Higher-Order Symb Comput (2006) 19:31–100
DOI 10.1007/s10990-006-8609-1

# The octagon abstract domain*

**Antoine Miné**†



**Abstract** This article presents the *octagon abstract domain*, a relational numerical abstract domain for static analysis by abstract interpretation. It allows representing conjunctions of constraints of the form $\pm X \pm Y \leq c$ where $X$ and $Y$ range among program variables and $c$ is a constant in $\mathbb{Z}$, $\mathbb{Q}$, or $\mathbb{R}$ automatically inferred. Abstract elements are represented using modified Difference Bound Matrices and we use a normalization algorithm loosely based on the shortest-path closure to compute canonical representations and construct best-precision abstract transfer functions. We achieve a quadratic memory cost per abstract element and a cubic worst-case time cost per abstract operation, with respect to the number of program variables.

In terms of cost and precision, our domain is in between the well-known fast but imprecise interval domain and the costly polyhedron domain. We show that it is precise enough to treat interesting examples requiring *relational* invariants, and hence, out of the reach of the interval domain. We also present a *packing* strategy that allows scaling our domain up to large programs by tuning the amount of relationality. The octagon domain was incorporated into the ASTRÉE industrial-strength static analyzer and was key in proving the absence of run-time errors in large critical embedded flight control software for Airbus planes.

**Keywords** Static analysis · Abstract interpretation · Numerical abstract domains · Relational numerical invariants

*This paper is the journal version of an earlier conference paper [44] sharing this title. However, the present version, extracted from the author's PhD [46] is extended in many ways and enriched with new experimental results.

†Partially supported by the exploratory project ASTRÉE of the *Réseau National de recherche et d'innovation en Technologies Logicielles* (RNTL).

A. Miné (✉)
Département d'Informatique, École Normale Supérieure, 45 rue d'Ulm, F-75230, Paris Cedex 05, France
e-mail: mine@di.ens.fr

## 1. Introduction

Writing correct programs has always been considered a great challenge and, generally, much more time and effort is needed to hunt down and eliminate *bugs* than to actually write programs. As we rely more and more on software, the consequences of a bug are more dramatic, causing great financial and even human losses. An extreme example is the overflow bug that caused the failure of the Ariane 5 launcher in 1996 [37]. Testing, one of the most widely used techniques to ensure the correctness of programs, is not sufficient. As only a few sample program behaviors can be observed, it misses bugs. Hence the need for formal methods to provide mathematically sound techniques that guarantee the full coverage of all program behaviors while relying on symbolic—as opposed to explicit—representations to achieve efficiency.

In this paper, we will work in the *Abstract Interpretation* framework [18, 20] which is a general theory of the approximation of program semantics. It allows, among other applications, designing static analyzers that are able to automatically discover, at compile-time, properties of the run-time behavior of programs. These analyzers are sound by construction: spurious behaviors can be reported but no behavior—and thus, no bug—is missed. A core concept is that of an *abstract domain*. An abstract domain is a class of computer-representable program properties together with a set of effective operators to manipulate them. Such operators include abstract counterparts for semantical transfer functions that model assignment and test statements, and set-theoretic operators such as union and intersection. They also include extrapolation operators such as widenings to compute, in finite time, over-approximations of least-fixpoints involved in the semantics of loops and recursive functions. Each abstract domain embeds some sort of approximation and there does not exist a single, all-purpose, abstract domain. It must be chosen depending on the properties that need to be inferred, but also the programming style of the analyzed programs and the amount of computing resources available for the static analysis. Once an abstract domain is designed, it can be plugged into a static analyzer based on Abstract Interpretation to perform the analysis fully automatically and directly on the source code.

In this paper, we are interested in *numerical* abstract domains. They focus on numerical properties of program variables and allow answering questions such as: "Can there be a division by zero?," "Can this computation overflow the precision of machine-integers?," "Can this array index exceed the array bounds?." Moreover, many non-numerical analyses are built on top of non-standard instrumented semantics that introduce numerical quantities, and hence, are parametrized by numerical abstract domains. Well-known examples include pointer aliasing analyses by Deutsch [23] and Venet [53], a shape analysis by Rugina [50], a string cleanness analysis by Dor et al. [25], analyses of $\pi$-calculus by Feret [26], parametric predicate abstractions by Cousot [16], and even liveness analyses such as the termination analysis by Colón and Sipma [13]. There already exist several numerical abstract domains. Well-know examples include the interval domain by Cousot and Cousot [17] that discovers variable bounds ($\bigwedge_i\ X_i \in [a_i, b_i]$), Karr's domain [35] that discovers affine equalities between variables ($\bigwedge_j \sum_i \alpha_{ij} X_i = \beta_j$), Cousot and Halbwachs' polyhedron domain [22] for affine inequalities ($\bigwedge_j \sum_i \alpha_{ij} X_i \leq \beta_j$), Granger's congruence domain [29] ($\bigwedge_i\ X_i \in a_i\mathbb{Z} + b_i$), etc. In this paper, we introduce a new numerical abstract domain, called the *octagon abstract domain*, which is able to represent and manipulate conjunctions of invariants of the form $\pm X \pm Y \leq c$, where $X$ and $Y$ are program variables, and $c$ is a constant. It can be seen as a restriction of the polyhedron domain where each inequality constraint only involves at most two variables and unit coefficients. Figure 1 presents a set of points together with its best abstraction within the interval, the octagon, and the polyhedron domain.

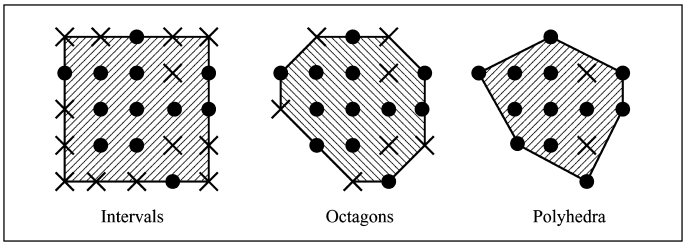


**Fig. 1** The same set of points • abstracted in the interval, the octagon, and the polyhedron domain. Spurious points caused by the approximation are denoted by the × symbol

The more precise the domain is, the fewer spurious points there are. We see that the octagon domain has a precision between that of the interval domain and the polyhedron domain.

### 1.1. The need for relational domains

A *relational* domain is a domain that is able to discover relationships between variables. For instance, the polyhedron domain is relational while the interval domain is non-relational. In order to motivate the necessity for low-cost relational numerical abstract domains, consider the following code fragment where the variable `Y` is incremented at most eleven times within a loop with loop counter `X`:

```
X := 10
Y := 0
while X ≥ 0 {
   X := X−1
   if random() { Y := Y+1 }
}
```

In order to prove the non-relational invariant $\texttt{Y} \leq 11$ at the end of the loop, it is necessary to first prove the relational loop invariant $\texttt{X}+\texttt{Y} \leq 10$, and then combine it with the loop exit condition $\texttt{X} = -1$. Thus, this invariant is out of the reach of the interval domain but can be established using the polyhedron domain. We will see that the octagon domain, while less precise than the polyhedron domain, is precise enough to treat this example. Other interesting properties that can be exactly represented using the octagon domain include mutual exclusion, $\neg(X \wedge Y)$, encoded as $X \geq 0 \wedge Y \geq 0 \wedge X + Y \leq 1$, as well as numerical properties on absolute values, such as $|X| \leq Y + 1$, encoded as $X - Y \leq 1 \wedge -X - Y \leq 1$. Finally, some analyses, such as the pointer and the $\pi$-calculus analyses proposed respectively by Venet [53] and Feret [26], require the use of relational underlying numerical abstract domains to discover non-uniform invariants—i.e., invariants able to distinguish objects at different positions within the same array or recursive data-structure, and different channel instances created at the same syntactic point.

The main advantage of the octagon domain over the polyhedron domain is its smaller worst-case cost. The octagon domain has a quadratic memory cost (per abstract element) and a cubic worst-case time cost (per abstract operation), with respect to the number of variables. The polyhedron domain has a memory and time cost that is unbounded in theory and exponential in practice. Although the polyhedron domain might be less costly on some

analyses, experience shows that its cost can grow in unpredictable ways while the octagon domain exhibits a very predictable cubic cost in practice.

### 1.2. Previous work

There has been much work on satisfiability algorithms for conjunctions of inequalities of a restricted form. Consider, first, so-called *potential constraints*, that is, constraints of the form $X - Y \leq c$. A core result by Bellman [7] is that the satisfiability of conjunctions of potential constraints in $\mathbb{Z}$, $\mathbb{Q}$, or $\mathbb{R}$ can be reduced to checking for the existence of a cycle with a strictly negative total weight in a weighted directed graph. This result was then extended by Harvey and Stuckey [32] and Jaffar et al. [33], to integer constraints of the form $\pm X \pm Y \leq c$. However, these works focus on satisfiability only and do not study the more complex problem of *manipulating* constraint conjunctions.

From Bellman's result, people from the model checking community of timed automata [24, 54] and timed Petri nets [40] derived a structure called Difference Bound Matrix (or DBM) allowing the manipulation of conjunctions of potential constraints. They developed algorithms to compute a canonical representation of DBMs, using the notion of shortest-path closure. They also developed algorithms to compute the intersection of DBMs and test inclusion and equality. Other algorithms presented in [54] and [40] are not useful for analysing general purpose programming languages by Abstract Interpretation while many—such as a union abstraction, general assignment and test transfer functions, widenings, etc.—are missing. The idea of using DBMs to design a full abstract domain that can infer potential constraints is already present in the PhD work of Bagnara [4, Chap. 5] and Jeannet [34, Section 2.4.3]. It has been effectively carried out simultaneously in the work of Shaham, Kolodner, and Sagiv, in [51], and in our previous paper [43]. Some constructions in the present paper are reminiscent of this abstract domain, but extended to the richer set of constraints $\pm X \pm Y \leq c$.

A first set of algorithms for the manipulation of constraints of the form $\pm X \pm Y \leq c$ was proposed by Balasundaram and Kennedy [6] to represent data access patterns in arrays and perform automatic loop parallelization—such constraint sets were denoted there as "simple sections". Alas, the authors fail to propose a normal form, although they acknowledge that it is required in the implementation of their union abstraction. Moreover, they present a single transfer function that abstracts nested loops of a simple form, which is too specific for our purpose. In the present paper, we choose to start from our abstract domain for potential constraints [43] and adapt the DBM representation and its algorithms. In particular, much work is required to adapt the normal form. We already presented, in a conference paper [44], an early construction of the octagon abstract domain. Since then, some more work has been done. In particular, we propose in the present paper new and enhanced transfer functions—such as backward assignments and transfer functions for interval linear forms—as well as encouraging experimental results related to the ASTRÉE project [3].

### 1.3. Overview of the paper

The paper is organized as follows. In Section 2, we show how to represent conjunctions of constraints of the form $\pm X \pm Y \leq c$, so-called octagons, using modified Difference Bound Matrices. Then, in Section 3, we present our normalization algorithm and its properties. In particular, we are able to prove a *saturation* property that will guarantee exactness and best-precision results for some of our abstract transfer functions. We will present both a cubic-time algorithm for rational and real constraints, and a quartic-time algorithm for the,

more complex, integer case. Section 4 is devoted to the design of all the abstract operators and transfer functions required by an abstract domain. Whenever possible, we will propose several abstractions of the same concrete function with different cost versus precision trade-offs. A few example analyses demonstrating the precision of the octagon domain are presented in Section 5. Finally, Section 6 presents the integration of the octagon domain within the ASTRÉE industrial-strength static analyzer aimed at proving the absence of run-time errors in large embedded reactive avionics software. We also present, in Section 6, a *packing* technique allowing us to improve the efficiency of the analysis by relating only selected variables together and achieve a practical cost that is *linear* in the program size. Experimental results show that, while being precise enough to eliminate hundreds of sources of imprecision, the octagon domain scales up to real-life programs of a few hundred thousand lines. Section 7 concludes. All our proofs are postponed to the appendix, together with a summary of all introduced symbols and notations.

## 2. Octagon representation

We suppose that we are given a program with a *finite* set of variables $\mathcal{V} \stackrel{\text{def}}{=} \{V_1, \ldots, V_n\}$. All variables live in a numerical set $\mathbb{I}$ that can be $\mathbb{Z}$, $\mathbb{Q}$, or $\mathbb{R}$. An environment $\rho \in (\mathcal{V} \to \mathbb{I})$ maps each variable to its value, at a given program point. An environment will often be assimilated to a point in $\mathbb{I}^n$.

We call *octagonal constraint* any constraint of the form $\pm V_i \pm V_j \leq c$ with $c \in \mathbb{I}$. We call *octagon* the set of points satisfying a conjunction of octagonal constraints. The name "octagon" comes from the fact that, in two dimensions $\mathcal{V} = \{V_1, V_2\}$, our sets are polyhedra with at most eight sides.

### 2.1. Potential constraints

First, we recall how to encode the subset of octagonal constraints, so-called *potential constraints*, that have the form $V_i - V_j \leq c$. The term *potential* comes from the fact that solutions of conjunctions of potential constraints are defined up to a constant. If $(v_1, \ldots, v_n)$ is such a solution, so is $(v_1 + x, \ldots, v_n + x)$ for every $x \in \mathbb{I}$. The set of points in $\mathbb{I}^n$ that satisfy a conjunction of potential constraints will be called a *potential set*.

*Potential graphs.* A conjunction of potential constraints can be represented as a directed weighted graph $\mathcal{G}$ with nodes $\mathcal{V}$ and weights with value in $\mathbb{I}$. Such a graph is called a *potential graph*. For each ordered pair of variables $(V_i, V_j) \in \mathcal{V}^2$, there will be an arc from $V_i$ to $V_j$ with weight $c$ if the constraint $V_j - V_i \leq c$ is in the constraint conjunction. We can assume, without loss of generality, that there is at most one arc from any given node to any other given node. If several upper bounds for the same variable difference appear in the conjunction, all but the smallest upper bound are obviously redundant. We use the following graph terminology: a *path* in $\mathcal{G}$ is a sequence of nodes, denoted by $\langle V_{i_1}, \ldots, V_{i_m} \rangle$, such that there is an arc from each $V_{i_k}$ to $V_{i_{k+1}}$; a path is said to be *simple* if its *internal nodes* $V_{i_2}, \ldots, V_{i_{m-1}}$ are pairwise distinct and different from $V_{i_1}$ and $V_{i_m}$; a *cycle* is a path $\langle V_{i_1}, \ldots, V_{i_m} \rangle$ such that $V_{i_m} = V_{i_1}$; a *simple cycle* is a cycle that is also a simple path.

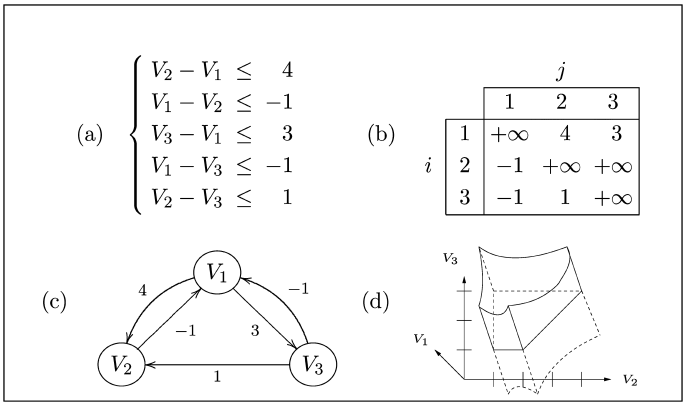


**Fig. 2** A potential constraint conjunction (a), its corresponding DBM **m** (b), potential graph $\mathcal{G}(\mathbf{m})$ (c), and potential set concretization $\gamma^{Pot}(\mathbf{m})$ (d)

*Difference bound matrices.* Let $\overline{\mathbb{I}} \stackrel{\text{def}}{=} \mathbb{I} \cup \{+\infty\}$ be the extension of $\mathbb{I}$ to $+\infty$. The order $\leq$ is extended by stating that $\forall c \in \mathbb{I},\ c \leq +\infty$—the extension of other operators to $\overline{\mathbb{I}}$ will be presented when needed.

An equivalent representation for potential constraint conjunctions is by means of a *Difference Bound Matrix*, or DBM for short. A DBM **m** is a $n \times n$ square matrix, where $n$ is the number of program variables, with elements in $\overline{\mathbb{I}}$. The element at line $i$, column $j$, where $1 \leq i \leq n$, $1 \leq j \leq n$, denoted by $\mathbf{m}_{ij}$, equals $c \in \mathbb{I}$ if there is a constraint of the form $V_j - V_i \leq c$ in our constraint conjunction, and $+\infty$ otherwise. DBMs were introduced by Dill [24] as a convenient constraint representation for the verification of timed systems and are now used pervasively in the model-checking of timed-automata and timed Petri nets. Given a fixed number $n$ of variables, we will denote by $\mathsf{DBM}$ the set $\overline{\mathbb{I}}^{n \times n}$ of all DBMs. The potential set described by a DBM **m** is given by the following *concretization* function $\gamma^{Pot} : \mathsf{DBM} \to \mathcal{P}(\mathcal{V} \to \mathbb{I})$:

$$\gamma^{Pot}(\mathbf{m}) \stackrel{\text{def}}{=} \{(v_1, \ldots, v_n) \in \mathbb{I}^n \mid \forall i, j,\ v_j - v_i \leq \mathbf{m}_{ij}\}.$$

Each DBM **m** can be seen as the adjacency matrix of a potential graph, that will be denoted in the following by $\mathcal{G}(\mathbf{m})$. Indeed, DBMs and potential graphs are just two different notations for the same objects. Figure 2 represents a conjunction of potential constraints together with its encoding as a potential graph and as a DBM, as well as its concretization. These notations are complementary. Some theorems and algorithms will be best described using the matrix notation, while others will use graph-related terms, such as paths and cycles. In the following, we will often present examples using the graph notation even when the corresponding algorithms are presented using the matrix notation, as constraint graphs are much easier to read.

In order to allow representing interval constraints $V_i \leq c$ and $V_j \geq d$ in DBMs, a common trick—used in both model-checking [40, 54] and abstract interpretation [43]—is to add a phantom variable $V_0$ whose value is the constant zero. Thus, we encode $V_i \leq c$ and $V_j \geq d$ respectively as $V_i - V_0 \leq c$ and $V_0 - V_j \leq -d$.

## 2.2. Octagonal constraints

In order to encode conjunctions of octagonal constraints, we introduce the following technique. Given the set of variables $\mathcal{V} = \{V_1, \ldots, V_n\}$, we derive the set $\mathcal{V}' \stackrel{\text{def}}{=} \{V'_1, \ldots, V'_{2n}\}$ containing twice as many variables. Each variable $V_i \in \mathcal{V}$ has both a *positive form* $V'_{2i-1}$, and a *negative form* $V'_{2i}$ in $\mathcal{V}'$. We will encode octagonal constraints on $\mathcal{V}$ as potential constraints on $\mathcal{V}'$. Intuitively, in a potential constraint, $V'_{2i-1}$ will represent $V_i$ while $V'_{2i}$ will represent $-V_i$. More formally:

| the constraint | is represented as |
|---|---|
| $V_i - V_j \leq c \quad (i \neq j)$ | $V'_{2i-1} - V'_{2j-1} \leq c$ and $V'_{2j} - V'_{2i} \leq c$ |
| $V_i + V_j \leq c \quad (i \neq j)$ | $V'_{2i-1} - V'_{2j} \leq c$ and $V'_{2j-1} - V'_{2i} \leq c$ |
| $-V_i - V_j \leq c \quad (i \neq j)$ | $V'_{2i} - V'_{2j-1} \leq c$ and $V'_{2j} - V'_{2i-1} \leq c$ |
| $V_i \leq c$ | $V'_{2i-1} - V'_{2i} \leq 2c$ |
| $V_i \geq c$ | $V'_{2i} - V'_{2i-1} \leq -2c$ |

Thus, a conjunction of octagonal constraints on $\mathcal{V}$ can be represented as a DBM of dimension $2n$, that is, a $2n \times 2n$ matrix with elements in $\overline{\mathbb{I}} = \mathbb{I} \cup \{+\infty\}$ or, equivalently, a potential graph with nodes in $\mathcal{V}'$ and weights in $\mathbb{I}$. In contrast to DBMs representing potential constraints, interval constraints can be directly encoded without the need of an extra variable representing the constant zero. Our encoding is exemplified in Fig. 3. This encoding may seem complex at first, but it serves an important purpose. Most of our operations will be constructed by considering, as a first approximation, that $V'_1$ to $V'_{2n}$ are *distinct variables*. Then, some correcting term is applied to take into account the fact that variables in $\mathcal{V}'$ are related by the constraints $\forall i,\ V'_{2i-1} = -V'_{2i}$. This way, we benefit from many existing properties and operators on potential constraints and DBMs.

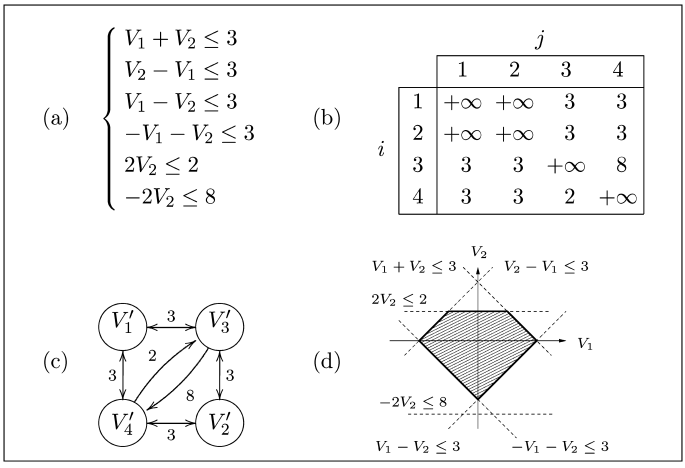


**Fig. 3** A conjunction of octagonal constraints (a), its encoding as a coherent DBM (b), and potential graph on $\mathcal{V}'$ (c), and the octagon it defines (d)

*Revised concretization.* Given a DBM $\mathbf{m}$ of dimension $2n$, we can define formally the octagon described by $\mathbf{m}$ using the following concretization $\gamma^{Oct} : \mathsf{DBM} \to \mathcal{P}(\mathcal{V} \to \mathbb{I})$:

$$\gamma^{Oct}(\mathbf{m}) \stackrel{\text{def}}{=} \{(v_1, \ldots, v_n) \in \mathbb{I}^n \mid (v_1, -v_1, \ldots, v_n, -v_n) \in \gamma^{Pot}(\mathbf{m})\}.$$

$\gamma^{Oct}$ refines the semantics of potential constraints, expressed using $\gamma^{Pot}$, with constraints inherent to our encoding, that is, $\forall i \geq 1,\ V'_{2i-1} = -V'_{2i}$. If we denote by $\Pi$ the plane $\{(v'_1, \ldots, v'_{2n}) \in \mathbb{I}^{2n} \mid \forall i,\ v'_{2i-1} = -v'_{2i}\}$, then there is a bijection between $\gamma^{Pot}(\mathbf{m}) \cap \Pi$ and $\gamma^{Oct}(\mathbf{m})$.

*Coherence.* Some octagonal constraints have two different encodings as potential constraints in $\mathcal{V}'$, and hence, are defined by two elements in the DBM. For instance, $V_i + V_j \leq c$ can be described by both potential constraints $V'_{2i-1} - V'_{2j} \leq c$ (that is, $\mathbf{m}_{(2j)(2i-1)} = c$) and $V'_{2j-1} - V'_{2i} \leq c$ (that is, $\mathbf{m}_{(2i)(2j-1)} = c$). We will say that a DBM is *coherent* if each constraint in such a related pair is equivalent to the other one. More formally:

$$\mathbf{m} \text{ is coherent} \iff \forall i, j,\ \mathbf{m}_{ij} = \mathbf{m}_{\bar{j}\bar{i}}$$

where the $\bar{\cdot}$ operator on indices is defined as:

$$\bar{\imath} \stackrel{\text{def}}{=} \begin{cases} i+1 & \text{if } i \text{ is odd} \\ i-1 & \text{if } i \text{ is even} \end{cases}$$

Intuitively, the $\bar{\cdot}$ operator corresponds to switching between the positive and the negative forms of a variable. Obviously, $\bar{\bar{\imath}} = i$. Also, the $\bar{\cdot}$ operator can be easily implemented using the `xor` bit-wise exclusive or operator as $\bar{\imath} - 1 = (i-1)$ `xor` $1$. The set of coherent DBMs will be denoted by $\mathsf{CDBM}$. From now on, we will only consider coherent DBMs when representing octagons.

## 2.3. Lattice structure and Galois connection

Let us consider the total order $\leq$ on $\mathbb{I}$, extended to $\bar{\mathbb{I}}$ by $\forall x,\ x \leq +\infty$. Its point-wise extension to matrices gives a partial order denoted by $\sqsubseteq^{\mathsf{DBM}}$ on the set $\mathsf{DBM}$ of Difference Bound Matrices. Intuitively, $\mathbf{m} \sqsubseteq^{\mathsf{DBM}} \mathbf{n}$ means that each constraint in $\mathbf{m}$ is *tighter* than the corresponding constraint in $\mathbf{n}$. The order $\sqsubseteq^{\mathsf{DBM}}$ corresponds to the subset inclusion of octagons in the sense that $\mathbf{m} \sqsubseteq^{\mathsf{DBM}} \mathbf{n} \Longrightarrow \gamma^{Oct}(\mathbf{m}) \subseteq \gamma^{Oct}(\mathbf{n})$. The converse is, however, not true. We can have $\gamma^{Oct}(\mathbf{m}) \subseteq \gamma^{Oct}(\mathbf{n})$ while $\mathbf{m}$ and $\mathbf{n}$ are incomparable with respect to $\sqsubseteq^{\mathsf{DBM}}$. Moreover, $\gamma^{Oct}$ is not one-to-one: we can have several DBM representations for a single octagon. Section 3 will be devoted entirely to studying this problem.

The set $\mathsf{DBM}$ has a greatest element $\top^{\mathsf{DBM}}$ for $\sqsubseteq^{\mathsf{DBM}}$, defined as $\forall i, j,\ \top^{\mathsf{DBM}}_{ij} \stackrel{\text{def}}{=} +\infty$. It is the only DBM representing the whole space: $\gamma^{Oct}(\top^{\mathsf{DBM}}) = \mathbb{I}^n$.

Many DBMs correspond to unsatisfiable constraint sets, and hence, represent the empty set $\emptyset$ via $\gamma^{Oct}$. However, $\mathsf{DBM}$ has no smallest element for $\sqsubseteq^{\mathsf{DBM}}$. We now enrich $\mathsf{DBM}$ with a new smallest element, denoted by $\bot^{\mathsf{DBM}}$, to obtain a lattice ($\mathsf{DBM}$, $\sqsubseteq^{\mathsf{DBM}}$, $\sqcup^{\mathsf{DBM}}$, $\sqcap^{\mathsf{DBM}}$, $\bot^{\mathsf{DBM}}$, $\top^{\mathsf{DBM}}$). This lattice is defined as follows:

$$\begin{aligned}
&\forall \mathbf{m}, \mathbf{n}, \quad \mathbf{m} \sqsubseteq^{\mathsf{DBM}} \mathbf{n} \iff \forall i, j,\ \mathbf{m}_{ij} \leq \mathbf{n}_{ij} \\
&\forall \mathbf{m}, \mathbf{n}, \quad (\mathbf{m} \sqcup^{\mathsf{DBM}} \mathbf{n})_{ij} \stackrel{\text{def}}{=} \max(\mathbf{m}_{ij}, \mathbf{n}_{ij}) \\
&\forall \mathbf{m}, \mathbf{n}, \quad (\mathbf{m} \sqcap^{\mathsf{DBM}} \mathbf{n})_{ij} \stackrel{\text{def}}{=} \min(\mathbf{m}_{ij}, \mathbf{n}_{ij})
\end{aligned}$$

$$\forall X^\sharp,\ \bot^{\mathsf{DBM}} \sqsubseteq^{\mathsf{DBM}} X^\sharp$$
$$\forall X^\sharp,\ \bot^{\mathsf{DBM}} \sqcup^{\mathsf{DBM}} X^\sharp \stackrel{\text{def}}{=} X^\sharp \sqcup^{\mathsf{DBM}} \bot^{\mathsf{DBM}} \stackrel{\text{def}}{=} X^\sharp$$
$$\forall X^\sharp,\ \bot^{\mathsf{DBM}} \sqcap^{\mathsf{DBM}} X^\sharp \stackrel{\text{def}}{=} X^\sharp \sqcap^{\mathsf{DBM}} \bot^{\mathsf{DBM}} \stackrel{\text{def}}{=} \bot^{\mathsf{DBM}}$$

where we use *bold* letters, such as $\mathbf{m}$, to refer to matrices in $\mathsf{DBM}$ and letters with a $\sharp$ exponent, such as $X^\sharp$, to refer to any element in $\mathsf{DBM}$—a matrix or $\bot^{\mathsf{DBM}}$. If we are interested in DBMs representing octagons, we can consider only the restriction of the lattice $\mathsf{DBM}$ to the subset $\mathsf{CDBM}$ of coherent matrices, extended with $\bot^{\mathsf{DBM}}$. It also forms a lattice.

By extending $\gamma^{Oct}$ so that $\gamma^{Oct}(\bot^{\mathsf{DBM}}) = \emptyset$ we obtain a monotonic concretization on $\mathsf{CDBM}$. Also, $\gamma^{Oct}$ is a complete $\sqcap^{\mathsf{DBM}}$−morphism—that is, whenever $\sqcap^{\mathsf{DBM}} B$ exists, then $\gamma^{Oct}(\sqcap^{\mathsf{DBM}} B) = \bigcap \{ \gamma^{Oct}(b) \mid b \in B \}$. When $\mathbb{I} \in \{\mathbb{Z}, \mathbb{R}\}$, the lattice is moreover complete, and hence, we can define a canonical *abstraction function* $\alpha : \mathcal{P}(\mathcal{V} \to \mathbb{I}) \to \mathsf{CDBM}$ that returns the best—i.e., smallest for $\sqsubseteq^{\mathsf{DBM}}$—DBM over-approximating a concrete set of points, following Cousot and Cousot in [19, Section 4.2.2]:

– $\alpha^{Oct}(R) \stackrel{\text{def}}{=} \bot^{\mathsf{DBM}}$ if $R = \emptyset$
– $(\alpha^{Oct}(R))_{ij} \stackrel{\text{def}}{=}$

$$\begin{cases} \max \{\rho(V_l) - \rho(V_k) \mid \rho \in R\} & \text{when } i = 2k-1,\ j = 2l-1 \\ & \text{or } \quad i = 2l,\ j = 2k \\ \max \{\rho(V_l) + \rho(V_k) \mid \rho \in R\} & \text{when } i = 2k,\ j = 2l-1 \\ \max \{-\rho(V_l) - \rho(V_k) \mid \rho \in R\} & \text{when } i = 2k-1,\ j = 2l \end{cases}$$

if $R \neq \emptyset$.

The function pair $(\alpha^{Oct}, \gamma^{Oct})$ forms a *Galois connection*, as introduced by Cousot and Cousot in [18], which is denoted as:

$$\mathcal{P}(\mathcal{V} \to \mathbb{I}) \underset{\alpha^{Oct}}{\overset{\gamma^{Oct}}{\leftrightarrows}} \mathsf{CDBM}\,.$$

When $\mathbb{I} = \mathbb{Q}$, the lattice is not complete. One can indeed construct a sequence of octagons with increasing rational bounds for a variable, such that the limit bound is no longer rational. Moreover, $\alpha^{Oct}(R)$ is not defined for every subset $R$ of $\mathbb{I}^n$—consider, for instance, $n = 1$ and $R \stackrel{\text{def}}{=} \{x \in \mathbb{Q} \mid x^2 \leq 2\}$; then, $(\alpha^{Oct}(R))_{10} = 2 \max R = 2\sqrt{2}$ which is not rational. Thus, $\alpha^{Oct}$ is a partial function, and we will say that the pair $(\alpha^{Oct}, \gamma^{Oct})$ forms a *partial Galois connection*.

## 3. Normalization algorithm

One must not confuse octagons, which are sets of points in $\mathcal{P}(\mathcal{V} \to \mathbb{I})$, and coherent DBMs, that serve to represent octagons as sets of octagonal constraints. In particular, as $\gamma^{Oct}$ is not one-to-one, one octagon can have several distinct representations in $\mathsf{CDBM}$. In this section we present a normal form for DBMs representing octagons. This normal form will be central for equality testing, but will also be used in many other abstract operations.

*Related work.* The $\gamma^{Pot}$ function is not one-to-one either and the problem of computing a normal form for DBMs representing potential sets has been well-studied in the model-checking

community [40, 54]. We build upon this work to construct our normalization for DBMs representing octagons but, as we will see, the adaptation is quite complex.

In contrast to DBMs representing potential sets, the case $\mathbb{I} = \mathbb{Z}$ is more complex than the cases $\mathbb{I} = \mathbb{Q}$ and $\mathbb{I} = \mathbb{R}$. We can only provide a normalization algorithm with a $\mathcal{O}(n^4)$ time cost for the former, while there exists a cubic algorithm for the latter. We will first focus on the rational and real cases, in Sections 3.1 to 3.4, and devote Section 3.5 entirely to the integer case.

### 3.1. Emptiness testing

We first consider the simpler case of determining whether $\gamma^{Oct}(\mathbf{m})$ is empty. A classical property of potential constraints, discovered by Bellman [7], is that their satisfiability can be tested by simply examining the simple cycles of the corresponding potential graph:

**Theorem 1.** $\gamma^{Pot}(\mathbf{m}) = \emptyset \iff \mathcal{G}(\mathbf{m})$ *has a simple cycle with a strictly negative total weight [14, Theorem 25.17].*

When $\mathbb{I} \neq \mathbb{Z}$, this theorem can be used directly to test the satisfiability of a conjunction of octagonal constraints, thanks to the following theorem:

**Theorem 2.** *When* $\mathbb{I} \in \{\mathbb{Q}, \mathbb{R}\}$, $\gamma^{Oct}(\mathbf{m}) = \emptyset \iff \gamma^{Pot}(\mathbf{m}) = \emptyset$.

Several algorithms exist to test for the existence of cycles with a strictly negative weight, such as the Bellman-Ford algorithm running in $\mathcal{O}(n \times s + n^2)$ time, where $n$ is the number of nodes and $s$ is the number of arcs in the graph—see, for instance, the classical textbook [14, Section 25.5]. We do not insist on using such techniques as we are about to provide an algorithm that will provide the emptiness information as a side-effect of solving a more complex problem.

When $\mathbb{I} = \mathbb{Z}$, Theorem 2 does not hold: we have $\gamma^{Pot}(\mathbf{m}) = \emptyset \Longrightarrow \gamma^{Oct}(\mathbf{m}) = \emptyset$ but the converse is not true. Indeed, a conjunction of integer octagonal constraints may have only non-integer solutions, as exemplified in Fig. 4, which is not possible for conjunctions of integer potential constraints. We postpone the presentation of a solution to the integer case to Section 3.5.

