Synchronous Programming*, Dec. 1996.
14. S. Minato. Zero-supressed BDDs for set manipulation in combinatorial problems. In *Proc. ACM/IEEE Design Automation Conference*, pages 272–277, 1993.
15. A. Miné. A new numerical abstract domain based on difference-bound matrices. In *Programs as Data Objects II*, volume 2053 of *LNCS*, pages 155–172. Springer-Verlag, May 2001.
16. A. Miné. The octagon abstract domain. In *Analysis, Slicing and Tranformation (in Working Conference on Reverse Engineering)*, IEEE, pages 310–319. IEEE CS Press, Oct. 2001.
17. J. Møller, J. Lichtenberg, H. R. Andersen, and H. Hulgaard. Difference decision diagrams. In *Computer Science Logic*, The IT University of Copenhagen, Denmark, 1999.
18. T. Murata. State equation, controllability and maximal matchings of Petri nets. *IEEE Transactions on Automatic Control*, AC-22(3):412–416, 1977.
19. New Polka: Convex Polyhedra Library. http://www.irisa.fr/prive/bjeannet/newpolka.html.
20. C. Piguet et al. Memory element of the Master-Slave latch type, constructed by CMOS technology. US Patent 5,748,522, 1998.
21. R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, pages 42–47, 1993.
22. A. Simon, A. King, and J. M. Howe. Two Variables per Linear Inequality as an Abstract Domain. In M. Leuschel, editor, *Proceedings of Logic Based Program Development and Transformation*, LNCS 2664, pages 71–89. Springer-Verlag, 2002.
23. F. Somenzi. CUDD: Colorado university decision diagram package. Available online at `http://vlsi.colorado.edu/~fabio/CUDD`.
24. F. Wang. Symbolic parametric safety analysis of linear hybrid systems with BDD-like data-structures. In *Proceedings of Computer Aided Verification*. Springer-Verlag, July 2004.