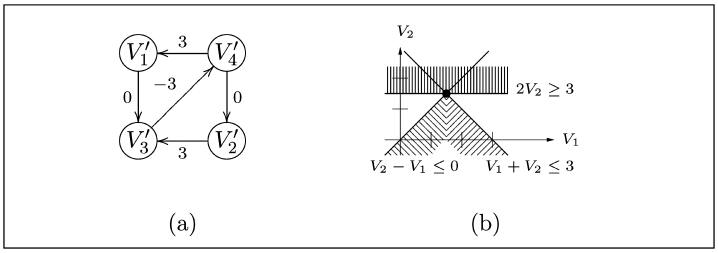


**Fig. 4** A potential graph $\mathcal{G}(\mathbf{m})$ in $\mathbb{Z}$ with no strictly negative cycle (a) and the corresponding octagon (b). $\gamma^{Oct}(\mathbf{m}) = \{(\frac{3}{2}, \frac{3}{2})\}$, which is empty in $\mathbb{Z}^2$

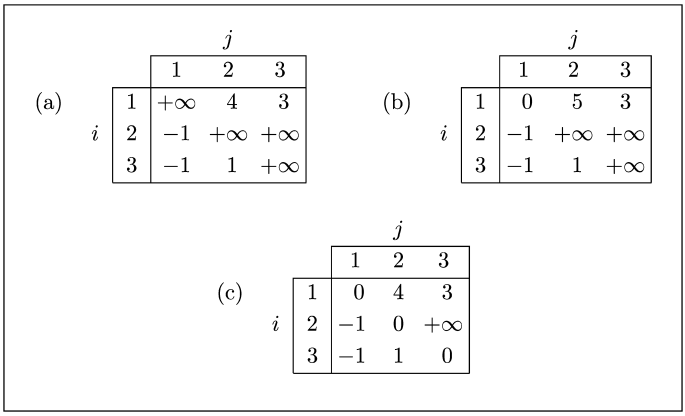


**Fig. 5** Three different DBMs with the same potential set concretization, which is also the same as in Fig. 2. Note that (a) and (b) are not even comparable with respect to $\sqsubseteq^{\mathsf{DBM}}$. Their closure is presented in (c)

### 3.2. Shortest-path closure

We now recall classical results on the normal form of Difference Bound Matrices representing potential sets. As exemplified in Fig. 5, different (possibly incomparable for $\sqsubseteq^{\mathsf{DBM}}$) DBMs can represent the same potential set. Whenever $\gamma^{Pot}(\mathbf{m})$ is not empty, $\mathcal{G}(\mathbf{m})$ has no cycle with a strictly negative weight, and hence, we can define the *shortest-path closure*—or, more concisely, *closure*—$\mathbf{m}^*$ of $\mathbf{m}$ as follows:

$$\begin{cases} \mathbf{m}^*_{ii} \stackrel{\text{def}}{=} 0 \\ \mathbf{m}^*_{ij} \stackrel{\text{def}}{=} \min\limits_{\substack{\text{all path from } i \text{ to } j \\ \langle i = i_1, i_2, \ldots, i_m = j \rangle}} \sum_{k=1}^{m-1} \mathbf{m}_{i_k i_{k+1}} & \text{if } i \neq j \end{cases}$$

where the min and $+$ operators are extended to $\overline{\mathbb{I}}$ as usual:

$$\min(x, +\infty) \stackrel{\text{def}}{=} \min(+\infty, x) \stackrel{\text{def}}{=} x$$
$$x + (+\infty) \stackrel{\text{def}}{=} (+\infty) + x \stackrel{\text{def}}{=} +\infty$$

The closure $\mathbf{m}^*$ of $\mathbf{m}$ corresponds exactly to the smallest DBM representing the potential-set $\gamma^{Pot}(\mathbf{m})$. Whenever $\gamma^{Pot}(\mathbf{m}) = \emptyset$, the closure $\mathbf{m}^*$ is not well-defined but a smallest element in $\mathsf{DBM}$ representing $\gamma^{Pot}(\mathbf{m})$ still exists; it is not a matrix but $\bot^{\mathsf{DBM}}$. By extending the $*$ operator so that $\mathbf{m}^* = \bot^{\mathsf{DBM}}$ whenever $\mathcal{G}(\mathbf{m})$ has a cycle with a strictly negative weight, we have in all cases:

$$\mathbf{m}^* = \inf_{\sqsubseteq^{\mathsf{DBM}}} \{X^\sharp \in \mathsf{DBM} \mid \gamma^{Pot}(\mathbf{m}) = \gamma^{Pot}(X^\sharp)\}.$$

*Floyd-Warshall algorithm.* One way of computing $\mathbf{m}^*$, when $\gamma^{Pot}(\mathbf{m})$ is not empty, is given by the classical Floyd-Warshall algorithm—see, for instance, [14, Section 26.2]. This algorithm has a cubic time cost with respect to the number of variables $n$. We now recall this algorithm.

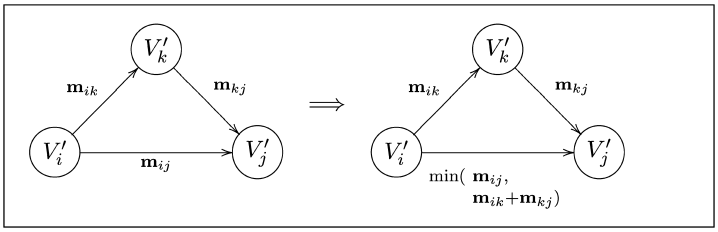


**Fig. 6** One step of propagation in the Floyd-Warshall algorithm

It is basically a loop computing $n$ matrices, $\mathbf{m}^1$ to $\mathbf{m}^n$, as follows:

$$\begin{cases} \mathbf{m}^0 \stackrel{\text{def}}{=} \mathbf{m} \\ \mathbf{m}^k_{ij} \stackrel{\text{def}}{=} \min\left(\mathbf{m}^{k-1}_{ij},\ \mathbf{m}^{k-1}_{ik} + \mathbf{m}^{k-1}_{kj}\right) \qquad \text{if } 1 \le i, j, k \le n \\ \mathbf{m}^*_{ij} \stackrel{\text{def}}{=} \begin{cases} \mathbf{m}^n_{ij} & \text{if } i \ne j \\ 0 & \text{if } i = j \end{cases} \end{cases}$$

A nice property of the Floyd-Warshall algorithm is that, whenever $\gamma^{Pot}(\mathbf{m}) = \emptyset$, the computed matrix $\mathbf{m}^n$ has at least one strictly negative diagonal coefficient, and hence, it solves both problems of checking for infeasibility and computing the closure when it exists.

*Implicit constraints.* The Floyd-Warshall algorithm has an interpretation in terms of *local constraints propagation*. For each node $V_k$ in turn, it checks, for all pairs $(V_i, V_j)$ in parallel, whether it would be shorter to pass through $V_k$ instead of taking the direct arc from $V_i$ to $V_j$. This can be depicted as a local transformation on the potential graph, as shown in Fig. 6. This also corresponds to adding the constraints:

$$V_j - V_k \le c \quad \text{and} \quad V_k - V_i \le d$$

to derive the constraint:

$$V_j - V_i \le c + d.$$

Such derived constraints, that are not explicitly encoded in the original DBM, will be called *implicit constraints*. Effectively, the closure makes *all* implicit constraints explicit.

### 3.3. Strong closure

We now extend the closure to coherent DBMs representing octagons. Let us consider a DBM $\mathbf{m}$ such that $\gamma^{Oct}(\mathbf{m}) \ne \emptyset$. It is easy to see that, if $\mathbf{m}$ is coherent, so is $\mathbf{m}^*$. Moreover, $\gamma^{Pot}(\mathbf{m}) = \gamma^{Pot}(\mathbf{n}) \Longrightarrow \gamma^{Oct}(\mathbf{m}) = \gamma^{Oct}(\mathbf{n})$, but the converse is not true: $\mathbf{m}^*$ may not be $\mathbf{m}$'s canonical representation for $\gamma^{Oct}(\mathbf{m})$. Indeed, Fig. 7 presents two *closed* DBMs representing the same octagon but different potential sets.

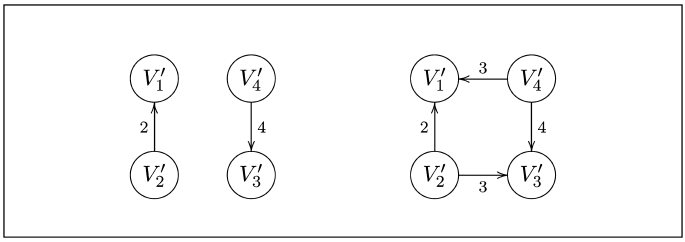


**Fig. 7** Two different *closed* potential graphs that represent the same octagon: $V_1 \leq 1 \ \wedge \ V_2 \leq 2$

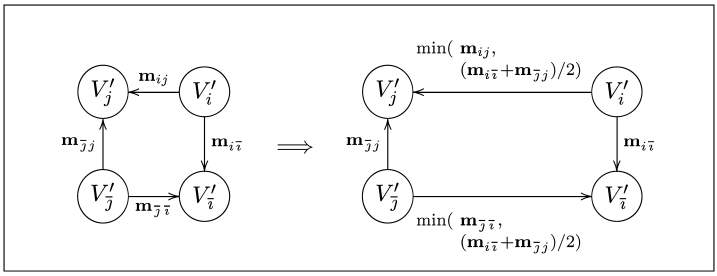


**Fig. 8** Additional propagation step performed by the strong closure

*Intuition.* As explained before, we can view the Floyd-Warshall algorithm as performing *local* constraints propagations of the form:

$$V'_j - V'_k \leq c \quad \text{and} \quad V'_k - V'_i \leq d \quad \Longrightarrow \quad V'_j - V'_i \leq c + d$$

on $\mathcal{V}'$ until no further propagation can be done. Our idea is to add a second form of local constraints propagation:

$$V'_j - V'_{\bar{\jmath}} \leq c \quad \text{and} \quad V'_{\bar{\imath}} - V'_i \leq d \quad \Longrightarrow \quad V'_j - V'_i \leq (c + d)/2$$

that is, replacing $\mathbf{m}_{ij}$ with $\min(\mathbf{m}_{ij}, (\mathbf{m}_{i\bar{\imath}} + \mathbf{m}_{\bar{\jmath}j})/2)$. This second transformation is valid because we are interested only in points in $\mathbb{I}^{\mathcal{V}'}$ such that $V'_i = -V'_{\bar{\imath}}$. On $\mathcal{V}$, it corresponds to adding the two unary constraints $-2V_i \leq c$ and $2V_j \leq d$ to derive the binary constraint $V_j - V_i \leq (c + d)/2$. Also, the second transformation works on pairs of edges that do not form a path in the potential graph, and hence, cannot be reduced to the first transformation. This is exemplified in Fig. 8.

*Formalization.* A DBM in $\mathbb{R}$ or $\mathbb{Q}$ that is stable by our two local transformations will be said to be *strongly closed*. This is formalized as follows:

*Definition 1.* $\mathbf{m}$ is strongly closed if and only if:

$$\begin{cases} \forall i, j, k & \mathbf{m}_{ij} \leq \mathbf{m}_{ik} + \mathbf{m}_{kj} \\ \forall i, j & \mathbf{m}_{ij} \leq (\mathbf{m}_{i\bar{\imath}} + \mathbf{m}_{\bar{\jmath}j})/2 \\ \forall i & \mathbf{m}_{ii} = 0 \end{cases}$$

As for the emptiness test of Theorem 2, we restrict ourselves to the case $\mathbb{I} \neq \mathbb{Z}$. Indeed, our definition of strong closure uses a division by 2 which is ill-defined on integers. The precise treatment of the case $\mathbb{I} = \mathbb{Z}$ is postponed to Section 3.5.

*Saturation.* Strongly closed DBMs exhibit a saturation property, that is, every octagonal constraint in a strongly closed DBM defines a half-space that actually touches the octagon:

**Theorem 3.** *If* $\mathbb{I} \in \{\mathbb{Q}, \mathbb{R}\}$ *and* $\mathbf{m}$ *is strongly closed, then:*

1. $\forall i, j,$ *if* $\mathbf{m}_{ij} < +\infty,$ *then* $\exists (v_1, \ldots, v_n) \in \gamma^{Oct}(\mathbf{m})$ *such that* $v'_j - v'_i = \mathbf{m}_{ij}$, *and*
2. $\forall i, j,$ *if* $\mathbf{m}_{ij} = +\infty,$ *then* $\forall M < +\infty,\ \exists (v_1, \ldots, v_n) \in \gamma^{Oct}(\mathbf{m})$ *such that* $v'_j - v'_i \geq M$,

*where the* $v'_k$ *are derived from the* $v_k$ *by* $v'_{2k-1} \stackrel{\text{def}}{=} v_k$ *and* $v'_{2k} \stackrel{\text{def}}{=} -v_k$.

This property of strongly closed DBMs will be used pervasively in our subsequent proofs: it provides a strong link between octagons and their representations.

*Best representation.* A first consequence of the saturation property is that there is a unique strongly closed DBM for any non-empty octagon $\gamma^{Oct}(\mathbf{m})$. We will denote it by $\mathbf{m}^\bullet$ and call it $\mathbf{m}$'s *strong closure*. It is the normal form we seek:

**Theorem 4.** *If* $\mathbb{I} \in \{\mathbb{Q}, \mathbb{R}\}$ *and* $\gamma^{Oct}(\mathbf{m}) \neq \emptyset$, *then:*

$$\begin{aligned}\mathbf{m}^\bullet &= (\alpha^{Oct} \circ \gamma^{Oct})(\mathbf{m}) \\ &= \inf_{\sqsubseteq^{\mathsf{DBM}}} \{X^\sharp \in \mathsf{DBM} \mid \gamma^{Oct}(\mathbf{m}) = \gamma^{Oct}(X^\sharp)\}.\end{aligned}$$

In the following section, we will see that $\mathbf{m}^\bullet$ always exists and can be computed in cubic time whenever $\gamma^{Oct}(\mathbf{m}) \neq \emptyset$. If we take care to extend $\bullet$ so that $\mathbf{m}^\bullet = \bot^{\mathsf{DBM}}$ whenever $\gamma^{Oct}(\mathbf{m}) = \emptyset$, then Theorem 4 is true for all elements in $\mathsf{CDBM}$. It is important to note that, while $\alpha^{Oct}$ is only a partial function when $\mathbb{I} = \mathbb{Q}$, $\alpha^{Oct} \circ \gamma^{Oct}$ is still always well-defined.

*Representation redundancy.* As for the shortest-path closure of DBMs representing potential sets, the effect of the strong closure is to make explicit all the implicit constraints. Thus, the normal form we choose to represent an octagon may contain many redundant constraints, and very few $+\infty$ matrix elements. This is unlike other relational abstract domains, such as Karr's linear equality domain [35] or the polyhedron domain [22], that always choose to remove as many redundant constraints as possible. This explains why, in some cases where a small number of linear constraints is sufficient to perform a program analysis, the polyhedron domain may use less memory and be faster than the octagon domain. Experience shows, however, that, in many cases, the polyhedron representations grow exponentially, while the octagon domain guarantees a quadratic representation size in the worst case.

### 3.4. Floyd-Warshall algorithm for strong closure

We now present a modified version of the Floyd-Warshall algorithm that uses our two local transformations to compute $\mathbf{m}^\bullet$ in cubic time:

*Definition 2.* The modified Floyd-Warshall algorithm is defined as follows:

$$(\mathbf{m}^\bullet)_{ij} \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } i = j \\ \mathbf{m}^n_{ij} & \text{if } i \neq j \end{cases}$$

where
$$\mathbf{m}^k \stackrel{\text{def}}{=} \begin{cases} \mathbf{m} & \text{if } k = 0 \\ S(C^{2k-1}(\mathbf{m}^{k-1})) & \text{if } 1 \leq k \leq n \end{cases}$$

and
$$(S(\mathbf{n}))_{ij} \stackrel{\text{def}}{=} \min(\mathbf{n}_{ij}, (\mathbf{n}_{i\bar{\imath}} + \mathbf{n}_{\bar{\jmath}j})/2)$$

and
$$\left(C^k(\mathbf{n})\right)_{ij} \stackrel{\text{def}}{=} \min(\mathbf{n}_{ij},\ \mathbf{n}_{ik} + \mathbf{n}_{kj},\ \mathbf{n}_{i\bar{k}} + \mathbf{n}_{\bar{k}j},\ \mathbf{n}_{ik} + \mathbf{n}_{k\bar{k}} + \mathbf{n}_{\bar{k}j},\ \mathbf{n}_{i\bar{k}} + \mathbf{n}_{\bar{k}k} + \mathbf{n}_{kj})$$

As the classical Floyd-Warshall algorithm, this algorithm performs $n$ steps. Each step computes a new matrix in quadratic time. However, each step now uses two passes: a $S$ pass and a $C^k$ pass. We recognize in $S$ our second local transformation, pictured in Fig. 8. $C^k$ looks like an inflated version of the classical Floyd-Warshall local transformation of Fig. 6. We check whether there is a shorter path from $i$ to $j$ via $k$. We also check for a shorter path via $\bar{k}$. Finally we check for a shorter path via $k$ and then $\bar{k}$, and also via $\bar{k}$ and then $k$. This increase in complexity as well as the interleaving of the $S$ and $C^k$ passes is important to ensure that the local characterization of the strong closure is attained for more and more elements; that is, to ensure that what is enforced by one pass is not destroyed by a later one. Alas, there does not seem to exist a simple and intuitive reason for the exact formulas presented in Definition 2.[1] It should be considered as a necessary technicality for the proof of the following theorem:

**Theorem 5.** *If $\gamma^{Oct}(\mathbf{m}) \neq \emptyset$ then $\mathbf{m}^\bullet$ computed by Definition 2 is the strong closure as defined by Definition 1.*

Not only does this algorithm compute the strong closure $\mathbf{m}^\bullet$ for any DBM $\mathbf{m}$ that represents a non-empty octagon, but it can also be used to determine whether a DBM represents an empty octagon:

**Theorem 6.** $\gamma^{Oct}(\mathbf{m}) = \emptyset \iff \exists i,\ \mathbf{m}^n_{ii} < 0$, *where $\mathbf{m}^n$ is defined as in Definition 2.*

*In-place implementation.* In Definition 2, $\mathbf{m}^\bullet$ is defined using $2n$ intermediate matrices: $\mathbf{m}^k$ and $C^{2k-1}(\mathbf{m}^k)$ for each $1 \leq k \leq n$. From a practical implementation point of view, allocating all these $2n$ DBMs is a waste of memory. A first optimization lies in the observation that only two matrices are needed at any given time as each intermediate matrix is defined solely using the last computed one. We can do even better: we can update the initial matrix in-place without any extra storage requirement, as implemented by the algorithm of Fig. 9. The intermediate matrix at step $k$ may be different from the

[1] During the final writing of the present paper, a simpler—yet still cubic—strong closure algorithm has been proposed by Bagnara et al. [5].

```
StrongClosure (DBM m of size 2n × 2n)

for k = 1 to n {
  for i = 1 to 2n
    for j = 1 to 2n
      m_ij ← min ( m_ij, m_i(2k−1) + m_(2k−1)j, m_i(2k) + m_(2k)j,
                   m_i(2k−1) + m_(2k−1)(2k) + m_(2k)j,
                   m_i(2k) + m_(2k)(2k−1) + m_(2k−1)j )
  for i = 1 to 2n
    for j = 1 to 2n
      m_ij ← min(m_ij, (m_iī + m_j̄j)/2)
}

for i = 1 to 2n
  if m_ii < 0 then return(⊥^DBM) else m_ii ← 0
return(m)
```

**Fig. 9** In-place modified Floyd-Warshall algorithm to compute the strong closure of a DBM

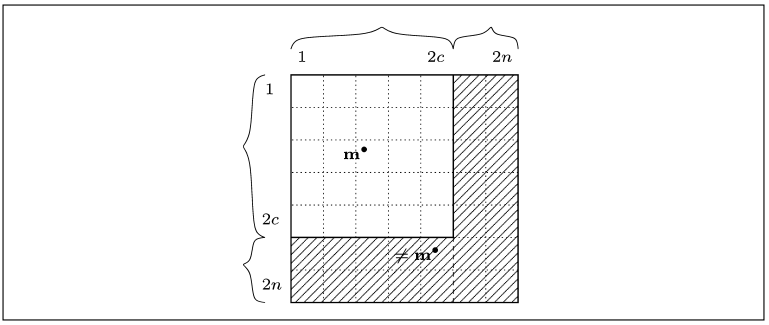


**Fig. 10** A matrix equal to $\mathbf{m}^\bullet$ except for the last $2(n-c)$ lines and columns

corresponding one in Definition 2, but the overall result $\mathbf{m}^\bullet$ is indeed the same. Although this new algorithm is much nicer from an implementation point of view, we will keep reasoning using the original version of Definition 2 which is much simpler to describe mathematically.

*Incremental version.* We now propose an *incremental* version of our modified Floyd-Warshall algorithm that is able to quickly compute the strong closure of matrices that are "almost" strongly closed. Suppose that $\mathbf{m}$ is strongly closed and that $\mathbf{n}_{ij} = \mathbf{m}_{ij}$ for all $i, j \leq 2c$, which is sketched in Fig. 10. From a constraint point of view, this means that we may only have altered unary constraints on variables in $V_{c+1}, \ldots, V_n$ and binary constraints such that one or both variables are in $V_{c+1}, \ldots, V_n$. Then, many computations in Definition 2 are useless and we can use the following faster algorithm to compute $\mathbf{n}^\bullet$:

$$(\mathbf{n}^\bullet)_{ij} \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } i = j \\ \mathbf{n}^n_{ij} & \text{if } i \neq j \end{cases}$$

where
$$\mathbf{n}^k \stackrel{\text{def}}{=} \begin{cases} \mathbf{n} & \text{if } k = 0 \\ S'^{2k-1}(C'^{2k-1}(\mathbf{n}^{k-1})) & \text{if } 1 \leq k \leq n \end{cases}$$

and
$$(S'^k(\mathbf{n}))_{ij} \stackrel{\text{def}}{=} \begin{cases} \mathbf{n}_{ij} & \text{if } i, j, k \leq 2c \\ \min(\mathbf{n}_{ij}, (\mathbf{n}_{i\bar{\imath}} + \mathbf{n}_{\bar{\jmath}j})/2) & \text{otherwise} \end{cases}$$

and
$$(C'^k(\mathbf{n}))_{ij} \stackrel{\text{def}}{=} \begin{cases} \mathbf{n}_{ij} & \text{if } i, j, k \leq 2c \\ \min\ (\mathbf{n}_{ij},\ \mathbf{n}_{ik} + \mathbf{n}_{kj}, & \text{otherwise} \\ \qquad \mathbf{n}_{i\bar{k}} + \mathbf{n}_{\bar{k}j}, & \\ \qquad \mathbf{n}_{ik} + \mathbf{n}_{k\bar{k}} + \mathbf{n}_{\bar{k}j}, & \\ \qquad \mathbf{n}_{i\bar{k}} + \mathbf{n}_{\bar{k}k} + \mathbf{n}_{kj}) & \end{cases}$$

As the sub-matrix from indices (1, 1) to $(2c, 2c)$ is left unmodified for the first $c$ iterations, we have a time cost proportional to $(n - c) \times n^2$. By virtually switching columns and lines, this algorithm extends to the case where the $n - c$ pairs of modified lines and columns are anywhere in the matrix, not necessarily at the end. We will denote by $Inc^\bullet_{i_1,\ldots,i_k}(\mathbf{n})$ the result of the algorithm when the modified lines and columns correspond to variables $V_{i_1}$ to $V_{i_k}$. One particularly useful instance is $Inc^\bullet_i$ that recovers, in quadratic time, the strong closure after one or several constraints involving $V_i$ have been modified. It is important to remark that $Inc^\bullet_{i_1,i_2}$ is not equivalent to $Inc^\bullet_{i_1} \circ Inc^\bullet_{i_2}$: our incremental strong closure must treat all the modified lines and columns at once. Finally, note that an in-place version of this incremental algorithm, in the spirit of Fig. 9, may be easily designed.

### 3.5. Integer case

Whenever $\mathbb{I} = \mathbb{Z}$, the emptiness test of Theorem 2 no longer works. Moreover, the strong closure of Definition 1 does no longer correspond to a normal form enjoying the saturation property, neither does the result of the modified Floyd-Warshall algorithm of Definition 2.

In [33], Jaffar et al. propose to consider constraint conjunctions that are not only closed by *transitivity* (that is, the addition of two constraints) but also by *tightening*, a new operation that allows deriving the constraint $x \leq \lfloor c/2 \rfloor$ from the constraint $2x \leq c$. They prove that constraint systems closed by transitivity and tightening are satisfiable if and only if no trivially unsatisfiable constraint appears in the system—such as $0 \leq c$, where the constant $c$ is strictly negative. Later, in [32], Harvey and Stuckey propose a practical algorithm to maintain the tightened transitive closure of a constraint system when new constraints are added. Even though [32, 33] are only interested in checking the satisfiability of constraint systems and not in constructing abstract domains, their ideas can be of use to construct a normal form enjoying the saturation property, which will prove sufficient for our needs.

*Strong closure with tightening.* We first restate the notion of tightened transitive closure from [32, 33] using our encoding of constraint conjunctions as DBMs:

*Definition 3.* A coherent DBM $\mathbf{m}$ in $\mathbb{Z}$ is *tightly closed* if and only if:

$$\begin{cases} \forall i, j, k, & \mathbf{m}_{ij} \leq \mathbf{m}_{ik} + \mathbf{m}_{kj} \\ \forall i, j, & \mathbf{m}_{ij} \leq (\mathbf{m}_{i\bar{\imath}} + \mathbf{m}_{\bar{\jmath}j})/2 \\ \forall i, & \mathbf{m}_{i\bar{\imath}} \text{ is even} \\ \forall i, & \mathbf{m}_{ii} = 0 \end{cases}$$

This simply amounts to stating that $\mathbf{m}$ is strongly closed with the extra requirement that all the elements $\mathbf{m}_{i\bar{\imath}}$ are even. Indeed, such elements correspond to bounds of expressions of the form $\pm 2V_k$.

An important theoretical contribution of this article is the saturation property of tightly closed DBMs in $\mathbb{Z}$ and, as a consequence, its normal form property:

**Theorem 7.** *Theorems 3 and 4 are true on tightly closed DBMs.*

*Harvey and Stuckey algorithm.* We now discuss the algorithm proposed by Harvey and Stuckey in [32] to compute the tight closure. It is an incremental algorithm: given a tightly closed set of octagonal constraints and an additional octagonal constraint, it is able to compute, in quadratic time, the tight closure of the set enriched with the new constraint. Note that this incremental algorithm is less powerful than our incremental strong closure algorithm $Inc^{\bullet}$. Indeed, $Inc^{\bullet}$ was able to recover, in quadratic time, the strong closure of a constraint set after *all* constraints related to one variable have been changed, not just one.

We now adapt the algorithm by Harvey and Stuckey to our encoding of octagonal constraints as DBMs. Suppose that the coherent DBM $\mathbf{m}$ is equal to a tightly closed DBM, except for the element at position $(i_0 j_0)$—and, by coherence, the element at position $(\overline{j_0}\,\overline{i_0})$. Moreover, suppose that, if $i_0 = \overline{j_0}$, then the changed element $\mathbf{m}_{i_0 j_0} = \mathbf{m}_{\overline{j_0}\,\overline{i_0}}$ is even. The incremental tight closure on $\mathbf{m}$ with respect to the position $(i_0, j_0)$ is denoted by $Inc^{T}_{i_0 j_0}(\mathbf{m})$ and defined as follows:

*Definition 4.*

$$\left(Inc^{T}_{i_0 j_0}(\mathbf{m})\right)_{ij} \stackrel{\text{def}}{=} \min\left(\mathbf{m}'_{ij},\ (\mathbf{m}'_{i\bar{\imath}} + \mathbf{m}'_{\bar{\jmath}j})/2\right)$$

where

$$\begin{aligned} \mathbf{m}'_{ij} \stackrel{\text{def}}{=} \min\big(&\mathbf{m}_{ij},\ \mathbf{m}_{ii_0} + \mathbf{m}_{i_0 j_0} + \mathbf{m}_{j_0 j}, && \text{if } i \neq \overline{j} \\ &\mathbf{m}_{i\overline{j_0}} + \mathbf{m}_{\overline{j_0}\,\overline{i_0}} + \mathbf{m}_{\overline{i_0} j}\big) \\ \mathbf{m}'_{ij} \stackrel{\text{def}}{=} \min\big(&\mathbf{m}_{ij},\ 2(\mathbf{m}_{i_0 j_0} + \mathbf{m}_{i\overline{j_0}} + (\mathbf{m}_{\overline{i_0} i_0}/2)), && \text{if } i = \overline{j} \\ &2(\mathbf{m}_{i_0 j_0} + \mathbf{m}_{ii_0} + (\mathbf{m}_{j_0\overline{j_0}}/2)), \\ &2\lfloor(\mathbf{m}_{ii_0} + \mathbf{m}_{i_0 j_0} + \mathbf{m}_{j_0\bar{\imath}})/2\rfloor\big) \end{aligned}$$

In $Inc^{T}_{i_0 j_0}$, we first propagate the new constraint $\mathbf{m}_{i_0 j_0}$ to obtain new unary and binary constraints in $\mathbf{m}'$. Note that the terms $2(\mathbf{m}_{i_0 j_0} + \mathbf{m}_{i\overline{j_0}} + (\mathbf{m}_{\overline{i_0} i_0}/2))$ and $2(\mathbf{m}_{i_0 j_0} + \mathbf{m}_{ii_0} + (\mathbf{m}_{j_0\overline{j_0}}/2))$ correspond respectively to the sum along the paths $\langle i, \overline{j_0}, \overline{i_0}, i_0, j_0, \bar{\imath}\rangle$ and $\langle i, i_0, j_0, \overline{j_0}, \overline{i_0}, \bar{\imath}\rangle$. We use *tightening* for the third derived unary constraint: $\mathbf{m}'_{i\bar{\imath}} \leq 2\lfloor(\mathbf{m}_{ii_0} + \mathbf{m}_{i_0 j_0} + \mathbf{m}_{j_0\bar{\imath}})/2\rfloor$.

In $\mathbf{m}'$, all the derived matrix coefficients corresponding to unary constraints are even. Finally, the new unary constraints are combined to derive new binary constraints: $(Inc^T_{i_0 j_0}(\mathbf{m}))_{ij} \leq (\mathbf{m}'_{i\bar{i}} + \mathbf{m}'_{\bar{j}j})/2$. Whenever the result of a division by 2 is not fed to the floor operator $\lfloor\cdot\rfloor$, it means that the division cannot produce half-integers.

*Cost considerations.* The incremental tight closure algorithm has a $\mathcal{O}(n^2)$ cost. In order to obtain the tight closure of an arbitrary DBM $\mathbf{m}$, we must start from $\top^{\mathsf{DBM}}$ and add all the constraints $\mathbf{m}_{ij}$ one by one and perform an incremental tight closure $Inc^T_{ij}$ after each addition. This leads to a $\mathcal{O}(n^4)$ total cost while our strong closure algorithm had a $\mathcal{O}(n^3)$ cost. It is not known to the author whether a better cost than $\mathcal{O}(n^4)$ can be achieved. This may be impossible as integer problems tend to be strictly more difficult than problems involving rationals or reals. Recall, for instance, that integer linear programming problems are NP-complete while rational linear programming problems have a polynomial complexity.

If time cost is a concern, one may consider using the original strong closure algorithm—without tightening—where the $S$ pass has been changed into:

$$(S(\mathbf{n}))_{ij} \stackrel{\text{def}}{=} \min(\mathbf{n}_{ij}, \lfloor(\mathbf{n}_{i\bar{i}} + \mathbf{n}_{\bar{j}j})/2\rfloor) \qquad (1)$$

or, better, into:

$$(S(\mathbf{n}))_{ij} \stackrel{\text{def}}{=} \begin{cases} \min(\mathbf{n}_{ij}, \lfloor(\mathbf{n}_{i\bar{i}} + \mathbf{n}_{\bar{j}j})/2\rfloor) & \text{if } i \neq \overline{j} \\ 2\lfloor \mathbf{n}_{ij}/2 \rfloor & \text{if } i = \overline{j} \end{cases} \qquad (2)$$

which is more precise than (1) and ensures that unary constraints are tight. Both resulting strong closure algorithms indeed return a DBM $\mathbf{m}^\bullet$ in $\mathbb{Z}$ such that $\gamma^{Oct}(\mathbf{m}^\bullet) = \gamma^{Oct}(\mathbf{m})$ and which is much smaller than $\mathbf{m}$ with respect to $\sqsubseteq^{\mathsf{DBM}}$. However, they do not return the *smallest* one as none of the modified $S$ functions preserve the transitive closure property enforced by the $C$ steps. As a consequence, the saturation property is not attained. This will affect most of the operators and transfer functions that will be presented in the rest of the paper. Our inclusion and equality tests will become semi-tests that can fail to detect that $\gamma^{Oct}(\mathbf{m}) \subseteq \gamma^{Oct}(\mathbf{n})$ or $\gamma^{Oct}(\mathbf{m}) = \gamma^{Oct}(\mathbf{n})$. Our abstractions for the union and the forget operators—among others—will not be the best ones, etc. They will *remain sound* in every situation, but they will not be as precise as they might be. This loosely amounts to abstracting integers as rationals by forgetting their "integral property," something which is commonly done in the polyhedron abstract domain. Whether to choose the strong closure or the tight closure when $\mathbb{I} = \mathbb{Z}$ becomes a cost versus precision trade-off.

In our practical experiments, we have opted in favour of running time: we use the modified strong closure (1) instead of the tight closure. We have yet to find a real-life example where the tight closure is needed to prove a meaningful invariant. In the following, we will focus on properties of strongly closed matrices on $\mathbb{Q}$ and $\mathbb{R}$, and leave implicit the fact that all these properties are also true for tightened matrices on $\mathbb{Z}$, thanks to Theorem 7, but not if the cubic strong closure is used when $\mathbb{I} = \mathbb{Z}$.

### 3.6. Summary of the closure operators

We have introduced several closure operators, with different scopes and costs. Some work on rationals and reals while others work on integers. Some are incremental, others are not. We recall them in the following table. For comparison purposes, we have also included the

corresponding operators on DBMs representing potential sets. For each operator, we include, in order, the cost of incrementally updating one constraint, of incrementally updating all constraints relating to one variable, and of updating all constraints.

| operator | domain | $\mathbb{I}$ | costs |
|---|---|---|---|
| closure | potential sets | $\mathbb{Z}, \mathbb{R}, \mathbb{Q}$ | $\mathcal{O}(n^2)$ / $\mathcal{O}(n^2)$ / $\mathcal{O}(n^3)$ |
| strong closure | octagons | $\mathbb{R}, \mathbb{Q}$ | $\mathcal{O}(n^2)$ / $\mathcal{O}(n^2)$ / $\mathcal{O}(n^3)$ |
| tight closure | octagons | $\mathbb{Z}$ | $\mathcal{O}(n^2)$ / $\mathcal{O}(n^3)$ / $\mathcal{O}(n^4)$ |

### 3.7. Equality and inclusion testing

We are able to compare two DBMs $\mathbf{m}$ and $\mathbf{n}$ using the $\sqsubseteq^{\mathsf{DBM}}$ order, but this does not always allow comparing two octagons as $\gamma^{Oct}$ is not one-to-one. The following theorems use the properties of the strong closure to solve this problem:

**Theorem 8.** $\mathbf{m}^\bullet = \mathbf{n}^\bullet \iff \gamma^{Oct}(\mathbf{m}) = \gamma^{Oct}(\mathbf{n})$.

**Theorem 9.** $\mathbf{m}^\bullet \sqsubseteq^{\mathsf{DBM}} \mathbf{n} \iff \gamma^{Oct}(\mathbf{m}) \subseteq \gamma^{Oct}(\mathbf{n})$.

Note that, when testing for inclusion, it is not necessary to close the right argument. Testing for equality or inclusion is done point-wise, and hence, has a quadratic cost, not counting the cost of the strong closure.

## 4. Abstract transfer functions

The concrete semantics of a program is the most precise formal expression of its behavior. It is generally not computable and defined by combining a small fixed set of generic semantical functions, so-called *transfer functions*, that model the effect on sets of environments of basic instructions such as assignments, tests, and control-flow joins. In order to derive a computable static analysis, we need to define a *sound* abstract counterpart in our octagon domain for each of these transfer functions. More formally, if $F$ is a $n$-array transfer function in the set of concrete environments $\mathcal{P}(\mathcal{V} \to \mathbb{I})$, we must design an abstract counterpart $F^\sharp$ in $\mathsf{CDBM}$ such that:

$$\begin{aligned} &\forall X_1^\sharp, \ldots, X_n^\sharp \in \mathsf{CDBM}, \\ &F(\gamma^{Oct}(X_1^\sharp), \ldots, \gamma^{Oct}(X_n^\sharp)) \;\subseteq\; (\gamma^{Oct} \circ F^\sharp)(X_1^\sharp, \ldots, X_n^\sharp). \end{aligned}$$

Whenever the inclusion is an equality, we say that the abstraction is *exact*. This is seldom the case because the result of a concrete operator is rarely exactly representable in the octagon abstract domain, even when all its argument are. Whenever $(\alpha^{Oct} \circ F)(\gamma^{Oct}(X_1^\sharp), \ldots, \gamma^{Oct}(X_n^\sharp))$ exists—which may not always be the case when $\mathbb{I} = \mathbb{Q}$ as $\alpha^{Oct}$ is partial—it defines the *best* abstraction. It is the smallest DBM with respect to $\sqsubseteq^{\mathsf{DBM}}$ that over-approximates the result of $F$. As a consequence, it represents the smallest octagon encompassing the concrete result and induces as few spurious program behaviors as possible. Sometimes, however, even when the best abstraction exists, it may be too costly or too complex to compute. In that case, one can settle for a non-optimal abstraction. A table of all the abstract transfer

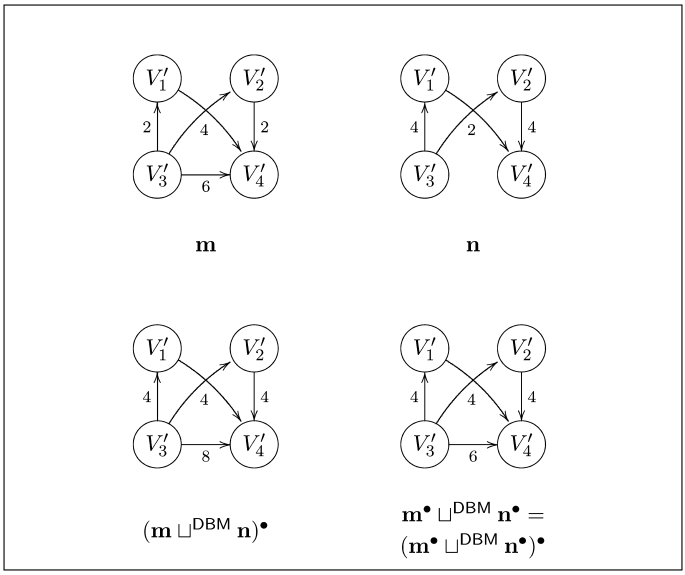


**Fig. 11** Abstract union of octagons, based on $\sqcup^{\mathsf{DBM}}$. DBMs should be strongly closed for best precision. This also ensures that the result is strongly closed

functions introduced in this section, together with a summary of their properties, is available in Section 4.8.

### 4.1. Abstract union and intersection

We start with abstractions of set-theoretic operators. The abstract union of environments is particularly important as it is used to model control-flow joins occurring when the two branches of a conditional meet and in loops.

*Union abstraction.* The union $\cup$ of two octagons may not be an octagon. Indeed, octagons are always convex, which is not a property preserved by union. By monotonicity of $\gamma^{Oct}$, $\mathbf{m} \sqcup^{\mathsf{DBM}} \mathbf{n}$ gives a sound abstraction of $\gamma^{Oct}(\mathbf{m}) \cup \gamma^{Oct}(\mathbf{n})$. A less obvious fact is that the precision of this operator greatly depends on which DBM arguments are used among all DBMs that represent the same octagons. In particular, the *best* abstraction for the union is only reached when the arguments are strongly closed DBMs, as illustrated in Fig. 11. Thus, we define our union abstraction as follows:

$$\mathbf{m} \cup^{Oct} \mathbf{n} \stackrel{\text{def}}{=} (\mathbf{m}^\bullet) \sqcup^{\mathsf{DBM}} (\mathbf{n}^\bullet).$$

and we have the following theorem:

**Theorem 10.**

$$\gamma^{Oct}(\mathbf{m} \cup^{Oct} \mathbf{n}) = \inf_{\subseteq} \{S \in Oct \mid S \supseteq \gamma^{Oct}(\mathbf{m}) \cup \gamma^{Oct}(\mathbf{n})\}.$$

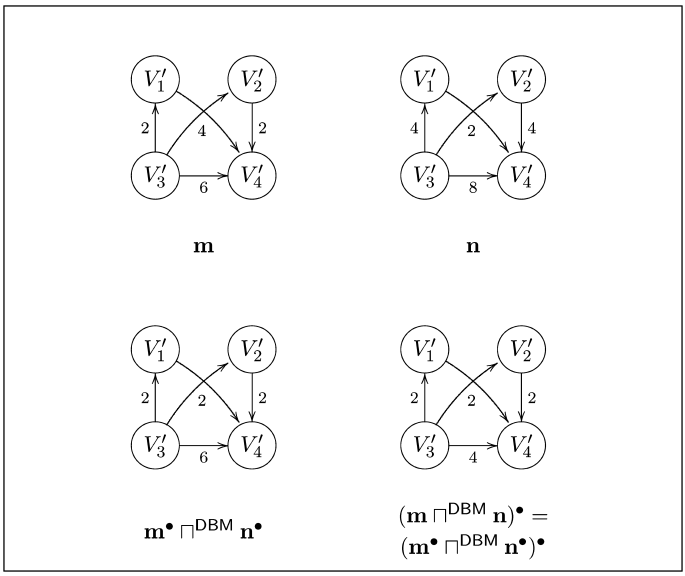


**Fig. 12** Exact intersection of octagons, based on $\sqcap^{\mathsf{DBM}}$. The arguments do not need to be strongly closed, and the result is seldom strongly closed

Another remarkable property is that $\sqcup^{\mathsf{DBM}}$ preserves the strong closure property:

**Theorem 11.** $\mathbf{m} \cup^{Oct} \mathbf{n}$ *is strongly closed.*

*Exact intersection.* The intersection of two octagons is always an octagon. The $\sqcap^{\mathsf{DBM}}$ operator always computes a DBM representing the exact intersection of two octagons, even when the DBM arguments are not strongly closed, as shown in Fig. 12, so, we define $\cap^{Oct}$ as $\sqcap^{\mathsf{DBM}}$ and we have:

**Theorem 12.** $\gamma^{Oct}(\mathbf{m} \cap^{Oct} \mathbf{n}) = \gamma^{Oct}(\mathbf{m}) \cap \gamma^{Oct}(\mathbf{n})$.

It is important to remark that the result of an intersection is seldom closed, even when the arguments are, as demonstrated in Fig. 12.

*Comparison with other work.* The intersection of DBMs representing potential sets has been used for a long time by the model-checking community, but no abstract union operator was defined. The idea of computing the point-wise maximum of upper bounds to compute the union of octagons is already present in the work of Balasundaram and Kennedy [6]. Although the authors remark that the bounds should be the tightest possible for the union to be precise while the result of an intersection may have loose bounds, they do not propose any way to actually "tighten" them. Thus, our strong closure algorithm is the key to obtaining an effective best union approximation.

Note that, in the model-checking community, it is traditional to perform only exact computations on a computable abstract model of the chosen program. In order to do this, one must represent exactly both conjunctions and disjunctions of potential constraints. One naive

solution [24, 54] is to use a *set* of DBMs to represent a symbolic union of potential sets. The problem is that such explicit representations tend to grow very large, and testing the inclusion and equality of two representations is costly. To address these problems, several alternate data structures have been proposed, based on the concept of decision diagrams. Two examples are *Clock Difference Diagrams* [36] and *Difference Decision Diagrams* [47]. Despite the lack of a canonical form for both data structures, inclusion, equality, and emptiness testing algorithms are proposed. It might be interesting, from a theoretical point of view, to see if these data-structures can be adapted to octagonal constraints. Another issue is the adaptation, to these decision diagrams, of the extra abstract transfer functions not provided by the model-checking community but required by an abstract domain—such as general assignments (Section 4.4) or widenings (Section 4.7). Alas, exact unions mean unbounded memory and time costs, regardless on how clever the chosen representation is. Recall, however, that we do not seek here exactness at any cost, but we strive for scalability. Instead of representing unions exactly, we advocate for the use of *partitioning techniques*. They can lessen the precision degradation due to inexact abstract unions by performing case analyses, while having a bounded and predictable cost. Also, they do not require any change in the abstract element representation nor the abstract transfer functions. Such techniques were successfully used within the ASTRÉE analyzer. We will not develop further this topic and refer the reader to [39] for more information. Other works on partitioning techniques include that of Handjieva and Tzolovski [31], and that of Bourdoncle [10].

### 4.2. Forget operator

From now on and up to Section 4.6, included, all the operators and transfer functions presented on octagons will abstract *strict* concrete ones. To simplify our presentation, we will thus present the image of non-$\bot^{\mathsf{DBM}}$ octagons, silently assuming that the image of $\bot^{\mathsf{DBM}}$ is always $\bot^{\mathsf{DBM}}$.

Given a concrete set of environments $R \in \mathcal{P}(\mathcal{V} \to \mathbb{I})$ and a variable $V_f \in \mathcal{V}$, the *forget operator* $\{\!|\, V_f \leftarrow ?\, |\!\}$ models the assignment of a non-deterministic value into $V_f$:

$$\begin{aligned} \{\!|\, V_f \leftarrow ?\, |\!\}(R) &\stackrel{\text{def}}{=} \{\rho[V_f \mapsto v] \mid \rho \in R,\ v \in \mathbb{I}\} \\ &= \{\rho \mid \exists v \in \mathbb{I},\ \rho[V_f \mapsto v] \in R\} \end{aligned}$$

where $\rho[V_f \mapsto v]$ is the function equal to $\rho$ except that it maps $V_f$ to $v$ instead of $\rho(v)$. Geometrically, this corresponds to a projection on $\mathcal{V} \setminus \{V_f\}$, followed by an extrusion along the dimension of $V_f$.

In order to implement an abstract counterpart $\{\!|\, V_f \leftarrow ?\, |\!\}^{Oct}$ in the octagon domain for the forget operator, a first idea is to simply remove all constraints involving $V_f$. Using our octagon encodings in terms of potential constraints, this amounts to removing all constraints involving $V'_{2f-1}$ and $V'_{2f}$:

$$(\{\!|\, V_f \leftarrow ?\, |\!\}^{Oct}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=} \begin{cases} \mathbf{m}_{ij} & \text{if } i \neq 2f-1, 2f \text{ and } j \neq 2f-1, 2f \\ 0 & \text{if } i = j = 2f-1 \text{ or } i = j = 2f \\ +\infty & \text{otherwise} \end{cases}$$

This operator is always sound, as proved by the following theorem:

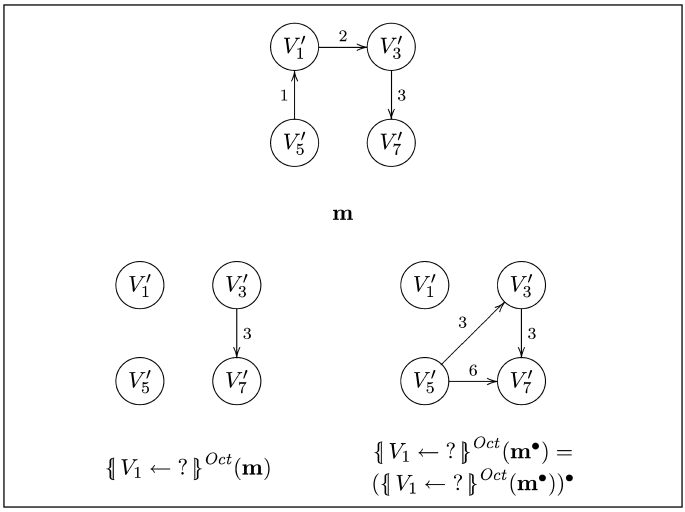


**Fig. 13** Forget operator on octagons—the potential graph parts involving $V'_2$, $V'_4$, $V'_6$, and $V'_8$ have been omitted for the sake of conciseness but can be easily recovered by coherence

**Theorem 13.** $\gamma^{Oct}(\{\!\{ V_f \leftarrow ? \}\!\}^{Oct}(\mathbf{m})) \supseteq \{\!\{ V_f \leftarrow ? \}\!\}(\gamma^{Oct}(\mathbf{m}))$.

Moreover, when the argument is strongly closed, it is exact:

**Theorem 14.** $\gamma^{Oct}(\{\!\{ V_f \leftarrow ? \}\!\}^{Oct}(\mathbf{m}^\bullet)) = \{\!\{ V_f \leftarrow ? \}\!\}(\gamma^{Oct}(\mathbf{m}))$.

Whenever the argument is not strongly closed, the result may not be as good. The intuitive reason is that, by forgetting constraints involving $V'_{2f-1}$ and $V'_{2f}$, we may also forget some implicit constraints on unmodified variables as we break all former paths passing through $V'_{2f-1}$ or $V'_{2f}$. If the argument is strongly closed, however, all implicit constraints have been made explicit and this problem does not occur. This is exemplified in Fig. 13. A final remark is that the forget operator preserves the strong closure:

**Theorem 15.** $\{\!\{ V_f \leftarrow ? \}\!\}^{Oct}(\mathbf{m})$ *is strongly closed whenever* $\mathbf{m}$ *is.*

The forget operator is quite useful as a fall-back assignment operator: $\{\!\{ V_i \leftarrow ? \}\!\}^{Oct}$ is always a sound abstraction for any assignment $\{\!\{ V_i \leftarrow expr \}\!\}$, regardless of the assigned expression *expr*. It is also a sound abstraction for any backward assignment $\{\!\{ V_i \rightarrow expr \}\!\}$—see Section 4.6. Finally, it can be used directly to abstract soundly the effect of unknown code portions—e.g., unknown libraries—or inputs from the environment—e.g., file data or key strokes.

### 4.3. Conversion operators

We now present operators for converting between octagons, intervals, and polyhedra. One application is to allow a static analyzer to switch dynamically between these abstract domains to adjust the cost versus precision trade-off. Another application is to design octagon transfer functions by switching momentarily into another abstract domain where this transfer function

already exists and converting the result back into the octagon domain. Conversions from intervals to octagons, and from octagons to polyhedra, are exact. Conversions from polyhedra to octagons and from octagons to intervals cannot be exact and generally induce an over-approximation.

*From intervals to octagons.* All interval abstract domain elements are exactly representable as octagons. Given an interval element $X^\sharp : \mathcal{V} \to ((\mathbb{I} \cup \{-\infty\}) \times (\mathbb{I} \cup \{+\infty\}))$ that maps each variable to a lower bound and an upper bound, we construct the following DBM containing only unary constraints:

$$(Oct(X^\sharp))_{ij} \stackrel{\text{def}}{=} \begin{cases} 2 \times \text{snd}\,(X^\sharp(V_k)) & \text{if } i = 2k,\ j = 2k-1 \\ -2 \times \text{fst}\,(X^\sharp(V_k)) & \text{if } j = 2k,\ i = 2k-1 \\ +\infty & \text{elsewhere} \end{cases}$$

where the fst and snd operators extract respectively the first and second component of a pair, that is, the lower and upper bounds of an interval.

*From octagons to intervals.* A straightforward application of the saturation property of the strong closure is the ability to easily project an octagon $\mathbf{m}$ onto any variable $V_i$ to obtain an interval, denoted by $\pi_i(\mathbf{m})$ and defined as follows:

$$\pi_i(\mathbf{m}) \stackrel{\text{def}}{=} \begin{cases} \emptyset & \text{if } \mathbf{m}^\bullet = \bot^{\mathsf{DBM}} \\ [-\mathbf{m}^\bullet_{(2i-1)\,(2i)}/2,\ \mathbf{m}^\bullet_{(2i)\,(2i-1)}/2] & \text{if } \mathbf{m}^\bullet \neq \bot^{\mathsf{DBM}} \end{cases}$$

We then have:

**Theorem 16.** $\pi_i(\mathbf{m}) = \{v \in \mathbb{I} \mid \exists (v_1, \ldots, v_n) \in \gamma^{Oct}(\mathbf{m}),\ v_i = v\}$.

An interval abstract domain element, that maps an interval to each variable, is obtained by projecting each variable independently. The result will be denoted by $Int(\mathbf{m})$. If we do not use the strong closure in all the $\pi_i$, we obtain sound intervals that are not as tight as possible. Otherwise, we obtain the *best interval abstraction* of an octagon.

*From octagons to polyhedra.* Converting a DBM $\mathbf{m}$ representing an octagon into a polyhedron is quite easy as a polyhedron can be represented internally as a list of linear inequality constraints. The resulting polyhedron will be denoted by $Poly(\mathbf{m})$.

*From polyhedra to octagons.* Converting a non-empty polyhedron $P$ in $\mathbb{R}^n$ or $\mathbb{Q}^n$ into an octagon is more subtle. Surprisingly, the *frame* representation of $P$ is more handy here than the constraint list representation. We recall that a frame consists in a finite set of vertices $\mathbf{V} = (V_1, \ldots, V_k)$ and a finite set of rays $\mathbf{R} = (R_1, \ldots, R_l)$ in $\mathbb{I}^n$ that represent the polyhedron:

$$\left\{ \sum_{i=1}^{k} \lambda_i V_i + \sum_{i=1}^{l} \mu_i R_i \mid \lambda_i \geq 0,\ \mu_i \geq 0,\ \sum_{i=1}^{k} \lambda_i = 1 \right\}.$$

The intuition behind our conversion is to first consider the smallest octagon containing all vertices in $\mathbf{V}$. This is simply the abstract union $\cup^{Oct}$ of $k$ octagons reduced to a single point.

Then, we set to $+\infty$ the upper bound of any constraint not stable under translation in the direction of all rays in $\mathbf{R}$. For instance, if there is ray with a strictly positive $i$-th coordinate, then the upper bound of all constraints involving $+V_i$ are set to $+\infty$.

More formally, let $v_i$ and $r_i$ denote respectively the $i$-th coordinate of a vertex $V \in \mathbf{V}$ and of a ray $R \in \mathbf{R}$. We generate a DBM $\mathbf{m}$ by applying the following rules, for every $i$ and $j \neq i$:

- if $\exists R \in \mathbf{R}$ such that $r_i > 0$, we set $\mathbf{m}_{(2i)(2i-1)} = +\infty$,
  otherwise, we set $\mathbf{m}_{(2i)(2i-1)} = 2 \max \{v_i \mid V \in \mathbf{V}\}$;
- if $\exists R \in \mathbf{R}$ such that $r_i < 0$, we set $\mathbf{m}_{(2i-1)(2i)} = +\infty$,
  otherwise, we set $\mathbf{m}_{(2i-1)(2i)} = -2 \min \{v_i \mid V \in \mathbf{V}\}$;
- if $\exists R \in \mathbf{R}$ such that $r_i > r_j$, we set

$$\mathbf{m}_{(2i-1)(2j-1)} = \mathbf{m}_{(2j)(2i)} = +\infty$$

  otherwise, we set

$$\mathbf{m}_{(2i-1)(2j-1)} = \mathbf{m}_{(2j)(2i)} = \max \{v_j - v_i \mid V \in \mathbf{V}\}$$

- if $\exists R \in \mathbf{R}$ such that $r_i > -r_j$, we set $\mathbf{m}_{(2i-1)(2j)} = +\infty$,
  otherwise, we set $\mathbf{m}_{(2i-1)(2j)} = \max \{-v_j - v_i \mid V \in \mathbf{V}\}$;
- if $\exists R \in \mathbf{R}$ such that $-r_i > r_j$, we set $\mathbf{m}_{(2i)(2j-1)} = +\infty$,
  otherwise, we set $\mathbf{m}_{(2i)(2j-1)} = \max \{v_j + v_i \mid V \in \mathbf{V}\}$;
- otherwise we set $\mathbf{m}_{ii} = 0$.

This results in a strongly closed DBM representing the smallest octagon enclosing the polyhedron, and it is computed in $\mathcal{O}(n^2 \times (|\mathbf{R}| + |\mathbf{V}|))$ time. We denote this octagon by $\mathbf{m} \stackrel{\text{def}}{=} \mathit{Oct}(P)$. Note that whether the *Oct* operation stands for a conversion from intervals or from polyhedra will always be obvious by the context.

The case $\mathbb{I} = \mathbb{Z}$ is a little more subtle. Indeed, the polyhedron domain in $\mathbb{Z}$ generally uses the same representation—and algorithms—as rational polyhedra, but with an altered semantics. Only the points with integer coordinates inside the rational polyhedron are considered. Formally, the concretization $\gamma^{Poly}(P)$ of $P$ is replaced with $\gamma^{Poly}(P) \cap \mathbb{Z}^n$. With respect to this semantics, the classical polyhedron operators remain sound, but lose some precision (in particular, they are no longer best or exact abstractions). The above conversion algorithm would generate a DBM with non-integer elements when $\mathbb{I} = \mathbb{Z}$. We argue that it is safe to simply use this algorithm and round each $\mathbf{m}_{ij}$ to $\lfloor \mathbf{m}_{ij} \rfloor$. We may miss points in $\gamma^{Poly}(P)$ but no point in $\gamma^{Poly}(P) \cap \mathbb{Z}^n$. Consider, for instance, the one-dimensional polyhedron $P$ such that $\gamma^{Poly}(P) = \{(0.25)\}$, then the constructed DBM will be:

$$\mathbf{m} = \begin{array}{c|cc} & V_1' & V_2' \\ \hline V_1' & 0 & \lfloor -0.5 \rfloor \\ V_2' & \lfloor 0.5 \rfloor & 0 \end{array} = \begin{array}{c|cc} & V_1' & V_2' \\ \hline V_1' & 0 & -1 \\ V_2' & 0 & 0 \end{array}$$

and $\mathbf{m} \neq \mathbf{m}^\bullet = \bot^{\mathsf{DBM}}$. The conversion has discovered that $\gamma^{Poly}(P) \cap \mathbb{Z}^n = \emptyset$. Note that the rounded matrix is not generally strongly closed. Our algorithm is not complete as it may not always return the smallest octagon that encompasses $\gamma^{Poly}(P) \cap \mathbb{Z}^n$.

### 4.4. Abstract assignment

Given a numerical expression *expr*, the effect of an assignment $X \leftarrow expr$ on a set of environments $R$ is given by the following concrete function:

$$\{\!\!\{ X \leftarrow expr \}\!\!\}(R) \stackrel{\text{def}}{=} \{\rho[X \mapsto v] \mid \rho \in R,\ v \in [\![ expr ]\!](\rho)\}$$

where $[\![ expr ]\!](\rho) \in \mathcal{P}(\mathbb{I})$ is the set of possible values taken by the expression *expr* in the environment $\rho$. Our numerical expressions include, but are not limited to, constants $c \in \mathbb{I}$, variables $V \in \mathcal{V}$, and unary and binary arithmetic operators $+$, $-$, and $\times$. A convenient trick is to enhance constants $c$ into constant intervals $[a, b]$, where $a \in \mathbb{I} \cup \{-\infty\}$ and $b \in \mathbb{I} \cup \{+\infty\}$. Each time the interval is evaluated, a new value within the specified bounds is randomly chosen. This allows modeling non-deterministic behaviors, such as inputs from physical sensors with a known limited range. It also allows feeding the octagon domain with simplified expressions where complex, non-linear parts have been abstracted away into intervals. Finally, it is quite useful to account for rounding errors appearing when modeling floating-point expressions into real expressions, as performed in [45, 46]. All three techniques are used in the ASTRÉE static analyzer [3, 9]. A class of expressions that appears frequently is that of linear expressions with interval constant coefficients: $[a_0, b_0] + \sum_k [a_k, b_k] \times V_k$, where $a_i \in \mathbb{I} \cup \{-\infty\}$ and $b_i \in \mathbb{I} \cup \{+\infty\}$, so-called *interval linear forms*, for which we will provide specifically tailored transfer functions. The concrete evaluation $[\![ expr ]\!]$ of a numerical expression *expr* is defined by structural induction in Fig. 14.

Note that we suppose that our programs manipulate perfect integer, rational, or real numbers. We do not take into account the semantics of machine-integers—that can overflow or wrap-around—nor that of floating-point numbers—where each operation induces some rounding error. As explained in our PhD [46], these problems can be treated separately from the design of an abstract domain on perfect numbers.

*Simple and exact abstractions.* Only a few simple assignment forms have an exact abstraction in the octagon domain: $X \leftarrow [a, b]$ and $X \leftarrow \pm Y + [a, b]$. Their abstractions are defined in Fig. 15. Assignments $V_{j_0} \leftarrow V_{j_0} + [a, b]$ and $V_{j_0} \leftarrow -V_{j_0} + [a, b]$ do not require strongly closed matrix arguments but preserve the strong closure. They are said to be *invertible* because there exists backward assignments with the exact same semantical effect, as we will see in Section 4.6. Other, non-invertible assignments in Fig. 15, such as $V_{j_0} \leftarrow \pm V_{i_0} + [a, b]$ when $i_0 \neq j_0$, require a strong closure argument due to the embedded forget operator. The result is not strongly closed, but can be strongly closed by merely performing an incremental strong closure with respect to the assigned variable $V_{j_0}$: $Inc^{\bullet}_{j_0}$. Thus, a transfer function that keeps matrices in strongly closed form can be computed in quadratic time, in the worst case.

$$[\![ expr ]\!] : (\mathcal{V} \to \mathbb{I}) \to \mathcal{P}(\mathbb{I})$$

$$\begin{array}{lcll}
[\![ X ]\!](\rho) & \stackrel{\text{def}}{=} & \{\ \rho(X)\ \} & \\
[\![ [a, b] ]\!](\rho) & \stackrel{\text{def}}{=} & \{\ x \in \mathbb{I} \mid a \leq x \leq b\ \} & \\
[\![ -e ]\!](\rho) & \stackrel{\text{def}}{=} & \{\ -x \mid x \in [\![ e ]\!](\rho)\ \} & \\
[\![ e_1 \cdot e_2 ]\!](\rho) & \stackrel{\text{def}}{=} & \{\ x \cdot y \mid x \in [\![ e_1 ]\!](\rho),\ y \in [\![ e_2 ]\!](\rho)\ \} & \cdot \in \{+, -, \times\}
\end{array}$$

**Fig. 14** Semantics of numerical expressions $[\![ expr ]\!]$

- $(\{\!|\, V_{j_0} \leftarrow [a,b] \,|\!\}^{Oct}_{exact}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=}$

$$\begin{cases} -2a & \text{if } i = 2j_0 - 1,\ j = 2j_0 \\ 2b & \text{if } i = 2j_0,\ j = 2j_0 - 1 \\ (\{\!|\, V_{j_0} \leftarrow ? \,|\!\}^{Oct}(\mathbf{m}^\bullet))_{ij} & \text{otherwise} \end{cases}$$

- $(\{\!|\, V_{j_0} \leftarrow V_{j_0} + [a,b] \,|\!\}^{Oct}_{exact}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=}$

$$\begin{cases} \mathbf{m}_{ij} - a & \text{if } i = 2j_0 - 1,\ j \neq 2j_0 - 1, 2j_0 \\ & \text{or } j = 2j_0,\ i \neq 2j_0 - 1, 2j_0 \\ \mathbf{m}_{ij} + b & \text{if } i \neq 2j_0 - 1, 2j_0,\ j = 2j_0 - 1 \\ & \text{or } j \neq 2j_0 - 1, 2j_0,\ i = 2j_0 \\ \mathbf{m}_{ij} - 2a & \text{if } i = 2j_0 - 1,\ j = 2j_0 \\ \mathbf{m}_{ij} + 2b & \text{if } i = 2j_0,\ j = 2j_0 - 1 \\ \mathbf{m}_{ij} & \text{otherwise} \end{cases}$$

- $(\{\!|\, V_{j_0} \leftarrow V_{i_0} + [a,b] \,|\!\}^{Oct}_{exact}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=}$

$$\begin{cases} -a & \text{if } i = 2j_0 - 1,\ j = 2i_0 - 1 \\ & \text{or } i = 2i_0,\ j = 2j_0 \\ b & \text{if } i = 2i_0 - 1,\ j = 2j_0 - 1 \\ & \text{or } i = 2j_0,\ j = 2i_0 \\ (\{\!|\, V_{j_0} \leftarrow ? \,|\!\}^{Oct}(\mathbf{m}^\bullet))_{ij} & \text{otherwise} \end{cases}$$

- $(\{\!|\, V_{j_0} \leftarrow -V_{j_0} \,|\!\}^{Oct}_{exact}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=}$

$$\begin{cases} \mathbf{m}_{\bar{i}j} & \text{if } i \in \{2j_0 - 1, 2j_0\} \text{ and } j \notin \{2j_0 - 1, 2j_0\} \\ \mathbf{m}_{i\bar{j}} & \text{if } i \notin \{2j_0 - 1, 2j_0\} \text{ and } j \in \{2j_0 - 1, 2j_0\} \\ \mathbf{m}_{\bar{i}\bar{j}} & \text{if } i \in \{2j_0 - 1, 2j_0\} \text{ and } j \in \{2j_0 - 1, 2j_0\} \\ \mathbf{m}_{ij} & \text{if } i \notin \{2j_0 - 1, 2j_0\} \text{ and } j \notin \{2j_0 - 1, 2j_0\} \end{cases}$$

- $\{\!|\, V_{j_0} \leftarrow -V_{i_0} \,|\!\}^{Oct}_{exact} \stackrel{\text{def}}{=} \{\!|\, V_{j_0} \leftarrow -V_{j_0} \,|\!\}^{Oct}_{exact} \circ \{\!|\, V_{j_0} \leftarrow V_{i_0} \,|\!\}^{Oct}_{exact}$
- $\{\!|\, V_{j_0} \leftarrow -V_{j_0} + [a,b] \,|\!\}^{Oct}_{exact} \stackrel{\text{def}}{=}$

$$\{\!|\, V_{j_0} \leftarrow V_{j_0} + [a,b] \,|\!\}^{Oct}_{exact} \circ \{\!|\, V_{j_0} \leftarrow -V_{j_0} \,|\!\}^{Oct}_{exact}$$

- $\{\!|\, V_{j_0} \leftarrow -V_{i_0} + [a,b] \,|\!\}^{Oct}_{exact} \stackrel{\text{def}}{=}$

$$\{\!|\, V_{j_0} \leftarrow V_{j_0} + [a,b] \,|\!\}^{Oct}_{exact} \circ \{\!|\, V_{j_0} \leftarrow -V_{i_0} \,|\!\}^{Oct}_{exact}$$

**Fig. 15** Exact abstract assignments. We suppose that $i_0 \neq j_0$

In order to deal with assignments that cannot be exactly modeled in the octagon domain, we propose several definitions, in increasing order of precision and cost. The most precise versions only work for limited subsets of assigned expressions.

*Interval-based abstraction.* A coarse method is to perform the assignment $V_i \leftarrow expr$ as in the interval domain. We first extract an interval abstract environment from the octagon. Then, we evaluate *expr* using interval arithmetics. Finally, we feed the obtained interval $[a, b]$ to the exact transfer function for $X \leftarrow [a, b]$ presented above. This can be formalized as:

$$\{\!|\, V_i \leftarrow expr \,|\!\}^{Oct}_{nonrel}(\mathbf{m}) \stackrel{\text{def}}{=} \{\!|\, V_i \leftarrow (\llbracket\, expr \,\rrbracket^{Int}(Int(\mathbf{m}))) \,|\!\}^{Oct}_{exact}(\mathbf{m})$$

where $[\![ expr ]\!]^{Int}(X^\sharp)$ denotes the evaluation of the expression *expr* in the interval abstract domain, on the interval abstract environment $X^\sharp$, as derived from regular interval arithmetics [48].

The low precision of this transfer function stems from two facts. Firstly, we do not infer any relational information of the form $\pm V_i \pm V_j$. Secondly, we do not use the existing relational information in $\mathbf{m}$ when computing the bounds of *expr*.

*Deriving new relational constraints.* Our idea here is to solve the first cause of precision loss in the interval-based assignment, while not handling the second one. We still use interval arithmetics to compute bounds of expressions using only the information available in $Int(\mathbf{m})$. However, we compute the bounds of $\pm expr \pm V_j$ for all $i \neq j$ to infer constraints of the form $\pm V_i \pm V_j$. For instance, in the assignment $X \leftarrow Y + Z$, we would infer relations such as $\min(Z) \leq X - Y \leq \max(Z)$ and $\min(Y) \leq X - Z \leq \max(Y)$. In order to obtain that much precision, it is important to simplify formally each $\pm expr \pm V_j$ before evaluating it using interval arithmetics. In our example, $(Y + Z) - Y$, whose upper bound evaluates to $\max(Y) + \max(Z) - \min(Y)$, has been replaced with $Z$, that has a tighter upper bound whenever $\min(Y) \neq \max(Y)$. As another example, consider the assignment $X \leftarrow 2Y$. Then, we can infer the relation $X - Y \leq \max(Y)$ instead of $X - Y \leq 2\max(Y) - \min(Y)$ because the expression $2Y - Y$ has been simplified into $Y$ before being evaluated using interval arithmetics.

We now propose a full formal definition of this idea, including the simplification step, but only in the simpler yet useful case of the assignment of interval linear forms. The transfer function is shown in Fig. 16, denoting $[\![ expr ]\!]^{Int}(Int(\mathbf{m}))$ by $Int(expr)$ for the sake of conciseness. The interval linear form operators $\boxplus$ and $\boxminus$ used in Fig. 16 are defined by respectively adding and subtracting the interval coefficients corresponding to the same variable. These operators are formally presented in Fig. 17. They actually perform the required expression simplification by allowing several occurrences of the same variable to cancel one another.

$$
(\{\!|\, V_{j_0} \leftarrow expr \,|\!\}^{Oct}_{rel}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=}
$$

$$
\begin{cases}
2\max\left(Int(expr)\right) & \text{if } i = 2j_0 \text{ and } j = 2j_0 - 1 \\
-2\min\left(Int(expr)\right) & \text{if } i = 2j_0 - 1 \text{ and } j = 2j_0 \\
\max\left(Int(expr \boxminus V_{i_0})\right) & \text{if } i = 2i_0 - 1,\ j = 2j_0 - 1,\ i_0 \neq j_0 \\
 & \text{or } i = 2j_0,\ j = 2i_0,\ i_0 \neq j_0 \\
\max\left(Int(expr \boxplus V_{i_0})\right) & \text{if } i = 2i_0,\ j = 2j_0 - 1,\ i_0 \neq j_0 \\
 & \text{or } i = 2j_0,\ j = 2i_0 - 1,\ i_0 \neq j_0 \\
\max\left(Int(V_{i_0} \boxminus expr)\right) & \text{if } i = 2j_0 - 1,\ j = 2i_0 - 1,\ i_0 \neq j_0 \\
 & \text{or } i = 2i_0,\ j = 2j_0,\ i_0 \neq j_0 \\
\max\left(Int(\boxminus expr \boxminus V_{i_0})\right) & \text{if } i = 2i_0 - 1,\ j = 2j_0,\ i_0 \neq j_0 \\
 & \text{or } i = 2j_0 - 1,\ j = 2i_0,\ i_0 \neq j_0 \\
\mathbf{m}^\bullet_{ij} & \text{otherwise}
\end{cases}
$$

**Fig. 16** Abstract assignment of interval linear forms. $\boxplus$ and $\boxminus$ are defined in Fig. 17, while $Int(expr)$ is the evaluation of *expr* using interval arithmetics

$$([a_0, b_0] + \sum_k ([a_k, b_k] \times V_k)) \boxplus ([a'_0, b'_0] + \sum_k ([a'_k, b'_k] \times V_k)) \stackrel{\text{def}}{=} ([a_0 + a'_0, b_0 + b'_0] + \sum_k ([a_k + a'_k, b_k + b'_k] \times V_k))$$

$$\boxminus ([a_0, b_0] + \sum_k ([a_k, b_k] \times V_k)) \stackrel{\text{def}}{=} ([-b_0, -a_0] + \sum_k ([-b_k, -a_k] \times V_k))$$

**Fig. 17** Addition and opposite of interval linear forms

This assignment definition is more precise than the interval-based one at the cost of more evaluations of interval arithmetic expressions. It is not the best abstraction as we still do not use the relational information in the octagon when computing the bounds for each $\pm expr \pm V_j$.

For the sake of efficiency, a practical implementation would avoid evaluating all these $4n$ very similar interval linear forms. It is possible to compute once and for all both bounds of *expr*, and then derive bounds for $\pm expr \pm V_j$ by removing the contribution of $V_j$. Suppose, for instance, that $expr \stackrel{\text{def}}{=} [a_0, b_0] + \sum_j [a_j, b_j] \times V_j$, each variable $V_j$ has range $[x_j, y_j]$, and *expr* evaluates in the interval domain to $[x, y]$. We suppose, moreover, that $\forall j,\ x_j \neq -\infty,\ y_j \neq +\infty$ and $x \neq -\infty,\ y \neq +\infty$. Then, for every $j \neq i$ such that $a_j \geq 1$, the upper bound for $V_i - V_j$ after the assignment $V_i \leftarrow expr$ is exactly $y - y_j$ and the lower bound for $V_i + V_j$ is exactly $x + x_j$. Occurrences of $+\infty$ and $-\infty$ in bounds add some more complexity. For instance, if $y = +\infty$, then we cannot find a finite upper bound for $V_i$, but we may still be able to compute a finite upper bound for some $\pm V_i \pm V_j$. As there are many similar cases to consider, we chose not to present this optimization fully formally here. It suffices to say that a careful implementation of this assignment operator leads to a cost similar to that of the plain interval-based assignment.

*Polyhedron-based best abstraction.* One can consider using the polyhedron abstract domain temporarily to perform the assignment transfer function, and convert the result back into an octagon, as follows:

$$\{\!|\, V_i \leftarrow expr \,|\!\}^{Oct}_{poly}(\mathbf{m}) \stackrel{\text{def}}{=} (Oct \circ \{\!|\, V_i \leftarrow expr \,|\!\}^{Poly} \circ Poly)(\mathbf{m}).$$

The use of this definition is limited because the polyhedron domain can only deal with assignments of linear expressions. The definition of $\{\!|\, V_i \leftarrow expr \,|\!\}^{Poly}$ can be found, for instance, in [22]. Whenever $\mathbb{I} \neq \mathbb{Z}$, the conversion to a polyhedron and the polyhedron assignment are exact while the conversion back to an octagon is a best abstraction, and hence, we obtain the *best* abstract assignment in the octagon domain.

The high precision attained by $\{\!|\, V_i \leftarrow expr \,|\!\}^{Oct}_{poly}$ calls for a great cost. Because of the way our conversion operators work, the assignment transfer function on the polyhedron domain is fed with a constraint representation, while we require its output in a frame representation. This means that at least one representation conversion will occur. This incurs an exponential cost at worse—consider, for instance, translating the box $\forall i,\ V_i \in [0, 1]$ by the assignment $V_1 \leftarrow V_1 + 1$; it requires the computation of a frame representation containing $2^n$ vertices.

*Which operator to choose.* Figure 18 presents a comparison of our three abstract assignments on a "complex" assignment example: $X \leftarrow Y - Z$. We have closed the three results and presented only the constraints involving the variable $X$ to allow an easier comparison.

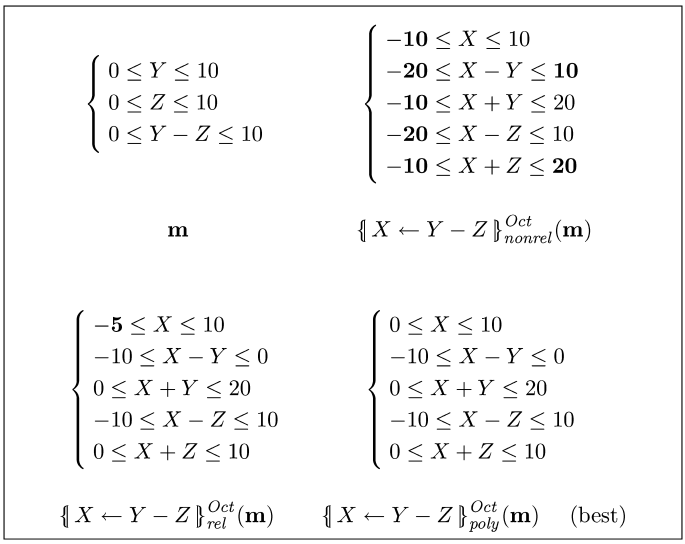


**Fig. 18** Constraints on variable $X$ derived after the assignment $X \leftarrow Y - Z$ using three different assignment transfer functions. Non-optimal bounds are shown in boldface

As the assignment $X \leftarrow Y - Z$ enforces invariants such as $X - Y + Z = 0$, involving three variables, its effect on an octagon cannot always be exactly represented. The polyhedron-based abstract assignment $\{\!\!\{ X \leftarrow Y - Z \}\!\!\}^{Oct}_{poly}$ gives the best result possible. Arguably, $\{\!\!\{ X \leftarrow Y - Z \}\!\!\}^{Oct}_{rel}$ is less precise: it is not able to use the constraint $0 \le Y - Z$ to infer the information $0 \le X$. Yet, it is much more precise than the interval-based abstract assignment $\{\!\!\{ X \leftarrow Y - Z \}\!\!\}^{Oct}_{nonrel}$. In Section 5.4, we will see an example where this extra precision is necessary and sufficient to perform an accurate analysis. If this precision is not sufficient, it is always possible to design other, more specialised, abstract transfer functions, adapted to selected expression forms and some precision versus cost trade-off.

In our implementation, we chose to use exact assignments $\{\!\!\{ \cdot \}\!\!\}^{Oct}_{exact}$ when possible, $\{\!\!\{ \cdot \}\!\!\}^{Oct}_{rel}$ when assigning an interval linear form, and $\{\!\!\{ \cdot \}\!\!\}^{Oct}_{nonrel}$ as a last resort. As the cost of the polyhedron-based assignment somewhat nullifies the gain, in time cost and ease of implementation, obtained by choosing the octagon domain instead of the polyhedron domain, we do not use it in actual static analyses. However, it is useful to perform regression tests and experimental studies of precision losses incurred when using non-optimal abstractions.

### 4.5. Abstract test

Given a boolean expression *test*, the effect of a test on a set of environments $R$ is to keep only the environments that can satisfy the given expression. It is given by the following concrete transfer function:

$$\{\!\!\{ \mathit{test} \ ? \}\!\!\}(R) \stackrel{\text{def}}{=} \{\rho \mid \rho \in R,\ \mathbf{T} \in [\![ \mathit{test} ]\!](\rho)\}$$

where $[\![ \mathit{test} ]\!](\rho) \in \mathcal{P}(\{\mathbf{T}, \mathbf{F}\})$ evaluates boolean expressions in a way similar to numerical expressions, except that it outputs a subset of booleans $\{\mathbf{T}, \mathbf{F}\}$, where $\mathbf{T}$ means "true" and $\mathbf{F}$ means "false". Tests include atomic comparisons of numerical expressions $\mathit{expr} \bowtie \mathit{expr}$,

$$
\begin{array}{l}
[\![\, test \,]\!] : (\mathcal{V} \to \mathbb{I}) \to \mathcal{P}(\{\mathbf{T}, \mathbf{F}\}) \\
\\
[\![\, e_1 \bowtie e_2 \,]\!](\rho) \quad \stackrel{\text{def}}{=} \quad \{\, \mathbf{T} \text{ if } \exists v_1 \in [\![\, e_1 \,]\!](\rho),\, v_2 \in [\![\, e_2 \,]\!](\rho),\; v_1 \bowtie v_2 \,\} \cup \\
\qquad\qquad\qquad\quad \{\, \mathbf{F} \text{ if } \exists v_1 \in [\![\, e_1 \,]\!](\rho),\, v_2 \in [\![\, e_2 \,]\!](\rho),\; v_1 \not\bowtie v_2 \,\} \\
\qquad\qquad\qquad\quad \bowtie \in \{=, \neq, <, >, \leq, \geq\} \\
\\
[\![\, t_1 \texttt{ and } t_2 \,]\!](\rho) \quad \stackrel{\text{def}}{=} \quad \{\, v_1 \wedge v_2 \mid v_1 \in [\![\, t_1 \,]\!](\rho),\, v_2 \in [\![\, t_2 \,]\!](\rho) \,\} \\
[\![\, t_1 \texttt{ or } t_2 \,]\!](\rho) \quad \stackrel{\text{def}}{=} \quad \{\, v_1 \vee v_2 \mid v_1 \in [\![\, t_1 \,]\!](\rho),\, v_2 \in [\![\, t_2 \,]\!](\rho) \,\} \\
[\![\, \texttt{not } t \,]\!](\rho) \quad \stackrel{\text{def}}{=} \quad \{\, \neg v \mid v \in [\![\, t \,]\!](\rho) \,\}
\end{array}
$$

**Fig. 19** Semantics of boolean expressions $[\![\, test \,]\!]$

where $\bowtie \in \{=, \neq, <, >, \leq, \geq\}$, linked using the `and`, `or`, and `not` boolean operators. The evaluation of a boolean expression $[\![\, test \,]\!]$ is defined in Fig. 19 using the boolean operators $\wedge$, $\vee$, and $\neg$ that correspond respectively to the logical "and", "or", and "not" operators on $\{\mathbf{T}, \mathbf{F}\}$.

*Preprocessing.* In order to simplify our definitions, we first show how the analysis of any test can be reduced to the analysis of atomic tests of the form $(e \leq 0\ ?)$. A first step is to transform our test into an equivalent test that does not use the `not` operator. This is done by "pushing" the `not` operators into the `and` and `or` operators using the De-Morgan laws, and reversing the comparison operators:

$$
\begin{array}{ll}
\texttt{not}(t_1 \texttt{ and } t_2) & \to (\texttt{not } t_1) \texttt{ or } (\texttt{not } t_2) \\
\texttt{not}(t_1 \texttt{ or } t_2) & \to (\texttt{not } t_1) \texttt{ and } (\texttt{not } t_2) \\
\texttt{not}(\texttt{not } t) & \to t \\
\texttt{not}(e_1 \leq e_2) & \to e_1 > e_2 \\
\text{etc.}\ldots &
\end{array}
$$

Then, provided that we have a transfer function for atomic tests $(e_1 \bowtie e_2\ ?)$, all `not`-free tests can be computed by structural induction using the already available $\cup^{Oct}$ and $\cap^{Oct}$ operators as follows:

$$
\begin{array}{l}
\{\!|\, (t_1 \texttt{ and } t_2)\ ?\ |\!\}^{Oct}(\mathbf{m}) \quad \stackrel{\text{def}}{=} \quad \{\!|\, t_1\ ?\ |\!\}^{Oct}(\mathbf{m}) \cap^{Oct} \{\!|\, t_2\ ?\ |\!\}^{Oct}(\mathbf{m}) \\
\{\!|\, (t_1 \texttt{ or } t_2)\ ?\ |\!\}^{Oct}(\mathbf{m}) \quad \stackrel{\text{def}}{=} \quad \{\!|\, t_1\ ?\ |\!\}^{Oct}(\mathbf{m}) \cup^{Oct} \{\!|\, t_2\ ?\ |\!\}^{Oct}(\mathbf{m})
\end{array}
$$

Finally, given an atomic test of the form $(e_1 \bowtie e_2\ ?)$, we group the expressions on the left side as follows: $(e_1 - e_2 \bowtie 0\ ?)$ and, whenever $\bowtie$ is not $\leq$, we do one of the following:

– If $\bowtie$ is $=$, our test will be abstracted as:

$$\{\!|\, e_1 - e_2 \leq 0\ ?\ |\!\}^{Oct}(\mathbf{m}) \;\cap^{Oct}\; \{\!|\, e_2 - e_1 \leq 0\ ?\ |\!\}^{Oct}(\mathbf{m}).$$

– If $\bowtie$ is $<$ and $\mathbb{I} = \mathbb{Z}$, then we can use the test:

$$\{\!|\, e_1 - e_2 + 1 \leq 0\ ?\ |\!\}^{Oct}(\mathbf{m}).$$

– If $\bowtie$ is $<$ and $\mathbb{I} \neq \mathbb{Z}$, as we have no way to represent strict inequalities exactly, we relax the test as a regular inequality:

$$\{\!|\, e_1 - e_2 \leq 0 \,?\, |\!\}^{Oct}(\mathbf{m}).$$

– The cases where $\bowtie \in \{>, \geq\}$ reduce to the cases $<$ and $\leq$ by exchanging $e_1$ and $e_2$.
– If $\bowtie$ is $\neq$ and $\mathbb{I} = \mathbb{Z}$, we combine two inequalities:

$$\{\!|\, e_1 - e_2 + 1 \leq 0 \,?\, |\!\}^{Oct}(\mathbf{m}) \;\cup^{Oct}\; \{\!|\, e_2 - e_1 + 1 \leq 0 \,?\, |\!\}^{Oct}(\mathbf{m}).$$

– If $\bowtie$ is $\neq$ and $\mathbb{I} \neq \mathbb{Z}$, there is generally no better abstraction than the identity, so, we choose:

$$\{\!|\, t \,?\, |\!\}^{Oct}(\mathbf{m}) \stackrel{\text{def}}{=} \mathbf{m}.$$

We now explain how to abstract atomic tests ($e \leq 0$ ?) using ideas similar to our assignment transfer functions.

*Simple and exact abstractions.* If the test has the shape of an octagonal constraint, it can be modeled exactly by simply *adding* the constraint to the DBM, as presented in Fig. 20. This test transfer function does not require a strongly closed argument. If, however, the argument is strongly closed, the result can be made strongly closed in quadratic time by applying the incremental strong closure with respect to any of the variables appearing in the test—even when the test involves two variables, the incremental strong closure needs to be performed with respect to only one of them.

*Interval-based abstraction.* When $e$ has an arbitrary form, it is always possible to fall back to the test transfer function in the interval domain:

$$\{\!|\, e \leq 0 \,?\, |\!\}^{Oct}_{nonrel}(\mathbf{m}) \stackrel{\text{def}}{=} (Oct \circ \{\!|\, e \leq 0 \,?\, |\!\}^{Int} \circ Int)(\mathbf{m}) \;\cap^{Oct}\; \mathbf{m}$$

where $\{\!|\, e \leq 0 \,?\, |\!\}^{Int}$ is the classical test abstraction in the interval domain—see [15] on how to derive test abstractions for generic non-relational domains. Because tests only *filter out* environments, it is safe to keep all the constraints of the argument DBM in the result, hence the intersection with $\mathbf{m}$ in our formula. This is quite convenient because $(Oct \circ \{\!|\, expr \leq 0 \,?\, |\!\}^{Int} \circ Int)(\mathbf{m})$ does not contain any relational constraint by itself. As a conclusion, we do not infer any new relational constraint but, at least, we keep all the ones that were valid before the test.

Due to the conversion to intervals, it is necessary for the argument $\mathbf{m}$ to be in strongly closed form to obtain maximum accuracy. In contrast to the interval-based assignment, one pass of incremental closure is not sufficient to obtain a strongly closed result as potentially all lines and columns may be modified. This gives a total cost which is, in the worst case, cubic in the number of variables, plus the cost of the transfer function in the interval domain—which is linear in the size of $e$—without much room for improvement.

*Deriving new relational constraints.* The interval-based abstraction is very poor. For instance, it is not able to prove that the constraint $X \leq Y$ holds after the test ($X - Y \leq 0$ ?), while our abstraction of simple tests $\{\!|\, e \leq 0 \,?\, |\!\}^{Oct}_{exact}$ can. In order to solve this problem, we propose an improvement of the interval-based abstraction able to derive new relational constraints.

- $(\{\!|\, V_{j_0} + [a,b] \leq 0\ ?\ |\!\}^{Oct}_{exact}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=}$

$$\begin{cases} \min(\mathbf{m}_{ij}, -2a) & \text{if } i = 2j_0,\ j = 2j_0 - 1 \\ \mathbf{m}_{ij} & \text{otherwise} \end{cases}$$

- $(\{\!|\, -V_{j_0} + [a,b] \leq 0\ ?\ |\!\}^{Oct}_{exact}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=}$

$$\begin{cases} \min(\mathbf{m}_{ij}, -2a) & \text{if } i = 2j_0 - 1,\ j = 2j_0 \\ \mathbf{m}_{ij} & \text{otherwise} \end{cases}$$

- $(\{\!|\, V_{j_0} - V_{i_0} + [a,b] \leq 0\ ?\ |\!\}^{Oct}_{exact}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=}$

$$\begin{cases} \min(\mathbf{m}_{ij}, -a) & \text{if } i = 2i_0 - 1,\ j = 2j_0 - 1 \\ & \text{or } i = 2j_0,\ j = 2i_0 \\ \mathbf{m}_{ij} & \text{otherwise} \end{cases}$$

- $(\{\!|\, V_{j_0} + V_{i_0} + [a,b] \leq 0\ ?\ |\!\}^{Oct}_{exact}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=}$

$$\begin{cases} \min(\mathbf{m}_{ij}, -a) & \text{if } i = 2i_0,\ j = 2j_0 - 1 \\ & \text{or } i = 2j_0,\ j = 2i_0 - 1 \\ \mathbf{m}_{ij} & \text{otherwise} \end{cases}$$

- $(\{\!|\, -V_{j_0} - V_{i_0} + [a,b] \leq 0\ ?\ |\!\}^{Oct}_{exact}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=}$

$$\begin{cases} \min(\mathbf{m}_{ij}, -a) & \text{if } i = 2i_0 - 1,\ j = 2j_0 \\ & \text{or } i = 2j_0 - 1,\ j = 2i_0 \\ \mathbf{m}_{ij} & \text{otherwise} \end{cases}$$

**Fig. 20** Exact abstract tests. We suppose that $i_0 \neq j_0$

However, unlike the simple test abstraction, it will work on all interval linear forms, and not only tests involving octagonal constraints. For instance, it will be able to prove that, after the test $(X \leq Y + Z\ ?)$, $X - Y$ is smaller than the upper bound of $Z$.

In order to derive some new relational constraints, we can remark that $e \leq 0$ implies $V_j - V_i \leq V_j - V_i - e$. Whenever $V_i$ or $V_j$ appears in $e$, there is a possibility that $V_j - V_i - e$ might be simplified and, once evaluated in the interval domain, gives a more precise upper bound for $V_j - V_i$ than the interval-based test. A formalization of this idea, when $e$ is an interval linear form, is presented in Fig. 21. The introduced operator, $\{\!|\, e \leq 0\ ?\ |\!\}^{Oct}_{rel}$, has a quadratic cost. It uses the $\boxplus$ and $\boxminus$ operators, introduced in Fig. 17, as well as the *Int*(*expr*) shortcut.

*Polyhedron-based best abstraction.* As for the assignment, whenever $e$ is a linear expression, the *best* abstraction can be computed at great cost by switching momentarily to the polyhedron abstract domain as follows:

$$\{\!|\, e \leq 0\ ?\ |\!\}^{Oct}_{poly}(\mathbf{m}) \stackrel{\text{def}}{=} (Oct \circ \{\!|\, e \leq 0\ ?\ |\!\}^{Poly} \circ Poly)(\mathbf{m})$$

where the classical test transfer function $\{\!|\, e \leq 0\ ?\ |\!\}^{Poly}$ for linear expressions in the polyhedron abstract domain is described, for instance, in [22]. This has an exponential worst-case

$$(\{\!\{ e \leq 0 \ ?\ \}\!\}^{Oct}_{rel}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=} \min(\mathbf{m}_{ij}, \mathbf{m}'_{ij})$$

where $\mathbf{m}'_{ij}$ is defined as:

$$\begin{cases} 2\max(\mathit{Int}(V_{j_0} \boxminus e)) & \text{if } \exists j_0,\ i = 2j_0,\ j = 2j_0 - 1 \\ -2\max(\mathit{Int}(\boxminus V_{j_0} \boxminus e)) & \text{if } \exists j_0,\ i = 2j_0 - 1,\ j = 2j_0 \\ \max(\mathit{Int}(V_{j_0} \boxminus V_{i_0} \boxminus e)) & \text{if } \exists i_0 \neq j_0,\ i = 2i_0 - 1,\ j = 2j_0 - 1 \\ & \text{or } \exists i_0 \neq j_0,\ i = 2j_0,\ j = 2i_0 \\ \max(\mathit{Int}(V_{j_0} \boxplus V_{i_0} \boxminus e)) & \text{if } \exists i_0 \neq j_0,\ i = 2i_0,\ j = 2j_0 - 1 \\ \max(\mathit{Int}(\boxminus V_{j_0} \boxminus V_{i_0} \boxminus e)) & \text{if } \exists i_0 \neq j_0,\ i = 2i_0 - 1,\ j = 2j_0 \\ \mathbf{m}_{ij} & \text{otherwise} \end{cases}$$

**Fig. 21** Abstract test of interval linear forms. $\boxplus$ and $\boxminus$ are defined in Fig. 17, while *Int*(*expr*) is the evaluation of *expr* using interval arithmetics

behavior. As for the polyhedron-based assignment, we will refrain from using it in practice. It is presented here merely for the sake of completeness.

It is useful to compare, on theoretical examples, the previous two methods with the best possible abstraction. Such an example is provided by Fig. 22. Note that $\{\!\{ e \leq 0 \ ?\ \}\!\}^{Oct}_{rel}$ is not guaranteed to be always at least as precise as the interval-based solution $\{\!\{ e \leq 0 \ ?\ \}\!\}^{Oct}_{nonrel}$—even though this is the case for the example of Fig. 22—so, it can be worth actually computing both and returning their intersection.

### 4.6. Abstract backward assignment

The backward assignment transfer function $\{\!\{ X \rightarrow \mathit{expr} \}\!\}(R)$ maps a set of environments $R$ to the set of environments that *can* lead to $R$ via an assignment $X \leftarrow \mathit{expr}$:

$$\{\!\{ X \rightarrow \mathit{expr} \}\!\}(R) \stackrel{\text{def}}{=} \{\rho \mid \exists v \in [\![ \mathit{expr} ]\!](\rho),\ \rho[X \mapsto v] \in R\}.$$

Viewing transfer functions as relations between environments, each backward assignment is really the *inverse relation* of the corresponding forward assignment. Backward assignment transfer functions are not generally used directly to define the concrete semantics of a program. Yet, they can be quite useful once abstracted. One classical application is to refine a static analysis by performing combined forward and backward passes, as proposed by Cousot and Cousot in [19, Section 6]. Another one is to backtrack from a user-specified program behavior to its origin, such as in Bourdoncle's *abstract debugging* [11].

We did not experiment with backward assignments yet. However, for the sake of completeness, we provide a few abstractions of backward assignments so that the octagon domain can be plugged as-is into existing or future backward-able analyzers. There are also plans to include these transfer functions within the ASTRÉE analyzer presented in Section 6.

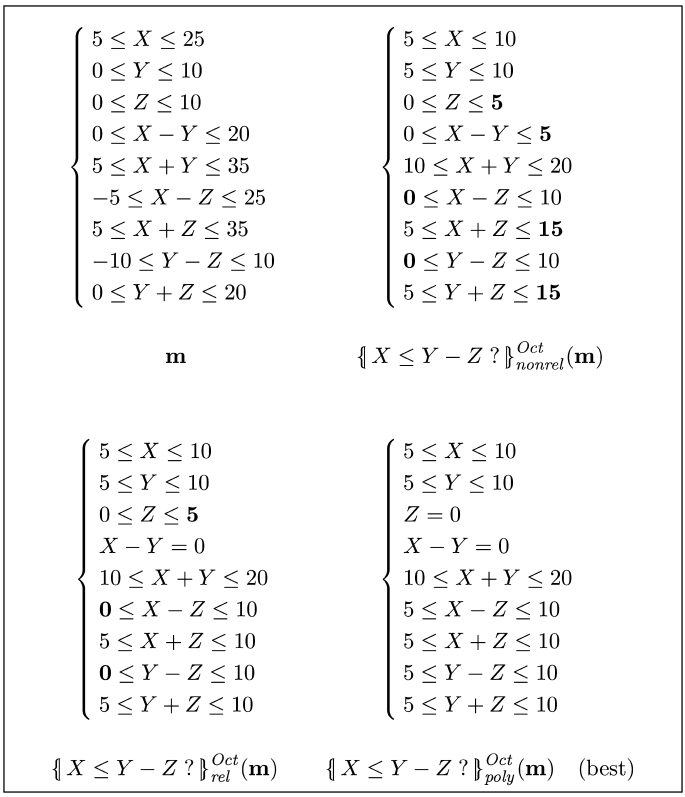


**Fig. 22** Constraints derived after the test ($X \le Y - Z$ ?) using three different abstract transfer functions. Non-optimal bounds are shown in boldface

*Simple and exact abstractions.* The backward assignments that can be modeled exactly are similar to the exact assignments $\{\!\!\{ \cdot \}\!\!\}^{Oct}_{exact}$. They are presented in Fig. 23. The backward assignments $V_{j_0} \to V_{j_0} + [a, b]$ and $V_{j_0} \to -V_{j_0} + [a, b]$ are invertible. They are semantically equivalent to, respectively, the forward assignments $V_{j_0} \leftarrow V_{j_0} - [a, b]$ and $V_{j_0} \leftarrow -V_{j_0} + [a, b]$. They do not require strongly closed arguments but preserve the strong closure. Other, non-invertible backward assignments, such as $V_{j_0} \to \pm V_{i_0} + [a, b]$ when $i_0 \neq j_0$, correspond to *substituting* $V_{j_0}$ with the assigned expression in all the constraints in $\mathbf{m}$. This generates new constraints that refine all the constraints related to $V_{i_0}$ but remove all information about $V_{j_0}$. Also, we may discover a trivially unsatisfiable constraint and return $\bot^{\mathsf{DBM}}$ directly. To obtain the maximum precision, the argument matrix must be strongly closed. The resulting matrix can then be strongly closed in quadratic time by invoking the incremental strong closure procedure $Inc^{\bullet}_{i_0, j_0}$. Indeed, all elements unrelated to $V_{i_0}$ or $V_{j_0}$ are left intact.

*Interval-based abstraction.* As for tests, we can use the backward assignment on the interval domain to discover interval information, but we need a way to recover some relational information as well. The idea is to keep in $\mathbf{m}$ all the constraints that are not invalidated by the backward assignment. Thus, we combine the interval transfer function together with the forget operator that abstracts non-deterministic backward assignments as well as non-deterministic

- $\{\!\!\{ V_{j_0} \to V_{j_0} + [a,b] \}\!\!\}^{Oct}_{exact} \stackrel{\text{def}}{=} \{\!\!\{ V_{j_0} \leftarrow V_{j_0} + [-b,-a] \}\!\!\}^{Oct}_{exact}$
- $\{\!\!\{ V_{j_0} \to -V_{j_0} + [a,b] \}\!\!\}^{Oct}_{exact} \stackrel{\text{def}}{=} \{\!\!\{ V_{j_0} \leftarrow -V_{j_0} + [a,b] \}\!\!\}^{Oct}_{exact}$
- if $\mathbf{m}^{\bullet}_{(2j_0)\,(2j_0-1)} \geq 2a$ and $\mathbf{m}^{\bullet}_{(2j_0-1)\,(2j_0)} \geq -2b$, then
  $(\{\!\!\{ V_{j_0} \to [a,b] \}\!\!\}^{Oct}_{exact}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=}$
  $$\begin{cases} \min(\mathbf{m}^{\bullet}_{ij},\ 2(\mathbf{m}^{\bullet}_{i\,(2j_0-1)} - a),\ 2(\mathbf{m}^{\bullet}_{i\,(2j_0)} + b)) & \text{if } i = \bar{j},\ i \notin \{2j_0-1, 2j_0\} \\ +\infty & \text{if } i \in \{2j_0-1, 2j_0\} \text{ or } j \in \{2j_0-1, 2j_0\} \\ \mathbf{m}^{\bullet}_{ij} & \text{otherwise} \end{cases}$$
  otherwise, $\{\!\!\{ V_{j_0} \to [a,b] \}\!\!\}^{Oct}_{exact}(\mathbf{m}) = \bot^{\mathsf{DBM}}$
- if $\mathbf{m}^{\bullet}_{(2i_0-1)\,(2j_0-1)} \geq a$ and $\mathbf{m}^{\bullet}_{(2j_0-1)\,(2i_0-1)} \geq -b$, then
  $(\{\!\!\{ V_{j_0} \to V_{i_0} + [a,b] \}\!\!\}^{Oct}_{exact}(\mathbf{m}))_{ij} \stackrel{\text{def}}{=}$
  $$\begin{cases} \min(\mathbf{m}^{\bullet}_{ij},\ \mathbf{m}^{\bullet}_{(2j_0-1)\,j} + b) \\ \quad \text{if } i = 2i_0 - 1,\ j \notin \{2i_0-1, 2j_0-1, 2i_0, 2j_0\} \\ \min(\mathbf{m}^{\bullet}_{ij},\ \mathbf{m}^{\bullet}_{i\,(2j_0)} + b) \\ \quad \text{if } j = 2i_0,\ i \notin \{2i_0-1, 2j_0-1, 2i_0, 2j_0\} \\ \min(\mathbf{m}^{\bullet}_{ij},\ \mathbf{m}^{\bullet}_{(2j_0)\,j} - a) \\ \quad \text{if } i = 2i_0,\ j \notin \{2i_0-1, 2j_0-1, 2i_0, 2j_0\} \\ \min(\mathbf{m}^{\bullet}_{ij},\ \mathbf{m}^{\bullet}_{i\,(2j_0-1)} - a) \\ \quad \text{if } j = 2i_0 - 1,\ i \notin \{2i_0-1, 2j_0-1, 2i_0, 2j_0\} \\ \min(\mathbf{m}^{\bullet}_{ij},\ \mathbf{m}^{\bullet}_{(2j_0)\,(2j_0-1)} - 2a) & \text{if } i = 2i_0,\ j = 2i_0 - 1 \\ \min(\mathbf{m}^{\bullet}_{ij},\ \mathbf{m}^{\bullet}_{(2j_0-1)\,(2j_0)} + 2b) & \text{if } i = 2i_0 - 1,\ j = 2i_0 \\ +\infty & \text{if } i \in \{2j_0-1, 2j_0\} \text{ or } j \in \{2j_0-1, 2j_0\} \\ \mathbf{m}^{\bullet}_{ij} & \text{otherwise} \end{cases}$$
  otherwise, $\{\!\!\{ V_{j_0} \to V_{i_0} + [a,b] \}\!\!\}^{Oct}_{exact}(\mathbf{m}) = \bot^{\mathsf{DBM}}$
- $\{\!\!\{ V_{j_0} \to -V_{i_0} + [a,b] \}\!\!\}^{Oct}_{exact} \stackrel{\text{def}}{=}$
  $\{\!\!\{ V_{j_0} \to V_{i_0} \}\!\!\}^{Oct}_{exact} \circ \{\!\!\{ V_{j_0} \to -V_{j_0} + [a,b] \}\!\!\}^{Oct}_{exact}$

**Fig. 23** Exact abstract backward assignments. We suppose that $i_0 \neq j_0$

forward assignments:

$$\{\!\!\{ V_i \to e \}\!\!\}^{Oct}_{nonrel}(\mathbf{m}) \stackrel{\text{def}}{=} (Oct \circ \{\!\!\{ V_i \to e \}\!\!\}^{Int} \circ Int)(\mathbf{m}) \cap^{Oct} \{\!\!\{ V_i \leftarrow ? \}\!\!\}^{Oct}(\mathbf{m}^{\bullet}).$$

$$
\begin{aligned}
&\{\!\!\{\, V_i \to e \,\}\!\!\}^{Oct}_{rel} \stackrel{\text{def}}{=} \\
&\quad (\{\!\!\{\, e \le \mathbf{m}_{(2i)\,(2i-1)}/2 \,?\, \}\!\!\}^{Oct}_{rel} \circ \{\!\!\{\, V_i \leftarrow ?\, \}\!\!\}^{Oct})(\mathbf{m}) \quad \cap^{Oct} \\
&\quad (\{\!\!\{\, e \ge -\mathbf{m}_{(2i-1)\,(2i)}/2 \,?\, \}\!\!\}^{Oct}_{rel} \circ \{\!\!\{\, V_i \leftarrow ?\, \}\!\!\}^{Oct})(\mathbf{m}) \quad \cap^{Oct} \\
&\quad \textstyle\bigcap^{Oct}_{i\neq j} (\{\!\!\{\, e \boxminus V_j \le \mathbf{m}_{(2j-1)\,(2i-1)} \,?\, \}\!\!\}^{Oct}_{rel} \circ \{\!\!\{\, V_i \leftarrow ?\, \}\!\!\}^{Oct})(\mathbf{m}) \quad \cap^{Oct} \\
&\quad \textstyle\bigcap^{Oct}_{i\neq j} (\{\!\!\{\, e \boxplus V_j \le \mathbf{m}_{(2j)\,(2i-1)} \,?\, \}\!\!\}^{Oct}_{rel} \circ \{\!\!\{\, V_i \leftarrow ?\, \}\!\!\}^{Oct})(\mathbf{m}) \quad \cap^{Oct} \\
&\quad \textstyle\bigcap^{Oct}_{i\neq j} (\{\!\!\{\, \boxminus e \boxminus V_j \le \mathbf{m}_{(2j-1)\,(2i)} \,?\, \}\!\!\}^{Oct}_{rel} \circ \{\!\!\{\, V_i \leftarrow ?\, \}\!\!\}^{Oct})(\mathbf{m}) \quad \cap^{Oct} \\
&\quad \textstyle\bigcap^{Oct}_{i\neq j} (\{\!\!\{\, \boxminus e \boxplus V_j \le \mathbf{m}_{(2j)\,(2i)} \,?\, \}\!\!\}^{Oct}_{rel} \circ \{\!\!\{\, V_i \leftarrow ?\, \}\!\!\}^{Oct})(\mathbf{m})
\end{aligned}
$$

**Fig. 24** Abstract backward assignment of interval linear forms

*Deriving new relational constraints.* For assignments and tests involving interval linear forms, we were able to refine the interval-based transfer functions by inferring some new relational constraints. This idea can be adapted here. Given the backward assignment $V_i \to e$ on $\mathbf{m}$, we can derive, for each variable $V_j \neq V_i$, four interval linear constraints by substitution: $e - V_j \le \mathbf{m}_{(2j-1)\,(2i-1)}$, $e + V_j \le \mathbf{m}_{(2j)\,(2i-1)}$, $-e - V_j \le \mathbf{m}_{(2j-1)\,(2i)}$, and $-e + V_j \le \mathbf{m}_{(2j)\,(2i)}$. The resulting transfer function is denoted by $\{\!\!\{\, V_i \to e \,\}\!\!\}^{Oct}_{rel}$ and defined formally in Fig. 24. We start from a coarse abstraction of the backward assignment, $\{\!\!\{\, V_i \leftarrow ?\, \}\!\!\}^{Oct}$, and then add all these constraints using the test transfer function for interval linear forms, $\{\!\!\{\, expr \le 0 \,?\, \}\!\!\}^{Oct}_{rel}$, presented in the preceding section. Note that each of the $4n$ test transfer functions applied will derive a quadratic number of constraints, and so, we generate a cubic total number of constraints. Hence, our transfer function has an unavoidable cubic cost. Moreover, our operator requires a strongly closed argument for best precision and it does not preserve the closure.

*Polyhedron-based best abstraction.* As for the test and assignment transfer functions, whenever the expression $e$ is linear, a *best* abstraction can be computed with exponential worst-case cost using momentarily the polyhedron abstract domain as follows:

$$\{\!\!\{\, V_i \to e \,\}\!\!\}^{Oct}_{poly}(\mathbf{m}) \stackrel{\text{def}}{=} (Oct \circ \{\!\!\{\, V_i \to e \,\}\!\!\}^{Poly} \circ Poly)(\mathbf{m}).$$

*Invertible assignments.* An assignment is said to be invertible whenever there exists a backward assignment with the exact same semantical effect. We have already seen simple examples of invertible assignments, such as $V_i \leftarrow V_i + [a, b]$, which has the same concrete semantics as $V_i \to V_i - [a, b]$. It is possible to exploit this fact and abstract an assignment using an abstract backward assignment transfer function or, conversely, to abstract a backward assignment using one of our abstractions for forward assignments.

We will now consider more closely the case of backward assignments of interval linear forms: $V_i \to [a_0, b_0] + \sum_j [a_j, b_j] \times V_j$. It is easy to see that, if $a_i = b_i$ and $a_i \neq 0$, then this backward assignment is invertible and has the exact same effect as the forward assignment:

$$V_i \leftarrow (1/a_i) \times V_i \;-\; ([a_0, b_0]/a_i) \;-\; \sum_{j\neq i} ([a_j, b_j]/a_i) \times V_j.$$

If $a_i \neq b_i$ and $0 \notin [a_i, b_i]$, the backward assignment may not be invertible. Indeed the inverse relation of the concrete semantics associates to $V_i$ the set $\{(1/x) \times V_i - ([a_0, b_0]/x) - \sum_{j \neq i} ([a_j, b_j]/x) \times V_j \mid x \in [a_i, b_i]\}$, which may not be expressible as an interval linear form. Still, we can model it using the following forward assignment:

$$V_i \leftarrow (1/[a_i, b_i]) \times V_i \; - \; ([a_0, b_0]/[a_i, b_i]) \; - \; \sum_{j \neq i} ([a_j, b_j]/[a_i, b_i]) \times V_j$$

where the division of two intervals is defined as usual in interval arithmetics [48]. In this second case, the forward assignment may yield more behaviors than the original backward assignment because we have forgotten some relationships—namely, between the choices of values within the intervals $[a_i, b_i]$ when evaluating the right-hand side. Yet, this is a sound abstraction and we can use our linear cost abstract assignment for interval linear forms $\{\!|\, V_i \leftarrow e \,|\!\}^{Oct}_{rel}$ of Fig. 16 instead of the cubic cost abstract backward assignment for interval linear forms $\{\!|\, V_i \rightarrow e \,|\!\}^{Oct}_{rel}$ of Fig. 24. We gain much time. However, it is not clear which operator gives the most precise answers. Indeed, both operators perform some abstractions which are difficult to compare in general. If precision is a concern and not cost, one can perform both operations and return the intersection of the results.

### 4.7. Extrapolation operators

Due to loops and recursive functions, the control-flow graph of a program generally contains cycles, which leads to computing least-fixpoints in the concrete semantics. In order to over-approximate such concrete fixpoints, one idea is to compute fixpoints in the abstract domain. Alas, abstract fixpoints are in general not computable if the abstract domain has an infinite height, which is the case of the octagon domain. In order to effectively compute abstractions of concrete least-fixpoints, Cousot and Cousot propose, in [17], to design so-called *widening* and *narrowing* extrapolation operators.

*Increasing iterations.* We recall from [17] that a binary operator $\triangledown$ in an abstract domain $\mathcal{D}^\sharp$ that is partially ordered by $\sqsubseteq^\sharp$ is a widening if and only if:

1. $\forall X^\sharp, Y^\sharp \in \mathcal{D}^\sharp,\ (X^\sharp \triangledown Y^\sharp) \sqsupseteq^\sharp X^\sharp, Y^\sharp$, and
2. for every chain $(X_i^\sharp)_{i \in \mathbb{N}}$, the increasing chain $(Y_i^\sharp)_{i \in \mathbb{N}}$ defined by

$$\begin{cases} Y_0^\sharp \stackrel{\text{def}}{=} X_0^\sharp \\ Y_{i+1}^\sharp \stackrel{\text{def}}{=} Y_i^\sharp \triangledown X_{i+1}^\sharp \end{cases}$$

is stable after a finite time, i.e., $\exists n < \omega,\ Y_{n+1}^\sharp = Y_n^\sharp$.

Then, if $F^\sharp$ is a sound abstraction of the operator $F$, the sequence $X_0^\sharp \stackrel{\text{def}}{=} \bot^\sharp$, $X_{i+1}^\sharp \stackrel{\text{def}}{=} X_i^\sharp \triangledown F^\sharp(X_i^\sharp)$ reaches, in finite time, a stable iterate. This iterate is an abstract post-fixpoint, and hence, it is a sound abstraction of $F$'s least-fixpoint.

In order to design a widening for octagons, we use the same idea as for the standard widening in the interval [17] and the polyhedron domains [30]: we remove unstable constraints. The resulting standard octagon widening $\triangledown^{Oct}_{std}$ is defined point-wise on DBMs as follows:

$$\left(\mathbf{m} \triangledown^{Oct}_{std} \mathbf{n}\right)_{ij} \stackrel{\text{def}}{=} \begin{cases} \mathbf{m}_{ij} & \text{if } \mathbf{m}_{ij} \geq \mathbf{n}_{ij} \\ +\infty & \text{otherwise} \end{cases}$$

More generally, any widening on initial segments, that is, intervals of the form $[-\infty, a]$, $a \in \overline{\mathbb{I}}$, gives rise to a widening on octagons by point-wise extension. For instance, Cousot and Cousot propose in [21, Section 8] to improve the standard interval widening in order to infer sign information. If an interval not containing 0 is not stable, they first try to see if 0 is a stable bound instead of deciding it should be set to $\pm\infty$. A further generalisation, presented in [21] and widely used in [8], is to design a widening parametrized by a *finite* set $\mathbb{T} \subseteq \mathbb{I}$ of thresholds. Each bound is enlarged to the threshold immediately greater. We bail out to $\pm\infty$ only when we are out of thresholds. This adapts nicely and gives a family of octagon widenings with thresholds $\triangledown_{th}^{Oct}$ parameterized by $\mathbb{T}$ as follows:

$$\left(\mathbf{m} \triangledown_{th}^{Oct} \mathbf{n}\right)_{ij} \stackrel{\text{def}}{=} \begin{cases} \mathbf{m}_{ij} & \text{if } \mathbf{m}_{ij} \geq \mathbf{n}_{ij} \\ \min\{x \mid x \in \mathbb{T} \cup \{+\infty\},\ x \geq \mathbf{n}_{ij}\} & \text{otherwise when } i \neq \overline{j} \\ \min\{2x \mid x \in \mathbb{T} \cup \{+\infty\},\ 2x \geq \mathbf{n}_{ij}\} & \text{otherwise when } i = \overline{j} \end{cases}$$

Note that, when $i = \overline{j}$, we use the set of thresholds $2\mathbb{T}$ instead of $\mathbb{T}$. This is because such matrix positions correspond to upper bounds of constraints of the form $\pm 2V_{\lceil i/2 \rceil} \leq \mathbf{m}_{i\overline{j}}$. We can prove the following property:

**Theorem 17.** *$\triangledown_{std}^{Oct}$ and $\triangledown_{th}^{Oct}$ are indeed widenings.*

*Decreasing iterations.* Once an abstract post-fixpoint is found, it is often possible to refine it to get a smaller abstraction of the concrete least-fixpoint using decreasing iterations. We recall here the required properties of a *narrowing* operator $\triangle$ as proposed by Cousot and Cousot in [17] to compute, in finite time, limits of sound decreasing iterations:

1. $\forall X^\sharp, Y^\sharp \in \mathcal{D}^\sharp,\ (X^\sharp \sqcap^\sharp Y^\sharp) \sqsubseteq^\sharp (X^\sharp \triangle Y^\sharp) \sqsubseteq^\sharp X^\sharp$—where $\sqcap^\sharp$ is the greatest lower bound with respect to $\sqsubseteq^\sharp$—and
2. for every chain $(X_i^\sharp)_{i\in\mathbb{N}}$, the chain $(Y_i^\sharp)_{i\in\mathbb{N}}$ defined by:

$$\begin{cases} Y_0^\sharp \stackrel{\text{def}}{=} X_0^\sharp \\ Y_{i+1}^\sharp \stackrel{\text{def}}{=} Y_i^\sharp \triangle X_{i+1}^\sharp \end{cases}$$

   is ultimately stationary after a finite time, i.e., $\exists n < \omega,\ Y_{n+1}^\sharp = Y_n^\sharp$.

Then, the sequence $Y_0^\sharp \stackrel{\text{def}}{=} X^\sharp$, $Y_{i+1}^\sharp \stackrel{\text{def}}{=} Y_i^\sharp \triangle F^\sharp(Y_i^\sharp)$ converges in finite time towards an abstraction of $F$'s least-fixpoint smaller than $X^\sharp$, provided that $X^\sharp$ is itself an abstraction of $F$'s least-fixpoint.

As for the widening, any narrowing on initial segments gives rise to a narrowing on the octagon domain by point-wise extension. We present here a "standard" narrowing based on the standard interval narrowing. It only refines constraints involving $+\infty$:

$$\left(\mathbf{m} \triangle_{std}^{Oct} \mathbf{n}\right)_{ij} \stackrel{\text{def}}{=} \begin{cases} \mathbf{n}_{ij} & \text{if } \mathbf{m}_{ij} = +\infty \\ \mathbf{m}_{ij} & \text{otherwise} \end{cases}$$

We can prove the following property:

**Theorem 18.** $\triangle^{Oct}_{std}$ *is indeed a narrowing.*

*Widening limitation.* It is important to remark that, even though, by definition of widenings, the sequence $\mathbf{m}^{i+1} \stackrel{\text{def}}{=} \mathbf{m}^i \triangledown^{Oct} \mathbf{n}^{i+1}$ always converges in finite time, the sequence $\mathbf{m}^{i+1} \stackrel{\text{def}}{=} (\mathbf{m}^i)^\bullet \triangledown^{Oct} \mathbf{n}^{i+1}$ may not. Figure 25 gives an example of such a diverging sequence. It corresponds to searching for a loop invariant in the program of Fig. 26 (where the calls to `random()` denote non-deterministic loop exit and test conditions) using the standard widening $\triangledown^{Oct}_{std}$.

This behavior is unlike that of most operators and transfer functions we presented earlier. Indeed, other operators could be safely used on DBMs that are either strongly closed or not,

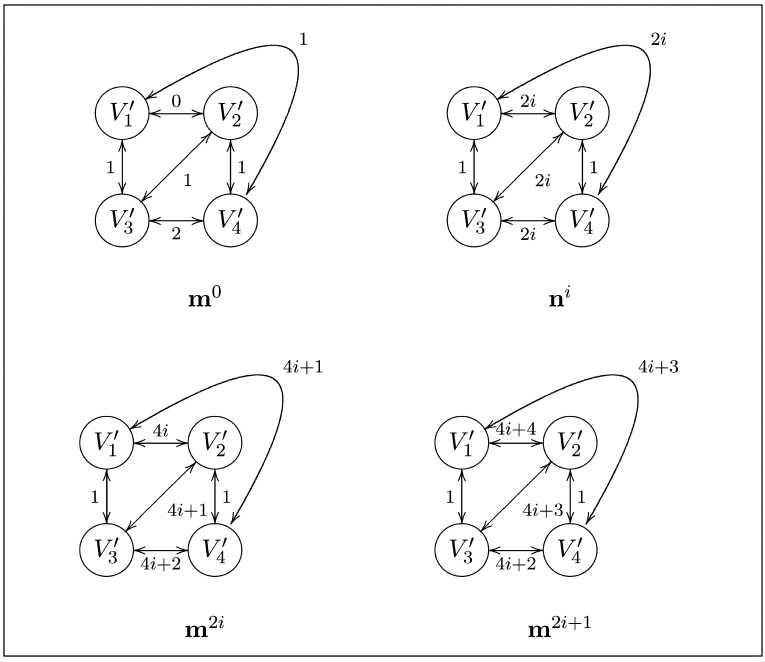


**Fig. 25** Example of infinite increasing chain defined by $\mathbf{m}^{i+1} \stackrel{\text{def}}{=} (\mathbf{m}^i)^\bullet \triangledown^{Oct}_{std} \mathbf{n}^{i+1}$

```
V1 := 0
V2 := [−1, 1]
while random() {
  if V1=V2 {
    if random() { V2 := V1+[−1, 1] }
    else        { V1 := V2+[−1, 1] }
  }
}
```

**Fig. 26** A program whose analysis leads to the infinite increasing chain of Fig. 25 if we wrongly close the widened iterates

generally at the cost of the some precision degradation. Closing the arguments of a widened sequence, on the other hand, jeopardizes the fixpoint computation. An intuitive explanation for this problem is that the proof of termination for the sequence $\mathbf{m}^{i+1} \stackrel{\text{def}}{=} \mathbf{m}^i \triangledown^{Oct} \mathbf{n}^{i+1}$ relies on replacing more and more matrix coefficients with $+\infty$, while the strong closure tends to reduce the number of $+\infty$ coefficients. More generally, it is know that widenings cannot be applied point-wise in *reduced products* of abstract domains. Indeed, our octagon domain can be seen as the reduced product of $2n^2$ abstract domains, each one of them focusing on an invariant of the form $\pm X \pm Y \leq c$. While many operators—such as the union and the widening—perform point-wise on these domains, the strong closure plays the role of a $2n^2-$way reduction between them.

Our solution to this problem is to always feed the result of the previous widening application $\mathbf{m}^i$ *unchanged* as *left* argument to the next widening application $\mathbf{m}^{i+1} \stackrel{\text{def}}{=} \mathbf{m}^i \triangledown^{Oct} \mathbf{n}^{i+1}$. Of course, the strong closure of the widened iterate $(\mathbf{m}^i)^\bullet$ can be safely used to compute the right argument of the next widening application, as in $\mathbf{n}^{i+1} \stackrel{\text{def}}{=} F((\mathbf{m}^i)^\bullet)$. Although this works perfectly well, having the correctness of an operator depend on which DBM $\mathbf{m}$ is used among all those representing the same concrete element $\gamma^{Oct}(\mathbf{m})$ is not fully satisfactory. As an illustration, the original polyhedron widening proposed in [22] also depended on the set of inequalities chosen to represent the arguments, but this was subsequently corrected by Halbwachs in [30]. The design of such a *semantical*, yet precise, widening for the octagon domain is a future work.[2]

Note that our narrowing does not suffer from the same limitation: any argument can be safely strongly closed. Moreover the right argument of the widening can be safely strongly closed; we thus compute $\mathbf{m}^{i+1} \stackrel{\text{def}}{=} \mathbf{m}^i \triangledown^{Oct} (\mathbf{n}^{i+1})^\bullet$. In contrast to the abstract union $\cup^{Oct}$, strongly closed arguments do not always give a more precise result as extrapolation operators are naturally non-monotonic—and the result of nested iterations with widening is even less predictable.

### 4.8. Summary

We have introduced quite a lot of abstract transfer functions with different precision versus cost trade-offs. Also, some functions require strongly closed arguments for best precision. Some functions preserve the strong closure, while others do not. For some functions, an incremental strong closure is sufficient to get the result in strongly closed form. The table in Fig. 27 tries to sum-up all these properties. Note that, in this table, the cost is given without that of the strong closure that may be required to get the expected precision or a strongly closed result. Likewise, the cost of the incremental strong closure that may be required to obtain a strongly closed result is omitted. Thus, depending on the case, you may have to add a cubic cost (for strong closure) or quadratic cost (for incremental strong closure).

Finally, we recall that some transfer functions are limited to certain expression forms:

- the *exact* assignments, tests, and backward assignments are limited to expressions involving at most two variables (including the left-hand variable for assignments and backward assignments) and unit coefficients;
- the *poly* assignments, tests, and backward assignments are limited to linear forms;
- the *rel* assignments and tests are limited to interval linear forms.

[2] See [5] for novel widening ideas proposed by Bagnara et al. during the writing of the present article.

| transfer function | cost | precision | arg. should be closed? | result is closed? |
|---|---|---|---|---|
| **set-theoretic** (Sect. 4.1) | | | | |
| $\cap^{Oct}$ | $n^2$ | exact | no | no |
| $\cup^{Oct}$ | $n^2$ | best | yes | yes |
| **assignment** (Sects. 4.2, 4.4) | | | | |
| $\{\!\| V \leftarrow ? \|\!\}^{Oct}$ | $n$ | exact | yes | yes |
| $\{\!\| V \leftarrow e \|\!\}^{Oct}_{exact}$ | $n$ | exact | may | via $Inc^{\bullet}$ |
| $\{\!\| V \leftarrow e \|\!\}^{Oct}_{nonrel}$ | $n$ | poor | yes | via $Inc^{\bullet}$ |
| $\{\!\| V \leftarrow e \|\!\}^{Oct}_{rel}$ | $n$ | medium | yes | via $Inc^{\bullet}$ |
| $\{\!\| V \leftarrow e \|\!\}^{Oct}_{poly}$ | $e^n$ | best | no | yes |
| **test** (Sect. 4.5) | | | | |
| $\{\!\| test\ ? \|\!\}^{Oct}_{exact}$ | 1 | exact | no | via $Inc^{\bullet}$ |
| $\{\!\| test\ ? \|\!\}^{Oct}_{nonrel}$ | $n^2$ | poor | yes | no |
| $\{\!\| test\ ? \|\!\}^{Oct}_{rel}$ | $n^2$ | medium | yes | no |
| $\{\!\| test\ ? \|\!\}^{Oct}_{poly}$ | $e^n$ | best | no | yes |
| **backward assignment** (Sect. 4.6) | | | | |
| $\{\!\| V \rightarrow e \|\!\}^{Oct}_{exact}$ | $n$ | exact | may | via $Inc^{\bullet}$ |
| $\{\!\| V \rightarrow e \|\!\}^{Oct}_{nonrel}$ | $n^2$ | poor | yes | no |
| $\{\!\| V \rightarrow e \|\!\}^{Oct}_{rel}$ | $n^3$ | medium | yes | no |
| $\{\!\| V \rightarrow e \|\!\}^{Oct}_{poly}$ | $e^n$ | best | no | yes |
| **extrapolation** (Sect. 4.7) | | | | |
| $\triangledown^{Oct}_{std}$ | $n^2$, $n^2$ | medium | NO | no |
| $\triangledown^{Oct}_{th}$ | $n^2$, $\|\mathbb{T}\| \times n^2$ | medium | NO | no |
| $\triangle^{Oct}_{std}$ | $n^2$, $n^2$ | medium | no | no |

**Fig. 27** Summary of our abstract transfer functions and their properties

Finally, for the extrapolation operators, two costs are given. The first one is the cost per operator application. The second one is an upper bound on the number of iterations before stabilisation—so-called the *maximal height* of ascending and descending chains. It is also recalled that widening arguments should not be strongly closed, as this disrupts the stabilisation of iterates.

## 5. Analysis examples

In this section, we provide various example analyses on program fragments to illustrate the usefulness of the octagon domain when analysing loops and numerical programs. These examples cannot be precisely analyzed with a non-relational abstract domain, such as the interval domain. Some of them require inferring bounds on variable sums, and hence, cannot be analyzed using the zone abstract domain we proposed in previous work [43]—the zone domain can only infer constraints of the form $X - Y \leq c$ and $\pm X \leq c$. In all but the last example, the octagon domain gives the expected, most precise, answer. The polyhedron domain gives strictly better results only in the last example, and the result obtained by the octagon domain in that case is still good.

### 5.1. Increasing loop counter

We first consider the following loop that iterates from 0 to $N$:

```
X := 0
N := [0, +∞]
while ① X < N {
  X := X + 1
} ②
```

We suppose that our analysis performs a standard widening at the program point ①, between the **while** keyword and the loop condition. This point is traversed when we first enter the loop and after each loop iteration, just before we test whether we should stay in the loop or exit. An interval analysis would only discover that $\mathtt{X} \in [0, +\infty]$ as the constraint $\mathtt{X} \geq 0$ is stable while the upper bound of X is not. The octagon domain will discover that the more precise, relational, constraint $\mathtt{X} \leq \mathtt{N}$ holds within the loop. Combined with the loop exit condition $\mathtt{X} \geq \mathtt{N}$, this allows proving that, at the end of the program ②, $\mathtt{X} = \mathtt{N}$. The polyhedron domain would find the very same invariants.

### 5.2. Decreasing loop counter

Consider now the following example where the loop counter I is decremented at each loop iteration while the index X is incremented:

```
I := 16
X := 1
while I > 0 {
  X := X + 1
  I := I − 1
}
```

Iterations with the standard widening in the octagon domain are able to prove that $\mathtt{X} + \mathtt{I} = 17$ is a loop invariant. However, as I's lower bound decreases at each iteration, the widening will only be able to infer that $\mathtt{I} \in [-\infty, 16]$. Decreasing iterations with the standard narrowing are required to prove that, within the loop, $\mathtt{I} \geq 0$. Alternatively, we can use a widening with thresholds instead of the standard widening, provided that $0 \in \mathbb{T}$. Combined with the loop exit condition $\mathtt{I} \leq 0$, this gives $\mathtt{I} = 0$, and so, $\mathtt{X} = 17$ at the end of the program.

As in the preceding example, the interval domain is not able to find a precise upper bound for X. The polyhedron domain would find the very same invariants as the octagon domain. It would also require the use of either a narrowing or a widening with thresholds.

### 5.3. Absolute value

Consider the following code that computes the absolute value Y of X before testing whether it is smaller than 69:

```
X := [−100, 100]
Y := X
if Y ≤ 0 {① Y := −Y ②} else {③}
④
if Y ≤ 69 {⑤}
```

Program points of interest have been numbered from ① to ⑤. In particular, at ⑤, we have $-69 \leq X \leq 69$, because the absolute value Y of X is assumed to be less than 69. The invariants computed in the octagon domain are:

① $-100 \leq X \leq 0 \ \wedge\ -100 \leq Y \leq 0 \ \wedge\ X - Y = 0 \ \wedge\ -200 \leq X + Y \leq 0$
② $-100 \leq X \leq 0 \ \wedge\ 0 \leq Y \leq 100 \ \wedge\ -200 \leq X - Y \leq 0 \ \wedge\ X + Y = 0$
③ $0 \leq X \leq 100 \ \wedge\ 0 \leq Y \leq 100 \ \wedge\ X - Y = 0 \ \wedge\ 0 \leq X + Y \leq 200$
④ $-100 \leq X \leq 100 \ \wedge\ 0 \leq Y \leq 100 \ \wedge\ -200 \leq X - Y \leq 0 \ \wedge\ 0 \leq X + Y \leq 200$
⑤ $-69 \leq X \leq 69 \ \wedge\ 0 \leq Y \leq 69 \ \wedge\ -138 \leq X - Y \leq 0 \ \wedge\ 0 \leq X + Y \leq 138$

Intuitively, one may think that the most precise bounds for X at ⑤ can only be discovered by an abstract domain able to represent the constraint $Y = |X|$. In fact, this intuition is false and the octagon domain, which cannot represent such a non-linear and non-convex constraint, finds the most precise bounds for X. The important point is that, at ④, we are able to infer the information $-Y \leq X \leq Y$ that will be combined by strong closure with the information $Y \leq 69$ at ⑤. This analysis works equally well if we modify the 100 and 69 constants. The same bounds can be found by the polyhedron domain, but not the interval domain or the zone abstract domain.

### 5.4. Rate limiter

Our last example is the following code implementing a *rate limiter*:

```
Y := 0
while random() {
 X := [−128, 128]
 D := [0, 16]
 S := Y
 ①
```

```
  R := X − S
  Y := X
  if R ≤ −D { ② Y:=S-D ③ } else
  if D ≤ R  { ④ Y:=S+D ⑤ }
}
```

At each loop iteration, a new value for the entry X is fetched within $[-128, 128]$ and a new maximum rate D is chosen in $[0, 16]$. The program then computes an output Y that tries to follow X but is forced to change slowly. The absolute difference between Y and its value in the preceding iteration is bounded by the current value of D. The variable S is used to store the value of Y from the previous iteration while R is a temporary variable used to avoid computing the difference $X - S$ twice.

The output Y is bounded by the range of X, that is, $Y \in [-128, 128]$. To prove this, suppose that $Y \in [-128, 128]$ at the start of a loop iteration. One of the three following cases may occur at the end of the same loop iteration:

- If $-D < R < D$, then $Y = X$.
- If $R \leq -D$, then $Y = S - D$. As $R = X - S$, we have $X - S \leq -D$, so, $S - D \geq X$. Thus, $X \leq Y \leq S$, so, $Y \in [-128, 128]$.
- If $R \geq D$, then $Y = S + D$. As $R = X - S$, we have $X - S \geq D$, so, $S + D \leq X$. Hence, $S \leq Y \leq X$, so, $Y \in [-128, 128]$.

*Interval analysis.* The interval analysis is not able to keep any relationship between R, X, and S. As a consequence, the tests $R \leq -D$ and $R \geq D$ do not refine the bounds for $S - D$ nor $S + D$. The analysis behaves exactly as if these tests were ignored, that is, as if Y was non-deterministically incremented or decremented by D at *each* loop iteration. An abstract semantics using the best interval transfer functions and exact fixpoint computations without widening would find that Y is unbounded, thus, no computable fixpoint abstraction can find finite bounds for Y.

*Octagonal analysis.* In order to find the most precise bounds for Y, that is $Y \in [-128, 128]$, one needs to exactly represent the constraint $R = X - S$, which is not possible in the octagon domain. Nevertheless, our non-optimal assignment transfer function for interval linear forms, $\{\!|\, V \leftarrow e \,|\!\}^{Oct}_{rel}$, is powerful enough to derive the constraint $R + S \in [-128, 128]$. Suppose that $Y \in [-M, M]$ at the beginning of the current abstract loop iteration. Then, at ①, we also have $S \in [-M, M]$ and the following computation occurs:

- At ②, the test implies $R + D \leq 0$, which implies $-R \geq 0$ and $S=(S + R) - R \geq -128$, so, $S \in [-128, M]$. At ③, $Y - S=-D \in [-16, 0]$, which gives $Y=(Y - S) + S \in [-144, M]$.
- At ④, the test implies $R - D \geq 0$, which implies $-R \leq 0$ and $S=(S + R) - R \leq 128$, so, $S \in [M, 128]$. At ⑤, $Y - S=D \in [0, 16]$, which gives $Y=(Y - S) + S \in [-M, 144]$.
- At the end of the current loop iteration, by union, we get $Y \in [-\max(M, 144), \max(M, 144)]$.

Thus, the iteration is stable if and only if $M \geq 144$. As a consequence, a static analysis using the widening with thresholds $\nabla^{Oct}_{th}$ on the octagonal domain will find as bound for Y the smallest threshold greater than 144. Even though this result is not optimal, we are still able to derive *finite bounds* for Y provided the widening we use has sufficiently many steps.

Finally, it is important to remark that, if we had used the less precise interval-based abstraction $\{\!|\, V \leftarrow e \,|\!\}^{Oct}_{nonrel}$ instead of $\{\!|\, V \leftarrow e \,|\!\}^{Oct}_{rel}$ for the assignments $\mathtt{R} := \mathtt{X} - \mathtt{S}$, $\mathtt{Y} := \mathtt{S} - \mathtt{D}$, and $\mathtt{Y} := \mathtt{S} + \mathtt{D}$, we would not have been able to find any bound at all. Also, the much more costly polyhedron-based abstraction $\{\!|\, V \leftarrow e \,|\!\}^{Oct}_{poly}$ would not have given any increase in precision with respect to $\{\!|\, V \leftarrow e \,|\!\}^{Oct}_{rel}$ on this example.

*Polyhedron analysis.* Unlike the three preceding examples, the polyhedron domain performs here strictly better than the octagon domain. Indeed, as it is able to manipulate constraints with three variables, it is able to prove the most precise invariant: $\mathtt{Y} \in [-128, 128]$. Moreover, the octagon domain required the use of a custom widening with a user-supplied threshold, while the polyhedron domain can find the most precise result using the standard widening, without any external help. If one is only interested in finding some bound for Y, and not finding the tightest one, the octagon domain is still sufficient.

## 6. Application to the ASTRÉE analyzer

The octagon abstract domain presented in this paper has been implemented as a freely available general-purpose library [41]. A simple academic analyzer using this library is included in the distribution and also available on-line [42]. More importantly, the library was incorporated into the ASTRÉE static analyzer [3, 8, 9] which is a joint work by the Abstract Interpretation teams at the École Normale Supérieure (ENS), in Paris, and the École Polytechnique, in Palaiseau.

### 6.1. Brief presentation of ASTRÉE

ASTRÉE is an efficient static analyzer, written in OCAML [49], focusing on the detection of run-time errors for programs written in a subset of the C programming language. Due to abstractions, the issued warnings may be either real bugs or spurious behaviors called "false alarms." The goal of ASTRÉE is to *prove* the absence of run-time errors meaning that, when analysing a correct program, the analyzer should issue very few or no alarm at all. However, we require ASTRÉE to be very precise only when the analyzed program belongs to a certain program family, that will be described bellow. Because we aim towards code *certification*, not bug searching, each alarm must be thoughtfully inspected manually. Thus, only up to a dozen alarms is acceptable, while a so-called *selectivity* of even 99% would require thousands of manual inspections which would be far too prohibitive. To achieve this goal, several abstract domains are used in combination during the analysis. They are written as modules using a common interface and can be easily plugged into the analyzer.

Currently, the family of programs considered by ASTRÉE is that of safety, critical, embedded, fly-by-wire software for Airbus planes [2]. The considered subset of the C programming language excludes recursion, union data-types, dynamic memory allocation, calls to the C library, and multi-threaded programs. However, it includes arrays, (possibly unbounded) loops, and floating-point computations. Run-time errors we check for include integer and floating-point divisions by zero, integer overflows, generation of floating-point $+\infty$, $-\infty$, or *NaN*, and out of bound array accesses. The considered program family has some features that make the analysis quite difficult:

– The considered programs are quite large: up to a few hundred thousand lines.
– There is a very large loop that runs for a long time ($3.6 \times 10^6$ iterations) and executes most of the code at each iteration (some parts get executed up to twelve times per iteration).
– There is a very large number of global variables (e.g., 14, 000 global variables in our 70, 000 lines example). They store the information for the current state and are read and written at least once per iteration.
– A large part of the program is devoted to floating-point computations (half the global variables are floating-point).
– Computation paths from input values with known bounds can be very long and may spread across many loop iterations, and hence, rounding-errors accumulate easily.

### 6.2. Integration of the octagon domain

Even though the invariants required to *express* the absence of run-time errors have an interval form, the interval domain is not sufficient to infer precise enough inductive invariants to derive tight enough bounds. Equivalently, the *proof* of absence of run-time error cannot be encoded using solely the non-relational interval abstraction. Thus, ASTRÉE makes use of relational domains. In particular, the octagon domain was key in the removal of hundreds of false alarms, some coming from code fragments semantically similar to those of Section 5.

We will now focus solely on the influence of the octagon abstract domain in ASTRÉE and refer the reader to the papers [9, 8, 27, 28, 38, 39] and the web-page [3] for more general informations about ASTRÉE and the other abstract domains it includes.

*Floating-point octagons.* In order to soundly abstract floating-point computations, we rely on the framework presented in [45, 46]. Floating-point expressions are first transformed into interval linear forms on reals by abstracting rounding errors as non-deterministic error intervals. Then, in order to achieve maximal efficiency, all the algorithms presented in the present paper have been implemented using floating-point numbers instead of arbitrary precision rationals. The soundness of our implementation is guaranteed by always rounding the manipulated upper bounds towards $+\infty$. Both steps result in some precision degradation but allow constructing an abstract domain that abstracts floating-point numbers using only floating-point computations. It is quite important to note that, currently, the polyhedron domain does not include abstract transfer functions to treat the interval linear forms that occur naturally when abstracting floating-point expressions. Also, it does not provide sound algorithms using floating-point numbers only. These facts, as well as the reduced algorithmic cost, advocated strongly in favor of the octagon domain for the relational analysis of real-life programs. It is interesting to note that we never encountered any false alarm that could only be removed by using more general linear invariants than octagonal ones. Parts of the program that were out of the reach of the octagon domain required the design of new abstract domains for non-linear or temporal invariants [27, 28], as well as the use of generic partitioning techniques.

*Widening.* In order to stabilise loop invariants in finite time, we use the widening with thresholds $\nabla_{th}^{Oct}$ in the octagon domain. All the invariants we encountered were in fact stable for bounds that are "large enough." More precisely, all abstract bounds stabilised to the smallest threshold larger than the concrete bound—plus an extra rounding error as the abstract computations are performed using floating-point arithmetics. Thus, the exact concrete bounds never needed to be known in advance nor inserted manually. Such an example is given by

the rate limiter of Section 5.4. As a consequence, the exact value of the widening thresholds is of no importance to prove that our variables are bounded and the thresholds do not need to be adapted from one program to another. However, as our programs are composed of many such computations in sequence, imprecision—that is, the difference between the stable abstract bound found and the actual concrete fixpoint—accumulates easily. Then, subsequent operations on stable but too large bounds may result in false alarms. This means that the set of thresholds should be sufficiently dense. It should not be much denser than required, however, as the number of thresholds directly affects the number of abstract iterations, and so, the analysis time. In ASTRÉE, we use, as set of thresholds, a simple piece-wise linear ramp with a few dozen values only. By trial-and-error, it was not difficult to find a ramp that is sufficiently precise for all the programs we analyze, and yet provides reasonable analysis times.

*Octagon packing.* Even though its cost is light compared to the polyhedron abstract domain, it would still be too costly to use the octagon domain to relate *all* live program variables in one large octagon, as there can be tens of thousands of them at any given program point. We decided, instead, to break down our variable set into *packs* of a few variables, each pack corresponding to variables that should be related together.

Given a fixed packing, an abstract environment then maps an octagon of the correct dimension to each pack. Transfer functions and operators are applied point-wise. Note that a variable may belong to several packs. Propagating information between the packs is possible by using common variables as pivot. It would enhance the precision. However, we found it easier, when extra precision was needed to remove false alarms, to refine the automatic packing strategy instead. The cost of the octagonal analysis depends on several parameters: the number of octagon packs, the size of octagon packs, but also the number of times the same variable appears in different octagons—this determines the number of octagons updated by each transfer functions application.

*Automatic packing technique.* Which variables to pack together can be specified manually in the analyzed program. After a few manual experiments, we developed a packing algorithm to automate this task for our considered program family. This algorithm traverses the code syntax tree and maps a pack to each syntactic C block, that is, code sequence bracketed within `{and}` but also bracketed and unbracketed if-then-else branches and loop bodies. In order to fill the pack for a given syntactic block with variables, we perform the following filtering steps:

1. We first gather all statements in that block, excluding the statements in its sub-blocks.
2. From these statements, we only keep simple C expressions. This includes assignments but not `if` and `while` statements.
3. From each such expression, we extract the variables it uses but ignore a variable if there is little chance that the expression behaves linearly with respect to this variable. More precisely, we do not scan the arguments of a bit-level C operator, a function call, an array lookup, or an "address-of" operator, but we scan recursively both arguments of the $+$, $-$, and $*$ arithmetic operators, as well as the `&&` and $\|$ logical operators and all comparison operators; we also scan the left argument of a `/` operator.
4. For each expression, if the set of extracted variables contains at least *two* variables, we add all extracted variables to the pack. If it contains only one variable, we do not add it. For instance, the assignment `X=Y+(Z&2)` will result in both `X` and `Y` being added to the

current pack, `Z` being ignored as argument to a bit-level operator. As another example, `X=3` does not contribute any variable to the pack.

In addition, Steps 3 and 4 are executed on expressions appearing in an if-then-else condition but the extracted variables are added to *both* the block enclosing the `if` statement and the blocks in the then and else branches. Variables are also extracted from each loop condition and added to both the block enclosing the loop and the loop body block. The effect of this filtering is to keep, for each assignment, only variables that have an opportunity to generate linear relational invariants. If we are to analyze the effect of a sequence of assignments and tests sharing common variables with the best possible precision, it is necessary to put *all* the variables of the involved expression in the *same* octagon pack as there is no information transfer between distinct packs. As packing all the extracted variables from all expressions in the same octagon would result in a huge octagon, we relate together only variables from expressions in the same syntactic block and from conditional expressions that relate the block to both its directly enclosing block and nested blocks. This strategy could be extended by considering the expressions in nested sub-blocks up to some nesting limit. This would result in larger packs—but less of them. In addition to variables extracted from expressions using Steps 3 and 4, we add to the octagon pack of loop bodies any variable that is either incremented or decremented, so that we are able to infer relationship between loop counters, as shown in Sections 5.1 and 5.2. It is quite important, for the considered family of programs, not to rely on variable *declaration* but on variable *usage* to define packing. Otherwise, this would result in all global variables being packed together in an octagon with thousands of variables.

We perform an optimization step before passing the packing information to the static analyzer. If the set of variables of a pack is included in the set of variables of a larger pack, then the smaller pack is discarded. We stress the fact that, even though we rely on a *local* analysis of the syntax to determine which variables should be related together, the packing is considered *globally* by the subsequent static analysis. Octagons are no longer attached to syntactic blocks and live throughout the abstract execution of the whole program.

The packing and automatic packing techniques are not tied to the octagon domain and can be used to limit the cost of any relational domain—indeed, in ASTRÉE, a similar technique is used also in a partitioning domain based on decision diagrams.

### 6.3. Results

We now present some experimental results with ASTRÉE on a few programs in the considered family, with varying code size.

*Packing results.* We first present statistics on the octagons selected by our automatic packing strategy. Our family has been split into two sub-families: the three lower, more recent, C programs have a slightly different structure. The code size is computed as the number of indented lines of C code after merging all the preprocessed source files together—thus eliminating all useless or redundant declarations in headers. Next to the number of variables, the number of octagons, and the average number of variables per octagon, we give the percentage of octagons that were proved useful *a posteriori*. More formally, we compare, after each transfer function application, the result obtained in the interval domain and in the octagon domain. If the bound for one variable in the octagon is strictly tighter than the bound

for the same variable in the interval domain, the octagon is tagged as useful—it needs to be useful only once to be tagged. As the memory consumption and the time cost depend respectively on $n^2$ and $n^3$, we show not only the average number of variables, but also the square—resp. cubic—root of the average of the squared—resp. cubed—sizes.

| Code size in lines | No. of vars. | No. of packs | Average size | $\sqrt{\sum n^2}$ | $\sqrt[3]{\sum n^3}$ | Useful (%) |
|---|---|---|---|---|---|---|
| 370 | 100 | 20 | 3.6 | 4.8 | 6.2 | 85 |
| 9, 500 | 1, 400 | 200 | 3.1 | 4.6 | 6.6 | 41 |
| 70, 000 | 14, 000 | 2, 470 | 3.5 | 5.2 | 7.8 | 57 |
| 70, 000 | 16, 000 | 2, 546 | 2.9 | 3.4 | 4.4 | 41 |
| 226, 000 | 47, 500 | 7, 429 | 3.5 | 4.5 | 5.8 | 52 |
| 400, 000 | 82, 000 | 12, 964 | 3.3 | 4.1 | 5.3 | 50 |

This table shows that the average size of the packs is almost constant while the number of packs grows roughly linearly with the code size. This means that the octagon domain with an adequate packing strategy has a time and memory cost that is *linear* with respect to the program size. Two other interesting pieces of information not presented in this table are that the largest packs contain only up to a *few dozen* variables and that a variable that is captured by a pack occurs on average one more time in a different pack.

*Analysis results.* We now compare the results of ASTRÉE on the considered program family with and without the octagon domain, all other abstract domains being enabled. We give, in both cases, the analysis time, the maximum memory consumption, and the number of false alarms. All the analyses have been carried on a 64-bit AMD Opteron 248 (2.2 GHz) workstation running Linux, using a single processor. We observed that, on a 32-bit architecture, the memory consumption is roughly one third smaller, which is due to the large usage of pointers in OCAML data-structures.

| | Without octagons | | | With octagons | | |
|---|---|---|---|---|---|---|
| Code size in lines | Analysis time | Max. memory | False alarms | Analysis time | Max. memory | False alarms |
| 370 | 1.7 s | 14 MB | 0 | 3.1 s | 16 MB | 0 |
| 9, 500 | 75 s | 75 MB | 8 | 160 s | 80 MB | 8 |
| 70, 000 | 3 h 17 mn | 537 MB | 58 | 1 h 16 mn | 582 MB | 0 |
| 70, 000 | 18 mn | 289 MB | 4 | 30 mn | 378 MB | 4 |
| 226, 000 | 7 h 28 mn | 1.0 GB | 165 | 6 h 36 mn | 1.3 GB | 1 |
| 400, 000 | 20 h 31 mn | 1.7 GB | 804 | 13 h 52 mn | 2.2 GB | 0 |

This table shows that the octagon domain is able to reduce the number of false alarms to only a few ones, and even to zero in some cases. Moreover, enabling the octagon domain adds roughly 30% to the total memory consumption in the worst case, which is very reasonable considering the precision gain. The analysis time does not seem to follow a logical pattern. Sometimes the analysis is longer with the octagon domain, which seems quite natural, but sometimes it is shorter. In order to explain this fact, we need to take into account the number of iterations with widening and narrowing of the main loop that are needed to stabilize our invariants. This is presented in the following table:

| | Without octagons | | With octagons | |
|---|---|---|---|---|
| Code size in lines | Number of iterations | Time per iteration | Number of iterations | Time per iteration |
| 370 | 12 | 0.14 s | 17 | 0.18 s |
| 9, 500 | 23 | 3.2 s | 39 | 4.1 s |
| 70, 000 | 159 | 74 s | 44 | 104 s |
| 70, 000 | 36 | 30 s | 38 | 49 s |
| 226, 000 | 144 | 178 s | 86 | 276 s |
| 400, 000 | 172 | 429 s | 96 | 520 s |

We now see clearly that the octagon domain makes each abstract iteration up to 65% slower, which is due to the extra time spent in octagon transfer functions and operators. The octagon domain also affects the number of required iterations, but in a non-easily predictable way. Sometimes, more iterations are required because we are trying to stabilize a greater amount of invariants. Sometimes, the octagon information can prove the stability of some variable bound quickly and save unstable widening steps in the interval domain. In our largest examples, the decreased number of iterations is sufficient to reduce the total analysis time even though each iteration takes longer. This shows that using an abstract domain adapted to the invariants of a program can increase both the precision and the efficiency of a static analysis at the same time.

*Octagon cost.* In order to determine more precisely which parts of the octagon domain are responsible for the increased computation time per iteration, we performed a few analyses using profiling. Unsurprisingly, we spend most of the analysis time closing our matrices: over 6% of the total analysis time is spent in the incremental strong closure. There is only one function in which the analyzer spends more time: the mark phase of OCAML's garbage collector—10% of the total analysis time. Also, the octagon algorithm coming right after the incremental strong closure is the forget operator, and it accounts for only 0.35% of the total analysis time. The non-incremental version of the strong closure corresponds to a negligible fraction of the analysis time because it is seldom called. We prefer to use the much faster incremental closure whenever possible.

### 6.4. Future work on ASTRÉE

ASTRÉE is currently limited to a small subset of the C programming language. Although this is quite realistic for the class of embedded command-control systems, it prevents us from exercising the octagon domain on the vast majority of available software. Important unsupported program features include pointer arithmetics, dynamic memory allocation, recursive data-structures, multi-threading, and interactions with operating systems via system and library calls. We plan to add support for some of these features in ASTRÉE. It will require a lot of work in areas orthogonal to numerical abstract domains. We hope to be able, in the future, to asses the usefulness of octagonal constraints for a broader subset of C programs. We would also like to determine what kinds of abstract transfer functions, widening operators, and packing strategies provide the best precision for a reasonable cost.

Another open question is how the octagon and the polyhedra domain compare in practice. Alas, experimentation is not possible in the context of ASTRÉE, for two reasons. Firstly, we are not aware of any polyhedron library that can abstract floating-point expressions,

nor interval linear forms, which are critical to the sound and precise analysis of our target software. Secondly, it would not be fair to compare the speed and memory consumption of a polyhedron library implemented using slow multi-precision rationals to the octagonal library implemented using fast machine floating-point numbers. What we propose to do, instead, is to plug the octagon abstract domain into other existing analyzers currently using the polyhedron domain. There is currently an ongoing effort to provide a common interface for both the octagon and polyhedron abstract domains, within the Apron project [1].

## 7. Conclusion

We have presented an abstract domain that is able to discover invariants of the form $\pm X \pm Y \leq c$ for a quadratic memory cost per abstract element and, in the worst case, a cubic time cost per abstract operation. When $\mathbb{I} = \mathbb{Q}$ and $\mathbb{I} = \mathbb{R}$, we have provided as many best abstract transfer functions and operators as possible. When $\mathbb{I} = \mathbb{Z}$ the choice is given to either lose a little precision, or retain best operators and transfer functions and have a $\mathcal{O}(n^4)$ worst-case time cost. At the crux of this construction lie modified versions of the Floyd-Warshall shortest path algorithm and Harvey and Stuckey's satisfiability testing algorithm. We have proved that the output of these adapted algorithms enjoys a saturation property which is fundamental to construct best precision operators. In order to provide the user with some control on the cost versus precision trade-off, we have provided several different ways to abstract all the basic semantical operators.

The octagon abstract domain presented here has been implemented as a robust and fast library in the C programming language and integrated into the ASTRÉE industrial-strength static analyzer [3]. Thanks to experimentations on real-life program analyses, we are able to provide experimental proof that the octagon domain indeed scales up to large programs while providing an important precision gain with respect to non-relational analyses. This precision improvement was key in the success of ASTRÉE, that is, the proof of absence of run-time errors in critical embedded fly-by-wire software found in Airbus planes.

*Future work.* Our work on the octagon domain may be extended in several directions. One issue is the closure algorithm for *integer* octagonal constraints. Unlike the rational and real octagonal constraints, which enjoy Floyd-Warshall-related cubic-time algorithms, the only closure algorithm for integer octagonal constraints we are aware of has a $\mathcal{O}(n^4)$ cost. Determining the exact complexity of the closure problem for integer octagonal constraints is interesting from a theoretic point-of-view, even though, from a practical point-of-view, we can safely use an incomplete closure if we do not mind a small precision loss. A second issue is the design of new widening and narrowing operators, possibly not based on point-wise extensions of interval-based operators. In particular, it would be quite interesting if we could design a widening operator that is insensitive to the chosen DBM representation of an octagon, so that it is possible to close the iterates. A third issue is the design of some more transfer functions. In particular, we designed inexact transfer functions on interval linear forms that are able to derive new relational constraints but only use non-relational information. New transfer functions that are, in terms of precision and cost, between these transfer functions and the costly polyhedron-based ones would be welcome. One may investigate whether best linear assignments and tests can be computed using a less costly technique than switching temporarily into the polyhedron domain. It is also interesting to look for new numerical

abstract domains that are more expressive than the octagon domain and still less costly than the polyhedron one. There is already some research in this direction: the so-called Two Variables per Linear Inequality domain by Simon et al. [52] for invariants of the form $\alpha X + \beta Y \leq c$ and the octahedra domain by Clarisó et al. [12] for invariants of the form $\sum_i \epsilon_i X_i \leq c$, $\epsilon_i \in \{-1, 0, 1\}$. Finally, further work is pursued on the ASTRÉE project at the ENS and the École Polytechnique to extend our analyzer to other program families and other kinds of properties to be proved and we are quite confident that the octagon domain will be useful in the future of ASTRÉE.

## Appendix A. List of symbols and notations

### A.1. Matrix-related definitions

| | | |
|---|---|---|
| DBM | set of all Difference Bound Matrices | Section 2.1 |
| CDBM | set of coherent Difference Bound Matrices | Section 2.2 |
| $i \mapsto \overline{i}$ | switches between positive and negative variables in $\mathcal{V}'$ | Section 2.2 |
| $\sqsubseteq^{\mathsf{DBM}}$ | point-wise partial order | Section 2.3 |
| $\sqcup^{\mathsf{DBM}}$ | least upper bound for $\sqsubseteq^{\mathsf{DBM}}$ | Section 2.3 |
| $\sqcap^{\mathsf{DBM}}$ | greatest lower bound for $\sqsubseteq^{\mathsf{DBM}}$ | Section 2.3 |
| $\top^{\mathsf{DBM}}$ | greatest element for $\sqsubseteq^{\mathsf{DBM}}$ | Section 2.3 |
| $\bot^{\mathsf{DBM}}$ | least element for $\sqsubseteq^{\mathsf{DBM}}$ | Section 2.3 |
| $*$ | shortest-path closure | Section 3.2 |
| $\bullet$ | strong closure | Section 3.3 |
| $Inc^{\bullet}$ | incremental strong closure | Section 3.4 |
| $Inc^{T}$ | incremental tight closure | Section 3.5 |
| $\gamma^{Pot}$ | potential set concretization | Section 2.1 |
| $\gamma^{Oct}$ | octagonal concretization | Section 2.2 |
| $\alpha^{Oct}$ | octagonal (partial) abstraction | Section 2.3 |

### A.2. Concrete semantic operators

| | | |
|---|---|---|
| $[\![ expr ]\!]$ | concrete expression evaluation | Fig. 14 |
| $[\![ test ]\!]$ | concrete test evaluation | Fig. 19 |
| $\{\!\| V \leftarrow ? \|\!\}$ | concrete forget transfer function | Section 4.2 |
| $\{\!\| V \leftarrow e \|\!\}$ | concrete assignment transfer function | Section 4.4 |
| $\{\!\| V \rightarrow e \|\!\}$ | concrete backward assignment transfer function | Section 4.6 |
| $\{\!\| test ? \|\!\}$ | concrete test transfer function | Section 4.5 |

A.3. Octagon abstract operators

| | | |
|---|---|---|
| $\cap^{Oct}$ | intersection abstraction | Section 4.1 |
| $\cup^{Oct}$ | union abstraction | Section 4.1 |
| $\triangledown^{Oct}_{std}$ | standard widening | Section 4.7 |
| $\triangle^{Oct}_{std}$ | standard narrowing | Section 4.7 |
| $\triangledown^{Oct}_{th}$ | widening with thresholds | Section 4.7 |
| $Oct$ | conversion from intervals to octagons | Section 4.3 |
| $Oct$ | conversion from polyhedra to octagons | Section 4.3 |
| $Int$ | conversion from octagons to intervals | Section 4.3 |
| $Poly$ | conversion from octagons to polyhedra | Section 4.3 |
| $\pi_i$ | extracts bounds of variable $i$ from octagon | Section 4.3 |
| $\{\!\mid V \leftarrow ? \mid\!\}^{Oct}$ | forget abstraction | Section 4.2 |
| $\{\!\mid V \leftarrow e \mid\!\}^{Oct}_{exact}$ | simple assignment abstraction | Fig. 15 |
| $\{\!\mid V \leftarrow e \mid\!\}^{Oct}_{nonrel}$ | interval-based assignment abstraction | Section 4.4 |
| $\{\!\mid V \leftarrow e \mid\!\}^{Oct}_{rel}$ | interval linear form assignment abstraction | Fig. 16 |
| $\{\!\mid V \leftarrow e \mid\!\}^{Oct}_{poly}$ | polyhedron-based assignment abstraction | Section 4.4 |
| $\{\!\mid test\ ? \mid\!\}^{Oct}_{exact}$ | simple test abstraction | Fig. 20 |
| $\{\!\mid test\ ? \mid\!\}^{Oct}_{nonrel}$ | interval-based test abstraction | Section 4.5 |
| $\{\!\mid test\ ? \mid\!\}^{Oct}_{rel}$ | interval linear form test abstraction | Fig. 21 |
| $\{\!\mid test\ ? \mid\!\}^{Oct}_{poly}$ | polyhedron-based test abstraction | Section 4.5 |
| $\{\!\mid V \rightarrow e \mid\!\}^{Oct}_{exact}$ | simple backward abstraction | Fig. 23 |
| $\{\!\mid V \rightarrow e \mid\!\}^{Oct}_{nonrel}$ | interval-based backward abstraction | Section 4.6 |
| $\{\!\mid V \rightarrow e \mid\!\}^{Oct}_{rel}$ | interval linear form backward abstraction | Fig. 24 |
| $\{\!\mid V \rightarrow e \mid\!\}^{Oct}_{poly}$ | polyhedron-based backward abstraction | Section 4.6 |

A.4. Other semantical operators

| | | |
|---|---|---|
| $\boxplus$ | addition of interval linear forms | Fig. 17 |
| $\boxminus$ | opposite of an interval linear form | Fig. 17 |
| $[\![ e ]\!]^{Int}$ | expression evaluation in the interval domain | Section 4.4 |

## Appendix B. Proofs of theorems

**Theorem 1.** $\gamma^{Pot}(\mathbf{m}) = \emptyset \iff \mathcal{G}(\mathbf{m})$ *has a simple cycle with a strictly negative total weight [14, Theorem 25.17].*

(Stated in Section 3.1.)

**Proof:** See, for instance, [14, Section 25.5, Theorem 25.17]. □

**Theorem 2.** *When* $\mathbb{I} \in \{\mathbb{Q}, \mathbb{R}\}$, $\gamma^{Oct}(\mathbf{m}) = \emptyset \iff \gamma^{Pot}(\mathbf{m}) = \emptyset$.

(Stated in Section 3.1.)

**Proof:** If $\gamma^{Pot}(\mathbf{m}) = \emptyset$ then, obviously, $\gamma^{Oct}(\mathbf{m}) = \emptyset$ by definition of $\gamma^{Oct}$.

Suppose conversely that $\gamma^{Pot}(\mathbf{m}) \neq \emptyset$. We prove that $\gamma^{Oct}(\mathbf{m}) \neq \emptyset$ as well. Let $\vec{v}\,' = (v'_1, \ldots, v'_{2n}) \in \gamma^{Pot}(\mathbf{m})$. We have $\forall i, j,\ v'_j - v'_i \leq \mathbf{m}_{ij}$. By coherence of $\mathbf{m}$, this implies $\forall i, j,\ v'_{\bar{i}} - v'_{\bar{j}} \leq \mathbf{m}_{\bar{j}\,\bar{i}} = \mathbf{m}_{ij}$, which means that $\vec{w}\,' \stackrel{\text{def}}{=} (-v'_2, -v'_1, \ldots, -v'_{2n}, -v'_{2n-1}) \in \gamma^{Pot}(\mathbf{m})$ as well. As $\gamma^{Pot}(\mathbf{m})$ is defined by an intersection of half-spaces, it is convex, so, the point $\vec{z}\,' \stackrel{\text{def}}{=} (\vec{v}\,' + \vec{w}\,')/2$ is also in $\gamma^{Pot}(\mathbf{m})$. Moreover, the coordinates $z'_i$ of $\vec{z}\,'$ imply: $\forall i,\ z'_{2i} = (v'_{2i} - v'_{2i-1})/2 = -z'_{2i-1}$. By definition of $\gamma^{Oct}$, this means that $(z_1, z_3, \ldots, z_{2n-1}) \in \gamma^{Oct}(\mathbf{m})$.

Note that, when $\mathbb{I} = \mathbb{Z}$, this proof does not hold because $\vec{z}\,'$ may not be in $\gamma^{Pot}(\mathbf{m})$ whenever some $v'_{2i} - v'_{2i-1}$ is not even. □

**Theorem 3.** *If* $\mathbb{I} \in \{\mathbb{Q}, \mathbb{R}\}$ *and* $\mathbf{m}$ *is strongly closed, then:*

1. $\forall i, j,$ *if* $\mathbf{m}_{ij} < +\infty,$ *then* $\exists (v_1, \ldots, v_n) \in \gamma^{Oct}(\mathbf{m})$ *such that* $v'_j - v'_i = \mathbf{m}_{ij}$, *and*
2. $\forall i, j,$ *if* $\mathbf{m}_{ij} = +\infty,$ *then* $\forall M < +\infty,\ \exists (v_1, \ldots, v_n) \in \gamma^{Oct}(\mathbf{m})$ *such that* $v'_j - v'_i \geq M$,

*where the* $v'_k$ *are derived from the* $v_k$ *by* $v'_{2k-1} \stackrel{\text{def}}{=} v_k$ *and* $v'_{2k} \stackrel{\text{def}}{=} -v_k$.

(Stated in Section 3.3.)

**Proof:**

1. Let $(i_0, j_0)$ be a pair such that $\mathbf{m}_{i_0 j_0} < +\infty$.
   The case $i_0 = j_0$ is trivial: as $\mathbf{m}$ is strongly closed, we have $\mathbf{m}_{i_0 j_0} = 0$ and any point $(v_1, \ldots, v_n) \in \gamma^{Oct}(\mathbf{m}) \neq \emptyset$ is such that $v'_{i_0} - v'_{i_0} \leq 0$.
   We now consider the much more complex case of $i_0 \neq j_0$. We denote by $\mathbf{m}'$ the matrix equal to $\mathbf{m}$ except that $\mathbf{m}'_{j_0 i_0} \stackrel{\text{def}}{=} \mathbf{m}'_{\bar{i}_0\,\bar{j}_0} \stackrel{\text{def}}{=} -\mathbf{m}_{i_0 j_0}$. It is a coherent DBM.
   Let us define $S$ by $S \stackrel{\text{def}}{=} \{(v_1, \ldots, v_n) \in \gamma^{Oct}(\mathbf{m}) \mid v'_{j_0} - v'_{i_0} = \mathbf{m}_{i_0 j_0}\}$ where the $v'_k$ are derived from the $v_k$ by stating that $v'_{2k-1} \stackrel{\text{def}}{=} v_k$ and $v'_{2k} \stackrel{\text{def}}{=} -v_k$. We first prove that $\gamma^{Oct}(\mathbf{m}') = S$.
   - As $\gamma^{Oct}(\mathbf{m}) \neq \emptyset$, there is no cycle with strictly negative weight in $\mathbf{m}$, so, $\mathbf{m}_{j_0 i_0} \geq -\mathbf{m}_{i_0 j_0}$ and, similarly, $\mathbf{m}_{\bar{i}_0\,\bar{j}_0} \geq -\mathbf{m}_{\bar{j}_0\,\bar{i}_0}$. This means that $\mathbf{m}' \sqsubseteq^{\mathsf{DBM}} \mathbf{m}$. Hence, $\gamma^{Oct}(\mathbf{m}') \subseteq \gamma^{Oct}(\mathbf{m})$.

- Consider $(v_1, \ldots, v_n) \in \gamma^{Oct}(\mathbf{m}')$. Then, $-\mathbf{m}'_{j_0 i_0} \leq v'_{j_0} - v'_{i_0} \leq \mathbf{m}'_{i_0 j_0}$, which, by definition of $\mathbf{m}'$, implies $v'_{j_0} - v'_{i_0} = \mathbf{m}'_{i_0 j_0}$. Together with the preceding point, this implies $\gamma^{Oct}(\mathbf{m}') \subseteq S$
- Conversely, if $(v_1, \ldots, v_n) \in S$, then it is easy to see that $\forall i,\ j,\ v'_j - v'_i \leq \mathbf{m}'_{ij}$.

To prove the desired property, it is now sufficient to check that $\gamma^{Oct}(\mathbf{m}')$ is not empty, that is, that $\mathbf{m}'$ has no simple cycle with a strictly negative total weight. Suppose that there exists such a simple cycle $\langle i = k_1, \ldots, k_l = i \rangle$. We distinguish several cases that all lead to a contradiction:

- If neither the arc from $j_0$ to $i_0$ nor the arc from $\overline{i_0}$ to $\overline{j_0}$ is in this strictly negative cycle, this cycle also exists in $\mathcal{G}(\mathbf{m})$ and $\gamma^{Oct}(\mathbf{m}) = \emptyset$, which is not true.
- Suppose now that the strictly negative cycle contains only one of these two arcs, say, the arc from $j_0$ to $i_0$. It contains this arc only once, as the cycle is simple. By adequately shifting the indices of the cycle, we can assume the existence of a strictly negative cycle of the form: $\langle k_1 = j_0, k_2 = i_0, k_3, \ldots, k_l = j_0 \rangle$, where $\langle k_2 = i_0, k_3, \ldots, k_l = j_0 \rangle$ is a path in $\mathcal{G}(\mathbf{m})$. This path is such that:

$$\sum_{x=2}^{l-1} \mathbf{m}_{k_x k_{x+1}} < -\mathbf{m}'_{j_0 i_0} = \mathbf{m}_{i_0 j_0}$$

  that is, there is a path in $\mathbf{m}$ from $i_0$ to $j_0$ with a weight strictly smaller than $\mathbf{m}_{i_0 j_0}$, which is impossible because, $\mathbf{m}$ being strongly closed, it is also closed.
- Finally, suppose that both arcs, from $j_0$ to $i_0$ and from $\overline{i_0}$ to $\overline{j_0}$, are in this cycle. Each can appear only once, so we can—without loss of generality—rewrite the cycle as: $\langle k_1 = \overline{j_0}, \ldots, k_a = j_0, k_{a+1} = i_0, \ldots, k_b = \overline{i_0}, k_{b+1} = \overline{j_0} \rangle$, where the sub-paths $\langle k_1 = \overline{j_0}, \ldots, k_a = j_0 \rangle$ and $\langle k_{a+1} = i_0, \ldots, k_b = \overline{i_0} \rangle$ are in $\mathcal{G}(\mathbf{m})$. We then have:

$$\left( \sum_{x=1}^{a-1} \mathbf{m}_{k_x k_{x+1}} \right) + \mathbf{m}'_{j_0 i_0} + \left( \sum_{x=a+1}^{b-1} \mathbf{m}_{k_x k_{x+1}} \right) + \mathbf{m}'_{\overline{i_0}\,\overline{j_0}} < 0.$$

  Because $\mathbf{m}$ is strongly closed, it is also closed, and we have:

$$\mathbf{m}_{\overline{j_0} j_0} \leq \sum_{x=1}^{a-1} \mathbf{m}_{k_x k_{x+1}} \quad \text{and} \quad \mathbf{m}_{i_0 \overline{i_0}} \leq \sum_{x=a+1}^{b-1} \mathbf{m}_{k_x k_{x+1}}$$

  which can be combined with the preceding inequality to give:

$$\mathbf{m}_{\overline{j_0} j_0} + \mathbf{m}'_{j_0 i_0} + \mathbf{m}_{i_0 \overline{i_0}} + \mathbf{m}'_{\overline{i_0}\,\overline{j_0}} < 0$$

  that is:

$$\mathbf{m}_{i_0 j_0} > (\mathbf{m}_{\overline{j_0} j_0} + \mathbf{m}_{i_0 \overline{i_0}})/2$$

  which contradicts the fact that $\mathbf{m}$ is strongly closed.

2. Let $(i_0, j_0)$ be a pair such that $\mathbf{m}_{ij} = +\infty$ and $M \in \mathbb{I}$. We denote by $\mathbf{m}'$ the DBM equal to $\mathbf{m}$ except that $\mathbf{m}'_{j_0 i_0} \stackrel{\text{def}}{=} \mathbf{m}'_{\overline{i_0}\,\overline{j_0}} \stackrel{\text{def}}{=} \min(\mathbf{m}_{j_0 i_0}, -M)$. We can prove analogously to (1.) that $\gamma^{Oct}(\mathbf{m}') = \{(v_1, \ldots, v_n) \in \gamma^{Oct}(\mathbf{m}) \mid v'_{j_0} - v'_{i_0} \geq M\}$ and $\gamma^{Oct}(\mathbf{m}') \neq \emptyset$. □

**Theorem 4.** *If $\mathbb{I} \in \{\mathbb{Q}, \mathbb{R}\}$ and $\gamma^{Oct}(\mathbf{m}) \neq \emptyset$, then:*

$$\mathbf{m}^\bullet = (\alpha^{Oct} \circ \gamma^{Oct})(\mathbf{m}) = \inf_{\sqsubseteq^{\mathsf{DBM}}} \{X^\sharp \in \mathsf{DBM} \mid \gamma^{Oct}(\mathbf{m}) = \gamma^{Oct}(X^\sharp)\}.$$

(Stated in Section 3.3.)

**Proof:** This is directly implied by the previous theorem. □

**Theorem 5.** *If $\gamma^{Oct}(\mathbf{m}) \neq \emptyset$ then $\mathbf{m}^\bullet$ computed by Definition 2 is the strong closure as defined by Definition 1.*

(Stated in Section 3.4.)

**Proof:** Suppose that $\gamma^{Oct}(\mathbf{m}) \neq \emptyset$ and let $\mathbf{m}^\bullet$ be the result computed by the modified Floyd-Warshall algorithm of Definition 2. We prove that $\mathbf{m}^\bullet$ satisfies the three criteria of Definition 1.

By definition $\forall i,\ \mathbf{m}^\bullet_{ii} = 0$.

We now prove that $\forall i, j,\ \mathbf{m}^\bullet_{ij} \leq (\mathbf{m}^\bullet_{i\bar{\imath}} + \mathbf{m}^\bullet_{\bar{\jmath}j})/2$.

First of all, we prove that for any matrix $\mathbf{n}$, $S(\mathbf{n})_{ij} \leq (S(\mathbf{n})_{i\bar{\imath}} + S(\mathbf{n})_{\bar{\jmath}j})/2$. Indeed, $\forall i,\ S(\mathbf{n})_{i\bar{\imath}} = \min(\mathbf{n}_{i\bar{\imath}}, (\mathbf{n}_{i\bar{\imath}} + \mathbf{n}_{i\bar{\imath}})/2) = \mathbf{n}_{i\bar{\imath}}$, so $\forall i, j,\ S(\mathbf{n})_{ij} \leq (\mathbf{n}_{i\bar{\imath}} + \mathbf{n}_{\bar{\jmath}j})/2 = (S(\mathbf{n})_{i\bar{\imath}} + S(\mathbf{n})_{\bar{\jmath}j})/2$. Applying this property for $\mathbf{n} \stackrel{\text{def}}{=} C^{2n-1}(\mathbf{m}^{n-1})$, we get that $\forall i, j,\ \mathbf{m}^n_{ij} \leq (\mathbf{m}^n_{i\bar{\imath}} + \mathbf{m}^n_{\bar{\jmath}j})/2$. This implies that if $i \neq j$, then $\mathbf{m}^\bullet_{ij} \leq (\mathbf{m}^\bullet_{i\bar{\imath}} + \mathbf{m}^\bullet_{\bar{\jmath}j})/2$. Whenever $i = j$, $\mathbf{m}^\bullet_{ii} = 0$ which is smaller than $(\mathbf{m}^\bullet_{i\bar{\imath}} + \mathbf{m}^\bullet_{\bar{\jmath}j})/2 = (\mathbf{m}^n_{i\bar{\imath}} + \mathbf{m}^n_{\bar{\jmath}j})/2$, or else, there would be a cycle with strictly negative total weight in $\mathcal{G}(\mathbf{m}^n)$ implying $\gamma^{Oct}(\mathbf{m}^n) = \emptyset$ , and hence, $\gamma^{Oct}(\mathbf{m}) = \emptyset$, which is not true.

Finally, we prove that $\forall i, j, k,\ \mathbf{m}^\bullet_{ij} \leq \mathbf{m}^\bullet_{ik} + \mathbf{m}^\bullet_{kj}$. This is a difficult property to prove which justifies the complexity of the modified Floyd-Warshall algorithm of Definition 2. We will use several lemmas.

**Lemma 1.** *Let $\mathbf{n}$ be a coherent DBM such that $\gamma^{Oct}(\mathbf{n}) \neq \emptyset$ and there exists some $k$ such that $\forall i, j,\ \mathbf{n}_{ij} \leq \mathbf{n}_{ik} + \mathbf{n}_{kj}$ and $\forall i, j,\ \mathbf{n}_{ij} \leq \mathbf{n}_{i\bar{k}} + \mathbf{n}_{\bar{k}j}$. We prove that $\forall i, j,\ S(\mathbf{n})_{ij} \leq S(\mathbf{n})_{ik} + S(\mathbf{n})_{kj}$.*

*Case 1.* $S(\mathbf{n})_{ik} = \mathbf{n}_{ik}$ and $S(\mathbf{n})_{kj} = \mathbf{n}_{kj}$.

We have obviously:

$$\begin{array}{lll} S(\mathbf{n})_{ij} & \leq \mathbf{n}_{ij} & \text{(by definition of } S(\mathbf{n})) \\ & \leq \mathbf{n}_{ik} + \mathbf{n}_{kj} & \text{(by hypothesis)} \\ & = S(\mathbf{n})_{ik} + S(\mathbf{n})_{kj} & \text{(case hypothesis)} \end{array}$$

*Case 2.* $S(\mathbf{n})_{ik} = (\mathbf{n}_{i\bar{\imath}} + \mathbf{n}_{\bar{k}k})/2$ and $S(\mathbf{n})_{kj} = \mathbf{n}_{kj}$ (or the symmetric case $S(\mathbf{n})_{ik} = \mathbf{n}_{ik}$ and $S(\mathbf{n})_{kj} = (\mathbf{n}_{k\bar{k}} + \mathbf{n}_{\bar{\jmath}j})/2$).

Using the hypothesis twice, we get $\mathbf{n}_{\overline{\jmath}j} \leq \mathbf{n}_{\overline{\jmath}\overline{k}} + \mathbf{n}_{\overline{k}j} \leq \mathbf{n}_{\overline{\jmath}\overline{k}} + (\mathbf{n}_{\overline{k}k} + \mathbf{n}_{kj})$ (1), so, we obtain:

$$\begin{array}{ll} S(\mathbf{n})_{ij} \leq \mathbf{n}_{i\overline{\imath}}/2 + \mathbf{n}_{\overline{\jmath}j}/2 & \text{(by definition of } S(\mathbf{n})) \\ \leq \mathbf{n}_{i\overline{\imath}}/2 + (\mathbf{n}_{\overline{\jmath}\overline{k}} + \mathbf{n}_{\overline{k}k} + \mathbf{n}_{kj})/2 & \text{(by (1))} \\ \leq \mathbf{n}_{i\overline{\imath}}/2 + \mathbf{n}_{\overline{k}k}/2 + \mathbf{n}_{kj} & \text{(by coherence)} \\ = S(\mathbf{n})_{ik} + S(\mathbf{n})_{kj} & \text{(case hypothesis)} \end{array}$$

*Case 3.* $S(\mathbf{n})_{ik} = (\mathbf{n}_{i\overline{\imath}} + \mathbf{n}_{\overline{k}k})/2$ and $S(\mathbf{n})_{kj} = (\mathbf{n}_{k\overline{k}} + \mathbf{n}_{\overline{\jmath}j})/2)$.

Now we use the fact that $\gamma^{Oct}(\mathbf{m}) \neq \emptyset$ so that the cycle $\langle k, \overline{k}, k\rangle$ has a positive weight, so, $0 \leq \mathbf{n}_{k\overline{k}} + \mathbf{n}_{\overline{k}k}$ (1) and:

$$\begin{array}{ll} S(\mathbf{n})_{ij} \leq (\mathbf{n}_{i\overline{\imath}} + \mathbf{n}_{\overline{\jmath}j})/2 & \text{(by definition of } S(\mathbf{n})) \\ \leq (\mathbf{n}_{i\overline{\imath}} + (\mathbf{n}_{\overline{k}k} + \mathbf{n}_{k\overline{k}}) + \mathbf{n}_{\overline{\jmath}j})/2 & \text{(by (1))} \\ = S(\mathbf{n})_{ik} + S(\mathbf{n})_{kj} & \text{(case hypothesis)} \end{array}$$

**Lemma 2.** *Let* $\mathbf{n}$ *be a coherent DBM such that* $\gamma^{Oct}(\mathbf{n}) \neq \emptyset$ *and there exists some* $k$ *such that* $\forall i \neq j,\ \mathbf{n}_{ij} \leq \mathbf{n}_{ik} + \mathbf{n}_{kj}$ *and* $\forall i \neq j,\ \mathbf{n}_{ij} \leq \mathbf{n}_{i\overline{k}} + \mathbf{n}_{\overline{k}j}$. *We prove that* $\forall o, \forall i \neq j,\ C^o(\mathbf{n})_{ij} \leq C^o(\mathbf{n})_{ik} + C^o(\mathbf{n})_{kj}$.

There are five different cases for the value of $C^o(\mathbf{n})_{ik}$ and five cases for the value of $C^o(\mathbf{n})_{kj}$:

| | |
|---|---|
| (1) $C^o(\mathbf{n})_{ik} = \mathbf{n}_{ik}$ | (1) $C^o(\mathbf{n})_{kj} = \mathbf{n}_{kj}$ |
| (2) $C^o(\mathbf{n})_{ik} = \mathbf{n}_{io} + \mathbf{n}_{ok}$ | (2) $C^o(\mathbf{n})_{kj} = \mathbf{n}_{ko} + \mathbf{n}_{oj}$ |
| (3) $C^o(\mathbf{n})_{ik} = \mathbf{n}_{i\overline{o}} + \mathbf{n}_{\overline{o}k}$ | (3) $C^o(\mathbf{n})_{kj} = \mathbf{n}_{k\overline{o}} + \mathbf{n}_{\overline{o}j}$ |
| (4) $C^o(\mathbf{n})_{ik} = \mathbf{n}_{io} + \mathbf{n}_{o\overline{o}} + \mathbf{n}_{\overline{o}k}$ | (4) $C^o(\mathbf{n})_{kj} = \mathbf{n}_{ko} + \mathbf{n}_{o\overline{o}} + \mathbf{n}_{\overline{o}j}$ |
| (5) $C^o(\mathbf{n})_{ik} = \mathbf{n}_{i\overline{o}} + \mathbf{n}_{\overline{o}o} + \mathbf{n}_{ok}$ | (5) $C^o(\mathbf{n})_{kj} = \mathbf{n}_{k\overline{o}} + \mathbf{n}_{\overline{o}o} + \mathbf{n}_{oj}$ |

In the following, we will denote by $(a, b)$ the case where the value of $C^o(\mathbf{n})_{ik}$ is defined by the $a$th case and the value of $C^o(\mathbf{n})_{kj}$ is defined by the $b$th case. We then have 25 different cases to inspect.

To reduce the number of cases to elaborate on, we use the strong symmetry of the definition of $C^o(\mathbf{n})$ with respect to $o$ and $\overline{o}$ together with the symmetry of the hypotheses with respect to $k$ and $\overline{k}$ and the fact that $\forall i, j,\ \mathbf{n}_{ij} = \mathbf{n}_{\overline{\jmath}\overline{\imath}}$ by coherence of $\mathbf{n}$.

We also use the fact that the analysis of the case $(a, b)$ for $a \neq b$ is very similar to the analysis of $(b, a)$, so, we will suppose $a \leq b$.

We will also often use the fact that $\forall i, j,\ \mathbf{n}_{ij} + \mathbf{n}_{ji} \geq 0$, which is the consequence of the fact that $\langle i, j, i\rangle$ is a cycle in $\mathbf{n}$ with positive weight since $\gamma^{Oct}(\mathbf{n}) \neq \emptyset$.

*Case 1.* (1, 1).

We have, by hypothesis, $\mathbf{n}_{ij} \leq \mathbf{n}_{ik} + \mathbf{n}_{kj}$ (1), so, obviously:

$$\begin{array}{ll} C^o(\mathbf{n})_{ij} \leq \mathbf{n}_{ij} & \text{(by definition of } C^o(\mathbf{n})) \\ \leq \mathbf{n}_{ik} + \mathbf{n}_{kj} & \text{(by (1))} \\ = C^o(\mathbf{n})_{ik} + C^o(\mathbf{n})_{kj} & \text{(case hypothesis)} \end{array}$$

*Case 2.* (1, 2) (and (1, 3) by $(o, \overline{o})$ symmetry).
Sub-case 1: $i \neq o$.
We have, by hypothesis, $\mathbf{n}_{io} \leq \mathbf{n}_{ik} + \mathbf{n}_{ko}$ (1), so:

$$\begin{array}{lll} C^o(\mathbf{n})_{ij} & \leq \mathbf{n}_{io} + \mathbf{n}_{oj} & \text{(by definition of } C^o(\mathbf{n})\text{)} \\ & \leq (\mathbf{n}_{ik} + \mathbf{n}_{ko}) + \mathbf{n}_{oj} & \text{(by (1))} \\ & = C^o(\mathbf{n})_{ik} + C^o(\mathbf{n})_{kj} & \text{(case hypothesis)} \end{array}$$

Sub-case 2: $i = o$.
We know that $\mathbf{n}_{ik} + \mathbf{n}_{ko} \geq 0$ (1), so:

$$\begin{array}{lll} C^o(\mathbf{n})_{ij} & \leq \mathbf{n}_{oj} & \text{(by definition of } C^o(\mathbf{n})\text{)} \\ & \leq (\mathbf{n}_{ik} + \mathbf{n}_{ko}) + \mathbf{n}_{oj} & \text{(by (1))} \\ & = C^o(\mathbf{n})_{ik} + C^o(\mathbf{n})_{kj} & \text{(case hypothesis)} \end{array}$$

*Case 3.* (1, 4) (and (1, 5) by $(o, \overline{o})$ symmetry).
Sub-case 1: $i \neq o$.
We have, by hypothesis, $\mathbf{n}_{io} \leq \mathbf{n}_{ik} + \mathbf{n}_{ko}$ (1), so:

$$\begin{array}{lll} C^o(\mathbf{n})_{ij} & \leq \mathbf{n}_{io} + \mathbf{n}_{o\overline{o}} + \mathbf{n}_{\overline{o}j} & \text{(by definition of } C^o(\mathbf{n})\text{)} \\ & \leq (\mathbf{n}_{ik} + \mathbf{n}_{ko}) + \mathbf{n}_{o\overline{o}} + \mathbf{n}_{\overline{o}j} & \text{(by (1))} \\ & = C^o(\mathbf{n})_{ik} + C^o(\mathbf{n})_{kj} & \text{(case hypothesis)} \end{array}$$

Sub-case 2: $i = o$.
As in the second case, we have $\mathbf{n}_{ik} + \mathbf{n}_{ko} \geq 0$ (1), so:

$$\begin{array}{lll} C^o(\mathbf{n})_{ij} & \leq \mathbf{n}_{i\overline{o}} + \mathbf{n}_{\overline{o}j} & \text{(by definition of } C^o(\mathbf{n})\text{)} \\ & \leq (\mathbf{n}_{ik} + \mathbf{n}_{ko}) + \mathbf{n}_{o\overline{o}} + \mathbf{n}_{\overline{o}j} & \text{(by (1))} \\ & = C^o(\mathbf{n})_{ik} + C^o(\mathbf{n})_{kj} & \text{(case hypothesis)} \end{array}$$

*Case 4.* (2, 2) (and (3, 3) by $(o, \overline{o})$ symmetry).
We know that $\mathbf{n}_{ok} + \mathbf{n}_{ko} \geq 0$ (1), so:

$$\begin{array}{lll} C^o(\mathbf{n})_{ij} & \leq \mathbf{n}_{io} + \mathbf{n}_{oj} & \text{(by definition of } C^o(\mathbf{n})\text{)} \\ & \leq \mathbf{n}_{io} + (\mathbf{n}_{ok} + \mathbf{n}_{ko}) + \mathbf{n}_{oj} & \text{(by (1))} \\ & = C^o(\mathbf{n})_{ik} + C^o(\mathbf{n})_{kj} & \text{(case hypothesis)} \end{array}$$

*Case 5.* (2, 3).
We have, by hypothesis, $\mathbf{n}_{o\overline{o}} \leq \mathbf{n}_{ok} + \mathbf{n}_{k\overline{o}}$ (1), so:

$$\begin{array}{lll} C^o(\mathbf{n})_{ij} & \leq \mathbf{n}_{io} + \mathbf{n}_{o\overline{o}} + \mathbf{n}_{\overline{o}j} & \text{(by definition of } C^o(\mathbf{n})\text{)} \\ & \leq \mathbf{n}_{io} + (\mathbf{n}_{ok} + \mathbf{n}_{k\overline{o}}) + \mathbf{n}_{\overline{o}j} & \text{(by (1))} \\ & = C^o(\mathbf{n})_{ik} + C^o(\mathbf{n})_{kj} & \text{(case hypothesis)} \end{array}$$

*Case 6.* (2, 4) (and (3, 5) by $(o, \overline{o})$ symmetry).

We use, as in the fourth case, $\mathbf{n}_{ok} + \mathbf{n}_{ko} \geq 0$ (1), so:

$$\begin{aligned} C^o(\mathbf{n})_{ij} &\leq \mathbf{n}_{io} + \mathbf{n}_{o\overline{o}} + \mathbf{n}_{\overline{o}j} && \text{(by definition of } C^o(\mathbf{n})) \\ &\leq \mathbf{n}_{io} + (\mathbf{n}_{ok} + \mathbf{n}_{ko}) + \mathbf{n}_{o\overline{o}} + \mathbf{n}_{\overline{o}j} && \text{(by (1))} \\ &= C^o(\mathbf{n})_{ik} + C^o(\mathbf{n})_{kj} && \text{(case hypothesis)} \end{aligned}$$

*Case 7.* (2, 5) (and (3, 4) by $(o, \overline{o})$ symmetry).

We use the fact that $\mathbf{n}_{o\overline{o}} + \mathbf{n}_{\overline{o}o} \geq 0$ (1), together with the hypothesis $\mathbf{n}_{o\overline{o}} \leq \mathbf{n}_{ok} + \mathbf{n}_{k\overline{o}}$ (2), so:

$$\begin{aligned} C^o(\mathbf{n})_{ij} &\leq \mathbf{n}_{io} + \mathbf{n}_{oj} && \text{(by definition of } C^o(\mathbf{n})) \\ &\leq \mathbf{n}_{io} + (\mathbf{n}_{o\overline{o}} + \mathbf{n}_{\overline{o}o}) + \mathbf{n}_{oj} && \text{(by (1))} \\ &\leq \mathbf{n}_{io} + ((\mathbf{n}_{ok} + \mathbf{n}_{k\overline{o}}) \\ &\quad + \mathbf{n}_{\overline{o}o}) + \mathbf{n}_{oj} && \text{(by (2))} \\ &= C^o(\mathbf{n})_{ik} + C^o(\mathbf{n})_{kj} && \text{(case hypothesis)} \end{aligned}$$

*Case 8.* (4, 4) (and (5, 5) by $(o, \overline{o})$ symmetry).

We use, as in the seventh case, $\mathbf{n}_{o\overline{o}} + \mathbf{n}_{\overline{o}o} \geq 0$ (1), together with the hypothesis $\mathbf{n}_{\overline{o}o} \leq \mathbf{n}_{\overline{o}k} + \mathbf{n}_{ko}$ (2), so:

$$\begin{aligned} C^o(\mathbf{n})_{ij} &\leq \mathbf{n}_{io} + \mathbf{n}_{o\overline{o}} + \mathbf{n}_{\overline{o}j} && \text{(by definition of } C^o(\mathbf{n})) \\ &\leq \mathbf{n}_{io} + \mathbf{n}_{o\overline{o}} + (\mathbf{n}_{\overline{o}o} + \mathbf{n}_{o\overline{o}}) + \mathbf{n}_{\overline{o}j} && \text{(by (1))} \\ &\leq \mathbf{n}_{io} + \mathbf{n}_{o\overline{o}} + ((\mathbf{n}_{\overline{o}k} + \mathbf{n}_{ko}) \\ &\quad + \mathbf{n}_{o\overline{o}}) + \mathbf{n}_{\overline{o}j} && \text{(by (2))} \\ &= C^o(\mathbf{n})_{ik} + C^o(\mathbf{n})_{kj} && \text{(case hypothesis)} \end{aligned}$$

*Case 9.* (4, 5).

We use, as in the seventh case, $\mathbf{n}_{o\overline{o}} + \mathbf{n}_{\overline{o}o} \geq 0$ (1) and $\mathbf{n}_{\overline{o}k} + \mathbf{n}_{k\overline{o}} \geq 0$ (2), so:

$$\begin{aligned} C^o(\mathbf{n})_{ij} &\leq \mathbf{n}_{io} + \mathbf{n}_{oj} && \text{(by definition of } C^o(\mathbf{n})) \\ &\leq \mathbf{n}_{io} + (\mathbf{n}_{o\overline{o}} + \mathbf{n}_{\overline{o}o}) \\ &\quad + (\mathbf{n}_{\overline{o}k} + \mathbf{n}_{k\overline{o}}) + \mathbf{n}_{oj} && \text{(by (1) and (2))} \\ &= \mathbf{n}_{io} + \mathbf{n}_{o\overline{o}} + \mathbf{n}_{\overline{o}k} \\ &\quad + \mathbf{n}_{k\overline{o}} + \mathbf{n}_{\overline{o}o} + \mathbf{n}_{oj} \\ &= C^o(\mathbf{n})_{ik} + C^o(\mathbf{n})_{kj} && \text{(case hypothesis)} \end{aligned}$$

**Lemma 3.** *We prove now that, given a coherent DBM* $\mathbf{n}$ *such that* $\gamma^{Oct}(\mathbf{n}) \neq \emptyset$ *and an index* $k$*, we have—without any other hypothesis—*$\forall i \neq j,\ C^k(\mathbf{n})_{ij} \leq C^k(\mathbf{n})_{ik} + C^k(\mathbf{n})_{kj}$.

We have the same five different cases for the value of $C^k(\mathbf{n})_{ik}$ and the same five cases for the value of $C^k(\mathbf{n})_{kj}$ as in the preceding lemma, so we have the same 25 different cases to inspect.

In order to reduce the number of cases to elaborate on, observe that $\mathbf{n}_{kk} \geq 0$ and $\mathbf{n}_{k\bar{k}} + \mathbf{n}_{\bar{k}k} \geq 0$ because $\mathbf{n}$ has no strictly negative cycle. This means that, in fact, $C^k(\mathbf{n})_{ik} = \min(\mathbf{n}_{ik}, \mathbf{n}_{i\bar{k}} + \mathbf{n}_{\bar{k}k})$. Cases 2, 4 and 5 are not relevant for the value of $C^k(\mathbf{n})_{ik}$. A similar result holds for $C^k(\mathbf{n})_{kj}$ and we get $C^k(\mathbf{n})_{kj} = \min(\mathbf{n}_{kj}, \mathbf{n}_{k\bar{k}} + \mathbf{n}_{\bar{k}j})$.

This means that we only have four different cases to study:

*Case 1.* (1, 1).
We have:

$$\begin{array}{ll} C^k(\mathbf{n})_{ij} \leq \mathbf{n}_{ik} + \mathbf{n}_{kj} & \text{(by definition of } C^k(\mathbf{n})) \\ \quad = C^k(\mathbf{n})_{ik} + C^k(\mathbf{n})_{kj} & \text{(case hypothesis)} \end{array}$$

*Case 2.* (1, 3).
We have:

$$\begin{array}{ll} C^k(\mathbf{n})_{ij} \leq \mathbf{n}_{ik} + \mathbf{n}_{k\bar{k}} + \mathbf{n}_{\bar{k}j} & \text{(by definition of } C^k(\mathbf{n})) \\ \quad = C^k(\mathbf{n})_{ik} + C^k(\mathbf{n})_{kj} & \text{(case hypothesis)} \end{array}$$

*Case 3.* (3, 1).
We have:

$$\begin{array}{ll} C^k(\mathbf{n})_{ij} \leq \mathbf{n}_{i\bar{k}} + \mathbf{n}_{\bar{k}k} + \mathbf{n}_{kj} & \text{(by definition of } C^k(\mathbf{n})) \\ \quad = C^k(\mathbf{n})_{ik} + C^k(\mathbf{n})_{kj} & \text{(case hypothesis)} \end{array}$$

*Case 4.* (3, 3).
We have $\mathbf{n}_{k\bar{k}} + \mathbf{n}_{\bar{k}k} \geq 0$ (1), so:

$$\begin{array}{ll} C^k(\mathbf{n})_{ij} \leq \mathbf{n}_{i\bar{k}} + \mathbf{n}_{\bar{k}j} & \text{(by definition of } C^k(\mathbf{n})) \\ \quad \leq \mathbf{n}_{i\bar{k}} + (\mathbf{n}_{\bar{k}k} + \mathbf{n}_{k\bar{k}}) + \mathbf{n}_{\bar{k}j} & \text{(by (1))} \\ \quad = C^k(\mathbf{n})_{ik} + C^k(\mathbf{n})_{kj} & \text{(case hypothesis)} \end{array}$$

Now we use all three lemmas to prove by induction on $o$ the following property $\forall\, 1 \leq k \leq o,\ \forall i, j$:

$$\mathbf{m}^o_{ij} \leq \mathbf{m}^o_{i\,(2k-1)} + \mathbf{m}^o_{(2k-1)\,j} \quad \text{and} \quad \mathbf{m}^o_{ij} \leq \mathbf{m}^o_{i\,(2k)} + \mathbf{m}^o_{(2k)\,j}.$$

– The case $o = 0$ is obvious.
– Suppose the property is true for $o - 1 \geq 0$.
Using the second lemma with all $2k-1$ and $2k$, for all $k \leq o-1$, we obtain $\forall i \neq j$:

$$\begin{array}{l} (C^{2o-1}(\mathbf{m}^{o-1}))_{ij} \leq (C^{2o-1}(\mathbf{m}^{o-1}))_{i\,(2k-1)} + (C^{2o-1}(\mathbf{m}^{o-1}))_{(2k-1)\,j} \\ (C^{2o-1}(\mathbf{m}^{o-1}))_{ij} \leq (C^{2o-1}(\mathbf{m}^{o-1}))_{i\,(2k)} + (C^{2o-1}(\mathbf{m}^{o-1}))_{(2k)\,j}. \end{array}$$

Using the third lemma with $2o-1$ and $2o$ and the observation that $\forall \mathbf{m},\ \forall o,\ C^o(\mathbf{m}) = C^{\overline{o}}(\mathbf{m})$ we obtain $\forall i \neq j$:

$$(C^{2o-1}(\mathbf{m}^{o-1}))_{ij} \leq (C^{2o-1}(\mathbf{m}^{o-1}))_{i\,(2o-1)} + (C^{2o-1}(\mathbf{m}^{o-1}))_{(2o-1)\,j}$$
$$(C^{2o-1}(\mathbf{m}^{o-1}))_{ij} \leq (C^{2o-1}(\mathbf{m}^{o-1}))_{i\,(2o)} + (C^{2o-1}(\mathbf{m}^{o-1}))_{(2o)\,j}.$$

Recall that, by definition, $(C^{2k-1}(\mathbf{m}^{o-1}))_{ii} = 0$.
Obviously, $\gamma^{Oct}(C^{2k-1}(\mathbf{m}^{o-1})) = \gamma^{Oct}(\mathbf{m}) \neq \emptyset$, so, for all $k$, the cycle $\langle i, k, i\rangle$ has a positive weight which means that $\forall k$ :

$$(C^{2k-1}(\mathbf{m}^{o-1}))_{ii} = 0 \leq (C^{2o-1}(\mathbf{m}^{o-1}))_{ik} + (C^{2o-1}(\mathbf{m}^{o-1}))_{ki}$$

and we have $\forall k \leq o,\ \forall i, j$:

$$(C^{2o-1}(\mathbf{m}^{o-1}))_{ij} \leq (C^{2o-1}(\mathbf{m}^{o-1}))_{i\,(2k-1)} + (C^{2o-1}(\mathbf{m}^{o-1}))_{(2k-1)\,j}$$
$$(C^{2o-1}(\mathbf{m}^{o-1}))_{ij} \leq (C^{2o-1}(\mathbf{m}^{o-1}))_{i\,(2k)} + (C^{2o-1}(\mathbf{m}^{o-1}))_{(2k)\,j}.$$

Now, we use the first lemma to obtain $\forall k \leq o$ and $\forall i, j$:

$$\begin{aligned}(S(C^{2o-1}(\mathbf{m}^{o-1})))_{ij} \leq\ & (S(C^{2o-1}(\mathbf{m}^{o-1})))_{i\,(2k-1)} \\ & + (S(C^{2o-1}(\mathbf{m}^{o-1})))_{(2k-1)\,j}\end{aligned}$$

$$\begin{aligned}(S(C^{2o-1}(\mathbf{m}^{o-1})))_{ij} \leq\ & (S(C^{2o-1}(\mathbf{m}^{o-1})))_{i\,(2k)} \\ & + (S(C^{2o-1}(\mathbf{m}^{o-1})))_{(2k)\,j}.\end{aligned}$$

The property for $o = n$ settles the proof. □

**Theorem 6.** *$\gamma^{Oct}(\mathbf{m}) = \emptyset \iff \exists i,\ \mathbf{m}^n_{ii} < 0$, where $\mathbf{m}^n$ is defined as in Definition 2.*

(Stated in Section 3.4.)

**Proof:** We can prove by induction on $k$ that, for all $k \leq n$, $\gamma^{Oct}(\mathbf{m}^k) = \gamma^{Oct}(\mathbf{m})$. In particular, if $\exists i,\ \mathbf{m}^n_{ii} < 0$, then $\gamma^{Oct}(\mathbf{m}^n) = \emptyset$, and hence, $\gamma^{Oct}(\mathbf{m})$ is empty.

Suppose conversely that $\gamma^{Oct}(\mathbf{m}) = \emptyset$. By Theorem 2, we also have $\gamma^{Pot}(\mathbf{m}) = \emptyset$. We will denote by $\mathbf{m}'^k$ the DBM computed at the $k$-th step of the *regular* Floyd-Warshall algorithm. $\gamma^{Pot}(\mathbf{m}) = \emptyset$ implies that there is some $x$ such that $\mathbf{m}'^n_{xx} < 0$. Now, we can prove by induction on $k$ that $\forall i, j,\ \mathbf{m}^k_{ij} \leq \mathbf{m}'^k_{ij}$. As a consequence, $\mathbf{m}^n_{xx} < 0$, which concludes the proof. □

**Theorem 7.** *Thms. 3 and 4 are true on tightly closed DBMs.*

(Stated in Section 3.5.)

**Proof:** Suppose that $\mathbf{m}$ is tightly closed. We first prove the saturation property:

1. $\forall i, j,$ if $\mathbf{m}_{ij} < +\infty,$ then $\exists (v_1, \ldots, v_n) \in \gamma^{Oct}(\mathbf{m})$ such that $v'_j - v'_i = \mathbf{m}_{ij}$.
2. $\forall i, j,$ if $\mathbf{m}_{ij} = +\infty,$ then $\forall M < +\infty,\ \exists (v_1, \ldots, v_n) \in \gamma^{Oct}(\mathbf{m})$ such that $v'_j - v'_i \geq M$.

as follows:

1. Let $(i_0, j_0)$ be a pair. Let us consider a DBM $\mathbf{n}$ equal to $\mathbf{m}$ except that $\mathbf{n}_{j_0 i_0} \stackrel{\text{def}}{=} \mathbf{n}_{\overline{i_0}\,\overline{j_0}} \stackrel{\text{def}}{=} -\mathbf{m}_{i_0 j_0}$. Analogously to Theorem 3, we prove that $\gamma^{Oct}(\mathbf{n}) = \{(v_1, \ldots, v_n) \in \gamma^{Oct}(\mathbf{m}) \mid v'_{j_0} - v'_{i_0} = \mathbf{m}_{i_0 j_0}\}$, and hence, the proof of the desired property reduces to the proof that $\gamma^{Oct}(\mathbf{n}) \neq \emptyset$.
Suppose that $\gamma^{Oct}(\mathbf{n}) = \emptyset$. As $\mathbf{m}$ is tightly closed, $\mathbf{n}$ can be tightly closed by one application of the incremental tight closure algorithm, at position $(j_0, i_0)$. Let $\mathbf{n}''$ denote the matrix $Inc^T_{j_0 i_0}(\mathbf{n})$, and $\mathbf{n}'$ the intermediate matrix computed by Harvey and Stuckey's algorithm stated in Definition 4. By Theorem 2 from [33] (or, equivalently, Theorem 1 in [32]), we have $\exists i,\ \mathbf{n}''_{ii} < 0$. Several cases can occur, each one leading to a contradiction:

– Suppose that $\mathbf{n}''_{ii} = \mathbf{n}'_{ii}$. This means that $\min(\mathbf{n}_{ii},\ \mathbf{n}_{ij_0} + \mathbf{n}_{j_0 i_0} + \mathbf{n}_{i_0 i},\ \mathbf{n}_{i\overline{i_0}} + \mathbf{n}_{\overline{i_0}\,\overline{j_0}} + \mathbf{n}_{\overline{j_0} i}) < 0$. Thus, one of the three following cases occurs: either $0 > \mathbf{n}_{ii} = \mathbf{m}_{ii}$, $0 > \mathbf{n}_{ij_0} + \mathbf{n}_{j_0 i_0} + \mathbf{n}_{i_0 i} = \mathbf{m}_{ij_0} - \mathbf{m}_{i_0 j_0} + \mathbf{m}_{i_0 i}$, or $0 > \mathbf{n}_{i\overline{i_0}} + \mathbf{n}_{\overline{i_0}\,\overline{j_0}} + \mathbf{n}_{\overline{j_0} i} = \mathbf{m}_{i\overline{i_0}} - \mathbf{m}_{\overline{j_0}\,\overline{i_0}} + \mathbf{m}_{\overline{j_0} i}$. Each inequality contradicts the fact that $\mathbf{m}$ is closed.
– If $\mathbf{n}''_{ii} = (\mathbf{n}'_{i\bar{\imath}} + \mathbf{n}'_{\bar{\imath} i})/2$, there are many cases depending on the values of $\mathbf{n}'_{i\bar{\imath}}$ and $\mathbf{n}'_{\bar{\imath} i}$.

  Suppose that no tightening is used to derive $\mathbf{n}'_{i\bar{\imath}}$ nor $\mathbf{n}'_{\bar{\imath} i}$, that is, $\mathbf{n}'_{i\bar{\imath}} \in \{\mathbf{n}_{i\bar{\imath}},\ 2(\mathbf{n}_{j_0 i_0} + \mathbf{n}_{i\overline{i_0}} + (\mathbf{n}_{\overline{j_0} j_0}/2)),\ 2(\mathbf{n}_{j_0 i_0} + \mathbf{n}_{ij_0} + (\mathbf{n}_{i_0\overline{i_0}}/2)),\ \mathbf{n}_{ij_0} + \mathbf{n}_{j_0 i_0} + \mathbf{n}_{i_0\bar{\imath}}\}$ and $\mathbf{n}'_{\bar{\imath} i} \in \{\mathbf{n}_{\bar{\imath} i},\ 2(\mathbf{n}_{j_0 i_0} + \mathbf{n}_{\bar{\imath}\,\overline{i_0}} + (\mathbf{n}_{\overline{j_0} j_0}/2)),\ 2(\mathbf{n}_{j_0 i_0} + \mathbf{n}_{\bar{\imath} j_0} + (\mathbf{n}_{i_0\overline{i_0}}/2)),\ \mathbf{n}_{\bar{\imath} j_0} + \mathbf{n}_{j_0 i_0} + \mathbf{n}_{i_0 i}\}$.
  This can be rewritten as: $\mathbf{n}'_{i\bar{\imath}} \in \{\mathbf{n}_{i\bar{\imath}},\ \mathbf{n}_{i\overline{i_0}} + \mathbf{n}_{\overline{i_0}\,\overline{j_0}} + \mathbf{n}_{\overline{j_0} j_0} + \mathbf{n}_{j_0 i_0} + \mathbf{n}_{i_0\bar{\imath}},\ \mathbf{n}_{ij_0} + \mathbf{n}_{j_0 i_0} + \mathbf{n}_{i_0\overline{i_0}} + \mathbf{n}_{\overline{i_0}\,\overline{j_0}} + \mathbf{n}_{\overline{j_0}\bar{\imath}},\ \mathbf{n}_{ij_0} + \mathbf{n}_{j_0 i_0} + \mathbf{n}_{i_0\bar{\imath}}\}$ and $\mathbf{n}'_{\bar{\imath} i} \in \{\mathbf{n}_{\bar{\imath} i},\ \mathbf{n}_{\bar{\imath}\,\overline{i_0}} + \mathbf{n}_{\overline{i_0}\,\overline{j_0}} + \mathbf{n}_{\overline{j_0} j_0} + \mathbf{n}_{j_0 i_0} + \mathbf{n}_{i_0 i},\ \mathbf{n}_{\bar{\imath} j_0} + \mathbf{n}_{j_0 i_0} + \mathbf{n}_{i_0\overline{i_0}} + \mathbf{n}_{\overline{i_0}\,\overline{j_0}} + \mathbf{n}_{\overline{j_0} i},\ \mathbf{n}_{\bar{\imath} j_0} + \mathbf{n}_{j_0 i_0} + \mathbf{n}_{i_0 i}\}$. Then, $\mathbf{n}'_{i\bar{\imath}} + \mathbf{n}'_{\bar{\imath} i}$ can be expressed as the sum along a cycle in $\mathbf{m}$ that passes exactly zero times, once, or twice through $\mathbf{n}_{j_0 i_0} = -\mathbf{m}_{i_0 j_0}$. If it does not pass through $\mathbf{n}_{j_0 i_0}$, then we have a cycle in $\mathbf{m}$ with a strictly negative weight, which is impossible because $\gamma^{Oct}(\mathbf{m}) \neq \emptyset$. If it passes once, then $\mathbf{m}_{i_0 j_0}$ is strictly greater than the sum along a path from $i_0$ to $j_0$ in $\mathbf{m}$, which is also impossible because $\mathbf{m}$ is closed. If it passes twice, then $2\mathbf{m}_{i_0 j_0}$ is strictly greater than the sum along *two* paths from $i_0$ to $j_0$ in $\mathbf{m}$, which is only possible if $\mathbf{m}_{i_0 j_0}$ is strictly greater than the sum along at least one of these paths, which is also impossible.

  Suppose now that $\mathbf{n}_{ij_0} + \mathbf{n}_{j_0 i_0} + \mathbf{n}_{i_0\bar{\imath}} = 2k + 1$ is odd and, by tightening, $\mathbf{n}'_{i\bar{\imath}} = 2k$, but no tightening is involved in the computation of $\mathbf{n}'_{\bar{\imath} i}$. As a first sub-case, suppose that $\mathbf{n}'_{\bar{\imath} i} = \mathbf{n}_{\bar{\imath} i}$. Then, we have $(\mathbf{n}_{ij_0} + \mathbf{n}_{j_0 i_0} + \mathbf{n}_{i_0\bar{\imath}} - 1) + \mathbf{n}_{\bar{\imath} i} < 0$, that is, $\mathbf{m}_{i_0 j_0} + 1 > \mathbf{m}_{i_0\bar{\imath}} + \mathbf{m}_{\bar{\imath} i} + \mathbf{m}_{ij_0}$. As $\mathbf{m}_{i_0 j_0} \leq \mathbf{m}_{i_0\bar{\imath}} + \mathbf{m}_{\bar{\imath} i} + \mathbf{m}_{ij_0}$, by closure of $\mathbf{m}$, we must have $\mathbf{m}_{i_0 j_0} = \mathbf{m}_{i_0\bar{\imath}} + \mathbf{m}_{\bar{\imath} i} + \mathbf{m}_{ij_0}$. By hypothesis, $2k + 1 = \mathbf{m}_{ij_0} - \mathbf{m}_{i_0 j_0} + \mathbf{m}_{i_0\bar{\imath}}$, and hence, $\mathbf{m}_{\bar{\imath} i} = -2k - 1$. This is impossible. By tightness of $\mathbf{m}$, $\mathbf{m}_{\bar{\imath} i}$ cannot be odd. Our second sub-case is: $\mathbf{n}'_{\bar{\imath} i} \neq \mathbf{n}_{\bar{\imath} i}$. Then, $\mathbf{n}'_{i\bar{\imath}} + \mathbf{n}'_{\bar{\imath} i}$ can be expressed as the sum *minus one* along a cycle in $\mathbf{n}$ that passes exactly twice through $\mathbf{n}_{j_0 i_0} = -\mathbf{m}_{i_0 j_0}$. Thus, $2\mathbf{m}_{i_0 j_0} + 1$ is strictly greater than the sum along two paths from $i_0$ to $j_0$. Moreover, this sum along the two paths is *odd*, and hence, the weight of the these two paths cannot be the same and $2\mathbf{m}_{i_0 j_0}$ is strictly greater than twice the weight of the path with smallest weight. We thus have proved that $\mathbf{m}_{i_0 j_0}$ is strictly greater than the weight of a path from $i_0$ to $j_0$ in $\mathbf{m}$, which is impossible. The situation where tightening is used for $\mathbf{n}'_{\bar{\imath} i}$ but not for $\mathbf{n}'_{i\bar{\imath}}$ is similar.
  For our last case, we suppose that tightening is used in both $\mathbf{n}'_{i\bar{\imath}}$ and $\mathbf{n}'_{\bar{\imath} i}$, that is, $\mathbf{n}'_{i\bar{\imath}} = 2k$ and $\mathbf{n}'_{\bar{\imath} i} = 2l$ where $\mathbf{n}_{ij_0} + \mathbf{n}_{j_0 i_0} + \mathbf{n}_{i_0\bar{\imath}} = 2k + 1$ and $\mathbf{n}_{\bar{\imath} j_0} + \mathbf{n}_{j_0 i_0} + \mathbf{n}_{i_0 i} = 2l + 1$. This means, in particular, that $\mathbf{m}_{i_0 j_0} = \mathbf{m}_{ij_0} + \mathbf{m}_{i_0\bar{\imath}} - (2k + 1) = \mathbf{m}_{\bar{\imath} j_0} + \mathbf{m}_{i_0 i} - (2l + 1)$, that is, $\mathbf{m}_{ij_0} + \mathbf{m}_{i_0\bar{\imath}}$ and $\mathbf{m}_{\bar{\imath} j_0} + \mathbf{m}_{i_0 i}$ are either both odd, or both even. Our hypothesis $\mathbf{n}'_{i\bar{\imath}} + \mathbf{n}'_{\bar{\imath} i} < 0$ can be rewritten as: $2\mathbf{m}_{i_0 j_0} + 2 > \mathbf{m}_{ij_0} + \mathbf{m}_{i_0\bar{\imath}} + \mathbf{m}_{\bar{\imath} j_0} + \mathbf{m}_{i_0 i}$. As,

by closure, $2\mathbf{m}_{i_0 j_0} \leq \mathbf{m}_{i j_0} + \mathbf{m}_{i_0 \bar{\imath}} + \mathbf{m}_{\bar{\imath} j_0} + \mathbf{m}_{i_0 i}$ and $\mathbf{m}_{i j_0} + \mathbf{m}_{i_0 \bar{\imath}} + \mathbf{m}_{\bar{\imath} j_0} + \mathbf{m}_{i_0 i}$ is even, we have $2\mathbf{m}_{i_0 j_0} = \mathbf{m}_{i j_0} + \mathbf{m}_{i_0 \bar{\imath}} + \mathbf{m}_{\bar{\imath} j_0} + \mathbf{m}_{i_0 i}$. If we had $\mathbf{m}_{i_0 i} + \mathbf{m}_{i j_0} \neq \mathbf{m}_{i_0 \bar{\imath}} + \mathbf{m}_{\bar{\imath} j_0}$, we would have $\mathbf{m}_{i_0 j_0} > \min(\mathbf{m}_{i_0 i} + \mathbf{m}_{i j_0}, \mathbf{m}_{i_0 \bar{\imath}} + \mathbf{m}_{\bar{\imath} j_0})$, which is impossible because $\mathbf{m}$ is closed. We can now suppose that $\mathbf{m}_{i_0 i} + \mathbf{m}_{i j_0} = \mathbf{m}_{i_0 \bar{\imath}} + \mathbf{m}_{\bar{\imath} j_0}$. This implies that $\mathbf{m}_{i_0 j_0} = \mathbf{m}_{i_0 i} + \mathbf{m}_{i j_0} = \mathbf{m}_{i_0 \bar{\imath}} + \mathbf{m}_{\bar{\imath} j_0}$. On the one hand, $(2k+1) = \mathbf{m}_{i j_0} - \mathbf{m}_{i_0 j_0} + \mathbf{m}_{i_0 \bar{\imath}} = \mathbf{m}_{i_0 \bar{\imath}} - \mathbf{m}_{i_0 i}$. On the other hand, $(2l+1) = \mathbf{m}_{\bar{\imath} j_0} - \mathbf{m}_{i_0 j_0} + \mathbf{m}_{i_0 i} = \mathbf{m}_{i_0 i} - \mathbf{m}_{i_0 \bar{\imath}}$. So, $k = -l$ and $\mathbf{n}'_{i\bar{\imath}} + \mathbf{n}'_{\bar{\imath} i} = 2(k+l) = 0$, which is in contradiction with $\mathbf{n}'_{i\bar{\imath}} + \mathbf{n}'_{\bar{\imath} i} < 0$.

2. The second point can be proved almost as the first one, except that $\mathbf{n}$ is constructed by changing $\mathbf{m}$'s elements $(j_0, i_0)$ and $(\overline{\imath_0}, \overline{j_0})$ into $\mathbf{n}_{j_0 i_0} \stackrel{\text{def}}{=} \mathbf{n}_{\overline{\imath_0}\,\overline{j_0}} \stackrel{\text{def}}{=} \min(\mathbf{m}_{j_0 i_0}, -M)$. Analogously, $\gamma^{Oct}(\mathbf{n}) = \{(v_1, \ldots, v_n) \in \gamma^{Oct}(\mathbf{m}) \mid v'_{j_0} - v'_{i_0} \geq M\}$ and $\gamma^{Oct}(\mathbf{m}) \neq \emptyset$.

As for Theorem 4, the normal form property is a consequence of the saturation property. □

**Theorem 8.** $\mathbf{m}^\bullet = \mathbf{n}^\bullet \iff \gamma^{Oct}(\mathbf{m}) = \gamma^{Oct}(\mathbf{n})$.

(Stated in Section 3.7.)

**Proof:** The $\Longrightarrow$ part is a consequence of the fact that $\forall \mathbf{m},\ \gamma^{Oct}(\mathbf{m}^\bullet) = \gamma^{Oct}(\mathbf{m})$. To obtain the $\Longleftarrow$ part, we first apply $\alpha^{Oct}$ to obtain $(\alpha^{Oct} \circ \gamma^{Oct})(\mathbf{m}) = (\alpha^{Oct} \circ \gamma^{Oct})(\mathbf{n})$, and then, use the fact that $\forall \mathbf{m} \in \mathsf{DBM},\ \mathbf{m}^\bullet = (\alpha^{Oct} \circ \gamma^{Oct})(\mathbf{m})$ (Theorem 4). □

**Theorem 9.** $\mathbf{m}^\bullet \sqsubseteq^{\mathsf{DBM}} \mathbf{n} \iff \gamma^{Oct}(\mathbf{m}) \subseteq \gamma^{Oct}(\mathbf{n})$.

(Stated in Section 3.7.)

**Proof:** The $\Longrightarrow$ part is a consequence of the fact that $\forall \mathbf{m},\ \gamma^{Oct}(\mathbf{m}^\bullet) = \gamma^{Oct}(\mathbf{m})$ and the monotonicity of $\gamma^{Oct}$. To obtain the $\Longleftarrow$ part, we first apply the monotonic function $\alpha^{Oct}$ to obtain $(\alpha^{Oct} \circ \gamma^{Oct})(\mathbf{m}) \sqsubseteq^{\mathsf{DBM}} (\alpha^{Oct} \circ \gamma^{Oct})(\mathbf{n})$, and then, use the fact that $\forall \mathbf{m} \in \mathsf{DBM},\ \mathbf{m}^\bullet = (\alpha^{Oct} \circ \gamma^{Oct})(\mathbf{m})$, proved by Theorem 4, to obtain $\mathbf{m}^\bullet \sqsubseteq^{\mathsf{DBM}} \mathbf{n}^\bullet$. We conclude by remarking that $\mathbf{n}^\bullet \sqsubseteq^{\mathsf{DBM}} \mathbf{n}$ always holds. □

**Theorem 10.** $\gamma^{Oct}(\mathbf{m} \cup^{Oct} \mathbf{n}) = \inf_{\subseteq} \{S \in Oct \mid S \supseteq \gamma^{Oct}(\mathbf{m}) \cup \gamma^{Oct}(\mathbf{n})\}$.

(Stated in Section 4.1.)

**Proof:** We first prove that $\mathbf{m} \cup^{Oct} \mathbf{n} = \inf_{\sqsubseteq^{\mathsf{DBM}}} \{\mathbf{o} \mid \gamma^{Oct}(\mathbf{o}) \supseteq \gamma^{Oct}(\mathbf{m}) \cup \gamma^{Oct}(\mathbf{n})\}$, which is a stronger result.

Whenever $\gamma^{Oct}(\mathbf{m}) = \emptyset$, $\mathbf{m}^\bullet = \bot^{\mathsf{DBM}}$ and the property is obvious. The same holds when $\gamma^{Oct}(\mathbf{n}) = \emptyset$.

We now suppose that $\gamma^{Oct}(\mathbf{m}),\ \gamma^{Oct}(\mathbf{n}) \neq \emptyset$. Using $\forall \mathbf{m},\ \gamma^{Oct}(\mathbf{m}^\bullet) = \gamma^{Oct}(\mathbf{m})$ and the monotonicity of $\gamma^{Oct}$, we get that $\mathbf{m} \cup^{Oct} \mathbf{n}$ effectively over-approximates $\gamma^{Oct}(\mathbf{m}) \cup \gamma^{Oct}(\mathbf{n})$ because $\gamma^{Oct}(\mathbf{m}) \cup \gamma^{Oct}(\mathbf{n}) = \gamma^{Oct}(\mathbf{m}^\bullet) \cup \gamma^{Oct}(\mathbf{n}^\bullet) \subseteq \gamma^{Oct}(\mathbf{m}^\bullet \sqcup^{\mathsf{DBM}} \mathbf{n}^\bullet) = \gamma^{Oct}(\mathbf{m} \cup^{Oct} \mathbf{n})$.

We now suppose that $\gamma^{Oct}(\mathbf{o}) \supseteq \gamma^{Oct}(\mathbf{m}) \cup \gamma^{Oct}(\mathbf{n})$ and prove that $\mathbf{m} \cup^{Oct} \mathbf{n} \sqsubseteq^{\mathsf{DBM}} \mathbf{o}$. Let $i_0$ and $j_0$ be two indices such that $\mathbf{m}^\bullet_{i_0 j_0},\ \mathbf{n}^\bullet_{i_0 j_0} < +\infty$. Using the saturation property of the strong closure (Theorem 3) we can find two points $\vec{v}^{\,\mathbf{m}} \in \gamma^{Oct}(\mathbf{m})$ and $\vec{v}^{\,\mathbf{n}} \in \gamma^{Oct}(\mathbf{n})$ such that $v'^{\mathbf{m}}_{j_0} - v'^{\mathbf{m}}_{i_0} = \mathbf{m}^\bullet_{i_0 j_0}$ and $v'^{\mathbf{n}}_{j_0} - v'^{\mathbf{n}}_{i_0} = \mathbf{n}^\bullet_{i_0 j_0}$, where primed versions of coordinates are defined, as before, as $x'_{2i-1} = -x'_{2i} = x_i$. As both $\vec{v}^{\,\mathbf{m}}$ and $\vec{v}^{\,\mathbf{n}}$ are also in $\gamma^{Oct}(\mathbf{o})$, we have $v'^{\mathbf{m}}_{j_0} - v'^{\mathbf{m}}_{i_0} \leq \mathbf{o}_{i_0 j_0}$ and $v'^{\mathbf{n}}_{j_0} - v'^{\mathbf{n}}_{i_0} \leq \mathbf{o}_{i_0 j_0}$, which means that $\mathbf{o}_{i_0 j_0} \geq \max(\mathbf{m}^\bullet_{i_0 j_0}, \mathbf{n}^\bullet_{i_0 j_0}) =$

$(\mathbf{m} \cup^{Oct} \mathbf{n})_{i_0 j_0}$. Whenever $\mathbf{m}^{\bullet}_{i_0 j_0} = +\infty$ or $\mathbf{n}^{\bullet}_{i_0 j_0} = +\infty$, the same reasoning allows proving that $\mathbf{o}_{i_0 j_0} \geq M$ for every $M$, that is $\mathbf{o}_{i_0 j_0} = +\infty = (\mathbf{m} \cup^{Oct} \mathbf{n})_{i_0 j_0}$. Because $\gamma^{Oct}$ is a complete $\sqcap^{\mathsf{DBM}}-$morphism, $\mathbf{m} \cup^{Oct} \mathbf{n} = \inf_{\sqsubseteq^{\mathsf{DBM}}} \{\mathbf{o} \mid \gamma^{Oct}(\mathbf{o}) \supseteq \gamma^{Oct}(\mathbf{m}) \cup \gamma^{Oct}(\mathbf{n})\}$ implies $\gamma^{Oct}(\mathbf{m} \cup^{Oct} \mathbf{n}) = \inf_{\subseteq} \{S \in Oct \mid S \supseteq \gamma^{Oct}(\mathbf{m}) \cup \gamma^{Oct}(\mathbf{n})\}$. □

**Theorem 11.** $\mathbf{m} \cup^{Oct} \mathbf{n}$ *is strongly closed.*

(Stated in Section 4.1.)

**Proof:** This is a direct consequence of $\mathbf{m} \cup^{Oct} \mathbf{n} = \inf_{\sqsubseteq^{\mathsf{DBM}}} \{\mathbf{o} \mid \gamma^{Oct}(\mathbf{o}) \supseteq \gamma^{Oct}(\mathbf{m}) \cup \gamma^{Oct}(\mathbf{n})\}$ which was proved in the previous theorem: $\mathbf{m} \cup^{Oct} \mathbf{n}$ is indeed the smallest of all DBMs representing $\gamma^{Oct}(\mathbf{m} \cup^{Oct} \mathbf{n})$. □

**Theorem 12.** $\gamma^{Oct}(\mathbf{m} \cap^{Oct} \mathbf{n}) = \gamma^{Oct}(\mathbf{m}) \cap \gamma^{Oct}(\mathbf{n})$.

(Stated in Section 4.1.)

**Proof:** This is a consequence of the fact that $\gamma^{Oct}$ is a complete $\sqcap^{\mathsf{DBM}}$-morphism. □

**Theorem 13.** $\gamma^{Oct}(\{\!|\, V_f \leftarrow ?\, |\!\}^{Oct}(\mathbf{m})) \supseteq \{\!|\, V_f \leftarrow ?\, |\!\}(\gamma^{Oct}(\mathbf{m}))$.

(Stated in Section 4.2.)

**Proof:** The property to prove can be restated as $\gamma^{Oct}(\{\!|\, V_f \leftarrow ?\, |\!\}^{Oct}(\mathbf{m})) \supseteq \{\vec{v} \in \mathbb{I}^n \mid \exists t \in \mathbb{I},\ \vec{v}[V_f \mapsto t] \in \gamma^{Oct}(\mathbf{m})\}$. Let us take $t \in \mathbb{I}$ and $\vec{v} = (v_1, \ldots, v_n) \in \gamma^{Oct}(\mathbf{m})$. We want to prove that $\vec{w} = (v_1, \ldots, v_{f-1}, t, v_{f+1}, \ldots, v_n) \in \gamma^{Oct}(\{\!|\, V_f \leftarrow ?\, |\!\}^{Oct}(\mathbf{m}))$, that is to say, $\forall i, j,\ w'_j - w'_i \leq (\{\!|\, V_f \leftarrow ?\, |\!\}^{Oct}(\mathbf{m}))_{ij}$, denoting by $w_i$ the $i$-th coordinate of $\vec{w}$ and defining the primed coordinate versions as usual: $w'_{2i-1} = -w'_{2i} = w_i$.

- If $i, j \notin \{2f, 2f-1\}$, we have $w'_j = v'_j$ and $w'_i = v'_i$, so, $w'_j - w'_i = v'_j - v'_i \leq \mathbf{m}_{ij} = (\{\!|\, V_f \leftarrow ?\, |\!\}^{Oct}(\mathbf{m}))_{ij}$.
- If $i$ or $j$ is in $\{2f-1,\ 2f\}$, but not both, then $w'_j - w'_i \leq +\infty = (\{\!|\, V_f \leftarrow ?\, |\!\}^{Oct}(\mathbf{m}))_{ij}$.
- Finally, if $i = j \in \{2f-1,\ 2f\}$, $w'_j - w'_i = 0 = (\{\!|\, V_f \leftarrow ?\, |\!\}^{Oct}(\mathbf{m}))_{ij}$.

□

**Theorem 14.** $\gamma^{Oct}(\{\!|\, V_f \leftarrow ?\, |\!\}^{Oct}(\mathbf{m}^{\bullet})) = \{\!|\, V_f \leftarrow ?\, |\!\}(\gamma^{Oct}(\mathbf{m}))$.

(Stated in Section 4.2.)

**Proof:** First, the property is obvious if $\mathbf{m}^{\bullet} = \bot^{\mathsf{DBM}}$, that is, $\gamma^{Oct}(\mathbf{m}) = \emptyset$, so, we will consider the case where $\mathbf{m}^{\bullet} \neq \bot^{\mathsf{DBM}}$. Using the preceding theorem and the fact that $\gamma^{Oct}(\mathbf{m}^{\bullet}) = \gamma^{Oct}(\mathbf{m})$, we get the first part of the equality: $\gamma^{Oct}(\{\!|\, V_f \leftarrow ?\, |\!\}^{Oct}(\mathbf{m}^{\bullet})) \supseteq \{\vec{v} \in \mathbb{I}^n \mid \exists t \in \mathbb{I},\ \vec{v}[V_f \mapsto t] \in \gamma^{Oct}(\mathbf{m}^{\bullet})\} = \{\vec{v} \in \mathbb{I}^n \mid \exists t \in \mathbb{I},\ \vec{v}[V_f \mapsto t] \in \gamma^{Oct}(\mathbf{m})\}$.

For the converse inclusion, let us take $\vec{v} = (v_1, \ldots, v_n) \in \gamma^{Oct}(\{\!|\, V_f \leftarrow ?\, |\!\}^{Oct}(\mathbf{m}^{\bullet}))$. We want to prove that there exists a $t$ such that $\vec{v}[V_f \mapsto t] \in \gamma^{Oct}(\mathbf{m})$. We first prove that:

$$\max\ \left\{v'_j - \mathbf{m}^{\bullet}_{(2f-1)\,j},\ -\mathbf{m}^{\bullet}_{(2f-1)(2f)}/2 \mid j \neq 2f-1,\ 2f\right\}$$
$$\leq \min\ \left\{\mathbf{m}^{\bullet}_{i\,(2f-1)} + v'_i,\ \mathbf{m}^{\bullet}_{(2f-1)(2f)}/2 \mid i \neq 2f-1,\ 2f\right\}$$

where, as usual, $v'_{2i-1} = -v'_{2i} = v_i$. Suppose that this is not true. This may be for one of the following reasons, each one of them leading to a contradiction:

- If $\exists i, j \notin \{2f-1, 2f\}$ such that $v'_j - \mathbf{m}^\bullet_{(2f-1)\,j} > \mathbf{m}^\bullet_{i\,(2f-1)} + v'_i$, then $v'_j - v'_i > \mathbf{m}^\bullet_{i\,(2f-1)} + \mathbf{m}^\bullet_{(2f-1)\,j} \geq \mathbf{m}^\bullet_{ij}$ by closure of $\mathbf{m}^\bullet$. This contradicts the fact that $v'_j - v'_i \leq (\!|\, V_f \leftarrow ?\,|\!)^{Oct}(\mathbf{m}^\bullet))_{ij} = \mathbf{m}^\bullet_{ij}$ when $i, j \notin \{2f-1, 2f\}$.
- If $\exists j \notin \{2f-1, 2f\}$ such that $v'_j - \mathbf{m}^\bullet_{(2f-1)\,j} > \mathbf{m}^\bullet_{(2f)\,(2f-1)}/2$, then $2v'_j > \mathbf{m}^\bullet_{(2f)\,(2f-1)} + 2\mathbf{m}^\bullet_{(2f-1)\,j} = \mathbf{m}^\bullet_{j\,(2f)} + \mathbf{m}^\bullet_{(2f)\,(2f-1)} + \mathbf{m}^\bullet_{(2f-1)\,j} \geq \mathbf{m}^\bullet_{j\,j}$ by closure of $\mathbf{m}^\bullet$. This is also impossible.
- A similar situation is when $\exists i \notin \{2f-1, 2f\}$ such that $\mathbf{m}^\bullet_{i\,(2f-1)} + v'_i > \mathbf{m}^\bullet_{(2f-1)\,(2f)}/2$.
- The last possibility is to have $-\mathbf{m}^\bullet_{(2f-1)\,(2f)}/2 > \mathbf{m}^\bullet_{(2f)\,(2f-1)}/2$, which would imply $\mathbf{m}^\bullet_{(2f)\,(2f-1)} + \mathbf{m}^\bullet_{(2f-1)\,(2f)} < 0$, and hence, $\gamma^{Oct}(\mathbf{m}) = \emptyset$, which contradicts our hypothesis.

Hence, there exists at least one $t \in \mathbb{I}$ such that:

$$\begin{array}{rcl} \max\limits_{j \neq 2f-1,2f} (v'_j - \mathbf{m}^\bullet_{(2f-1)\,j}) \leq t & \leq & \min\limits_{i \neq 2f-1,2f} (v'_i + \mathbf{m}^\bullet_{i\,(2f-1)}) \\ -\mathbf{m}^\bullet_{(2f-1)\,(2f)}/2 \leq t & \leq & \mathbf{m}^\bullet_{(2f)\,(2f-1)}/2 \end{array}$$

We now prove that any such t is a good choice, i.e., $\vec{v}[V_f \mapsto t] \in \gamma^{Oct}(\mathbf{m})$. We will denote $(v_1, -v_1, \ldots, v_{f-1}, -v_{f-1}, t, -t, v_{f+1}, -v_{f+1}, \ldots, v_n, -v_n)$ by $\vec{w}\,'$, and by $w'_k$ its $k$-th coordinate. We only need to prove that $\forall i, j,\ w'_j - w'_i \leq \mathbf{m}^\bullet_{ij}$:

- If $i \notin \{2f-1, 2f\}$ and $j \notin \{2f-1, 2f\}$, then $w'_j - w'_i = v'_j - v'_i \leq (\!|\, V_f \leftarrow ?\,|\!)^{Oct}(\mathbf{m}^\bullet))_{ij} = \mathbf{m}^\bullet_{ij}$.
- If $i = j = 2f-1$ or $i = j = 2f$, then $w'_j - w'_i = 0 = \mathbf{m}^\bullet_{ij}$.
- If $i = 2f-1$ and $j \notin \{2f-1, 2f\}$, then $w'_j - w'_i = v'_j - t \leq \mathbf{m}^\bullet_{ij}$ because $t \geq \max_{j \neq 2f-1,2f} (v'_j - \mathbf{m}^\bullet_{(2f-1)\,j})$.
- If $j = 2f-1$ and $i \notin \{2f-1, 2f\}$, then $w'_j - w'_i = t - v'_i \leq \mathbf{m}^\bullet_{ij}$ because $t \leq \min_{i \neq 2f-1,2f} (v'_i + \mathbf{m}^\bullet_{i\,(2f)})$.
- If $i = 2f$ and $j \notin \{2f-1, 2f\}$, then $w'_j - w'_i = v'_j + t \leq \mathbf{m}^\bullet_{j\,(2f-1)} = \mathbf{m}^\bullet_{ij}$ using the case where $i = 2f-1$ and the coherence.
- If $j = 2f$ and $i \notin \{2f-1, 2f\}$, then $w'_j - w'_i = -t - v'_i \leq \mathbf{m}^\bullet_{(2f-1)\,\overline{i}} = \mathbf{m}^\bullet_{ij}$ using the case where $j = 2f-1$ and the coherence.
- If $j = \overline{i} = 2f-1$, then $w'_j - w'_i = 2t \leq \mathbf{m}^\bullet_{(2f)\,(2f-1)}$ by definition of $t$. The case $i = \overline{j} = 2f-1$ is similar.

□

**Theorem 15.** $(\!|\, V_f \leftarrow ?\,|\!)^{Oct}(\mathbf{m})$ *is strongly closed whenever* $\mathbf{m}$ *is.*

(Stated in Section 4.2.)

**Proof:** Suppose that $\mathbf{m}$ is strongly closed and let $\mathbf{m}'$ denote the matrix $(\!|\, V_f \leftarrow ?\,|\!)^{Oct}(\mathbf{m})$. By Definition 1, we must prove three properties on $\mathbf{m}'$:

- We have easily $\forall i,\ \mathbf{m}'_{ii} = 0$. When $i \in \{2f, 2f-1\}$, this is enforced by the definition of $\mathbf{m}'$ and otherwise we have $\mathbf{m}'_{ii} = \mathbf{m}_{ii}$ which also equals 0 because $\mathbf{m}$ is itself strongly closed.
- Let $i$, $j$, and $k$ be three variables. If $i$, $j$, and $k$ are all different from $2f-1$ and $2f$, then $\mathbf{m}'_{ij} = \mathbf{m}_{ij} \leq \mathbf{m}_{ik} + \mathbf{m}_{kj} = \mathbf{m}'_{ik} + \mathbf{m}'_{kj}$. If $i$, $j$, and $k$ are all in $\{2f, 2f-1\}$, then

$\mathbf{m}'_{ij} = \mathbf{m}'_{ik} + \mathbf{m}'_{kj} = 0$. In all other cases, at least one of $\mathbf{m}'_{ik}$ and $\mathbf{m}'_{kj}$ is $+\infty$, and hence, $\mathbf{m}'_{ij} \leq \mathbf{m}'_{ik} + \mathbf{m}'_{kj} = +\infty$.

– Finally, we prove that $\forall i, j,\ \mathbf{m}'_{ij} \leq (\mathbf{m}'_{i\bar{\imath}} + \mathbf{m}'_{\bar{\jmath}j})/2$. If $i \neq 2f-1, 2f$ and $j \neq 2f-1, 2f$, then $\mathbf{m}'_{ij} = \mathbf{m}_{ij}$, $\mathbf{m}'_{i\bar{\imath}} = \mathbf{m}_{i\bar{\imath}}$ and $\mathbf{m}'_{\bar{\jmath}j} = \mathbf{m}_{\bar{\jmath}j}$, so, the property is a consequence of $\mathbf{m}$ being strongly closed. If $i = 2f-1$ or $i = 2f$ or $j = 2f-1$ or $j = 2f$, then at least one of $\mathbf{m}'_{i\bar{\imath}}$ and $\mathbf{m}'_{\bar{\jmath}j}$ is $+\infty$, so $(\mathbf{m}'_{i\bar{\imath}} + \mathbf{m}'_{\bar{\jmath}j})/2 = +\infty \geq \mathbf{m}'_{ij}$. □

**Theorem 16.** $\pi_i(\mathbf{m}) = \{v \in \mathbb{I} \mid \exists (v_1, \ldots, v_n) \in \gamma^{Oct}(\mathbf{m}),\ v_i = v\}$.

(Stated in Section 4.3.)

**Proof:** This is an easy consequence of the saturation property of $\bullet$ (Theorem 3). □

**Theorem 17.** *$\triangledown^{Oct}_{std}$ and $\triangledown^{Oct}_{th}$ are indeed widenings.*

(Stated in Section 4.7.)

**Proof:** First note that $\triangledown^{Oct}_{std}$ is a special case of $\triangledown^{Oct}_{th}$ where $\mathbb{T} = \emptyset$, so, we will only provide a proof for $\triangledown^{Oct}_{th}$.

The fact that $\mathbf{m}, \mathbf{n} \sqsubseteq^{\mathsf{DBM}} \mathbf{m} \triangledown^{Oct}_{th} \mathbf{n}$ is quite obvious. Consider a sequence $(\mathbf{m}^k)_{k\in\mathbb{N}}$ defined by $\mathbf{m}^{k+1} \stackrel{\text{def}}{=} \mathbf{m}^k \triangledown^{Oct}_{th} \mathbf{n}^{k+1}$. We can prove by induction on $k$ that, for each matrix position $(i, j)$, $i \neq \overline{j}$, $\mathbf{m}^k_{ij}$ can only take values in the set $\mathbb{T} \cup \{+\infty, \mathbf{m}^0_{ij}\}$, which is finite. A similar property holds for elements at positions $(i, \overline{i})$. As a consequence, $\mathbf{m}^k$ can only take a value within a finite set of matrices and, as it is an increasing sequence, it must be stable after some finite $k$. □

**Theorem 18.** *$\triangle^{Oct}_{std}$ is indeed a narrowing.*

(Stated in Section 4.7.)

**Proof:** We obviously have $\mathbf{m} \sqcap^{\mathsf{DBM}} \mathbf{n} \sqsubseteq^{\mathsf{DBM}} \mathbf{m} \triangle^{Oct}_{std} \mathbf{n} \sqsubseteq^{\mathsf{DBM}} \mathbf{m}$. Consider a sequence defined by $\mathbf{m}^{k+1} \stackrel{\text{def}}{=} \mathbf{m}^k \triangle^{Oct}_{std} \mathbf{n}^{k+1}$. Consider the set $S^k$ of matrix positions $(i, j)$ such that $\mathbf{m}^k_{ij} = +\infty$. A consequence of the definition of $\triangle^{Oct}_{std}$ is that $S^k$ is decreasing for $\subseteq$. So, there exists some $k$ such that $S^k = S^{k+1}$. For such a $k$, whenever $\mathbf{m}^k_{ij} = +\infty$, we also have $\mathbf{m}^{k+1}_{ij} = +\infty$. If $\mathbf{m}^k_{ij} \neq +\infty$, then, by definition of $\triangle^{Oct}_{std}$, we have $\mathbf{m}^{k+1}_{ij} = \mathbf{m}^k_{ij}$. We thus have proved that $\mathbf{m}^{k+1} = \mathbf{m}^k$ for this $k$. □

**Acknowledgments** I would like to thank the former and present members of the ASTRÉE team: B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, D. Monniaux, and X. Rival. I would also like to thank the anonymous referees as well as Olivier Danvy for their insightful comments on the present article.

## References

1. ACI Sécurité & Informatique. Analyse de PROgrammes Numériques. `http://www.cri.ensmp.fr/apron/`
2. Airbus. `http://www.airbus.com/`

3. Astrée. Analyse Statique de logiciels Temps-RÉel embarqués (static analysis of critical real-time embedded software) analyzer page. http://www.astree.ens.fr/
4. Bagnara, R.: Data-flow analysis for constraint logic-based languages. PhD thesis, Dipartimento di Informatica, Università di Pisa, Corso Italia 40, I-56125 Pisa, Italy (1997)
5. Bagnara, R., Hill, P.M., Mazzi, E., Zaffanella, E.: Widening operators for weakly-relational numeric abstractions. Quaderno 399, Dipartimento di Matematica, Università di Parma, Italy (2005)
6. Balasundaram, V., Kennedy, K.: A technique for summarizing data access and its use in parallelism enhancing transformations. In: ACM PLDI'89, ACM Press, pp. 41–53. (1989)
7. Bellman, R.: On a routing problem. In: Quarterly of Applied Mathematics, vol. **16**, pp. 87–90 (1958)
8. Blanchet, B., Cousot, P., Cousot, R., Feret, J., Mauborgne, L., A., Miné, Monniaux, D., Rival, X.: Design and implementation of a special-purpose static program analyzer for safety-critical real-time embedded software, invited chapter. In: The Essence of Computation: Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones, LNCS, Springer, pp. 85–108. (2002)
9. Blanchet, B., Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., Rival, X.: A static analyzer for large safety-critical software. In: ACM PLDI'03, vol. 548030, pp. 196–207, ACM Press, (2003)
10. Bourdoncle, F.: Abstract interpretation by dynamic partitioning. Journal of Functional Programming **2**(4), 407–423 (1992)
11. Bourdoncle, F.: Abstract debugging of higher-order imperative languages. In: ACM PLDI'93, pp. 46–55. ACM Press, (1993)
12. Clarisó, R., Cortadella, J.: The octahedron abstract domain. In: SAS'04, vol. 3148 of LNCS, pp. 312–327. Springer (2004)
13. Colón, M.A., Sipma, H.B.: Synthesis of linear ranking functions. In: TACAS'01, vol. 2031 of LNCS, pp. 67–81 (2001)
14. Cormen, T., Leiserson, C., Rivest, R.: Introduction to Algorithms. The MIT Press (1990)
15. Cousot, P.: The calculational design of a generic abstract interpreter. In: Calculational System Design, NATO ASI Series F. IOS Press (1999)
16. Cousot, P.: Verification by abstract interpretation. In: Proc. Int. Symp. on Verification—Theory & Practice—Honoring Zohar Manna's 64th Birthday, vol. 2772, pp. 243–268. Springer (2003)
17. Cousot, P., Cousot, R.: Static determination of dynamic properties of programs. In ISOP'76, pp. 106–130. Dunod, Paris, France (1976)
18. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: ACM POPL'77, pp. 238–252. ACM Press (1977)
19. Cousot, P., Cousot, R.: Abstract interpretation and application to logic programs. Journal of Logic Programming **13**(2–3), 103–179 (1992)
20. Cousot, P., Cousot, R.: Abstract interpretation frameworks. Journal of Logic and Computation **2**(4), 511–547 (1992)
21. Cousot, P., Cousot, R.: Comparing the Galois connection and widening/narrowing approaches to abstract interpretation, invited paper. In: PLILP'92, LNCS, pp. 269–295. Springer (1992)
22. Cousot, P., Halbwachs, N.: Automatic discovery of linear restraints among variables of a program. In: ACM POPL'78, pp. 84–97. ACM Press (1978)
23. Deutsch, A. Interprocedural may-alias analysis for pointers: Beyond k-limiting. In: ACM PLDI'94, pp. 230–241. ACM Press (1994)
24. Dill, D.L.: Timing assumptions and verification of finite-state concurrent systems. In: Proc. International Workshop on Automatic verification Methods for Finite State Systems vol. 407 of LNCS, pp. 197–212, Springer
25. Dor, N., Rodeh, M., Sagiv, M.: Cleanness checking of string manipulations in C programs via integer analysis. In: SAS'01, vol. 2126 of LNCS. Springer (2001)
26. Feret, J.: Abstract interpretation of mobile systems. JLAP (2004)
27. Feret, J.: Static analysis of digital filters. In: ESOP'04, vol. 2986 of LNCS. Springer (2004)
28. Feret, J.: The arithmetic-geometric progression abstract domain. In: VMCAI'05, vol. 3385 of LNCS. Springer (2005)
29. Granger, P.: Static analysis of arithmetical congruences. International Journal of Computer Mathematics, **30**, 165–190 (1989)
30. Halbwachs, N.: Détermination automatique de relations linéaires vérifiées par les variables d'un programme. PhD thesis, Université scientifique et medicale de Grenoble, France (1979)
31. Handjieva, M., Tzolovski, S.: Refining static analyses by trace-based partitioning using control flow. In: SAS'98, vol. 1503 of LNCS, pp. 200–214 (1998)
32. Harvey, W., Stuckey, P.: A unit two variable per inequality integer constraint solver for constraint logic programming. In: ACSC'97, vol. 19, pp. 102–111 (1997).

33. Jaffar, J., Maher, M., Stuckey, P., Yap, H.: Beyond finite domains. In: PPCP'94, vol. 874 of LNCS, pp. 86–94. Springer (1994)
34. Jeannet, B.: Partitionnement dynamique dans l'analyse de relations Linéaires et application à la Vérification de programmes synchrones. PhD thesis, Institut National Polytechnique de Grenoble, France (2000)
35. Karr, M.: Affine relationships among variables of a program. Acta Informatica, pp. 133–151 (1976)
36. Larsen, K., Weise, C., Yi, W., Pearson, J.: Clock difference diagrams. Nordic Journal of Computing, **6**(3), 271–298 (1999)
37. Lions, J.L.: ARIANE 5, flight 501 failure, report by the Inquiry Board (1996)
38. Mauborgne, L.: ASTRÉE: Verification of absence of run-time error. In: Building the Information Society (18th IFIP World Computer Congress), vol. 156, pp. 385–392. Springer (2004)
39. Mauborgne, L., Rival, X.: Trace partitioning in abstract interpretation based static analyzers. In ESOP'05, vol. 3444 of LNCS, pp. 5–20. Springer (2005)
40. Measche, M., Berthomieu, B.: Time Petri-nets for analyzing and verifying time dependent communication protocols. Protocol Specification, Testing and Verification III, pp. 161–172 (1983)
41. Miné, A.: The octagon abstract domain library. `http://www.di.ens.fr/ mine/oct/`
42. Miné, A.: On-line octagon abstract domain sample analyzer. `http://cgi.di.ens.fr/cgi-bin/mine/octanalhtml/octanalweb/`
43. Miné, A.: A new numerical abstract domain based on difference-bound matrices. In: PADO II, vol. 2053 of LNCS, pp. 155–172. Springer (2001)
44. Miné, A.: The octagon abstract domain. In: AST 2001 in WCRE 2001, IEEE, pp. 310–319. IEEE CS Press (2001)
45. Miné, A.: Relational abstract domains for the detection of floating-point run-time errors. In: ESOP'04, vol. 2986 of LNCS, pp. 3–17. Springer (2004)
46. Miné, A.: Weakly relational numerical abstract domains. PhD thesis, École Polytechnique, Palaiseau, France (2004). `http://www.di.ens.fr/ mine/these/`
47. Møller, J., Lichtenberg, J., Andersen, H.R., Hulgaard, H.: Difference decision diagrams. In: CSL'99, vol. 1683 of LNCS, pp. 111–125. Springer (1999)
48. Moore, R.E.: Interval Analysis. Prentice Hall (1966)
49. OCaml.: The objective Caml system. `http://paulliac.inria.fr/ocaml`
50. Rugina, R.: Quantitative shape analysis. In: SAS'04, vol. 3148 of LNCS, pp. 228–245. Springer (2004)
51. Shaham, R., Kolodner, E.K., Sagiv, M.: Automatic removal of array memory leaks in java. In: CC'00, LNCS, pp. 50–66. Springer (2000)
52. Simon, A., King, A., Howe, J.: Two variables per linear inequality as an abstract domain. In: LOPSTR'02, vol. 2664 of LNCS, pp. 71–89. Springer (2002)
53. Venet, A.: Nonuniform alias analysis of recursive data structures and arrays. In: SAS'02, vol. 2477 of LNCS, pp. 36–51. Springer (2002)
54. Yovine, S.: Model-checking timed automata. In: Embedded Systems, vol. 1494 of LNCS. Springer (1998)