An Abstract Domain to Infer Octagonal Constraints with Absolute Value ⋆

Liqian Chen[1], Jiangchao Liu[2], Antoine Miné[2,3], Deepak Kapur[4], and Ji Wang[1]

[1] National Laboratory for Parallel and Distributed Processing,
National University of Defense Technology, Changsha, P.R.China
{lqchen,wj}@nudt.edu.cn
[2] École Normale Supérieure, Paris, France
{jliu, mine}@di.ens.fr
[3] CNRS, France
[4] University of New Mexico, NM, USA
kapur@cs.unm.edu

Abstract. The octagon abstract domain, devoted to discovering octagonal constraints (also called Unit Two Variable Per Inequality or UTVPI constraints) of a program, is one of the most commonly used numerical abstractions in practice, due to its quadratic memory complexity and cubic time complexity. However, the octagon domain itself is restricted to express convex sets and has limitations in handling non-convex properties which are sometimes required for proving some numerical properties in a program. In this paper, we intend to extend the octagon abstract domain with absolute value, to infer certain non-convex properties by exploiting the absolute value function. More precisely, the new domain can infer relations of the form $\{\pm X \pm Y \leq c, \pm X \pm |Y| \leq d, \pm|X| \pm |Y| \leq e\}$. We provide algorithms for domain operations such that the new domain still enjoys the same asymptotic complexity as the octagon domain. Moreover, we present an approach to support strict inequalities over rational or real-valued variables in this domain, which also fits for the octagon domain. Experimental results of our prototype are encouraging; The new domain is scalable and able to find non-convex invariants of interest in practice but without too much overhead (compared with that using octagons).

1 Introduction

The precision and efficiency of program analysis based on abstract interpretation [9, 10] rely a lot on the chosen abstract domains. Most existing numerical abstract domains (such as intervals [8], octagons [24], polyhedra [11], etc.) can only express convex sets, due to the fact that they usually utilize a conjunction of convex constraints to represent abstract elements. At control-flow joins in programs, an abstract domain often exploits a join operation to abstract the disjunction (union) of the convex constraint sets from the incoming edges into a conjunction of new convex constraints. The convexity limitations

⋆ This work is supported by the 973 Program under Grant No. 2014CB340703, the NSFC under Grant Nos. 61120106006, 61202120, 91118007, the NSF under Grant No. CCF-1248069 and an international fellowship by the Chinese Academy of Sciences.

of abstract domains may lead to imprecision in the analysis and thus may cause many false alarms. E.g., to remove a division-by-zero false alarm, the analysis needs to find a range excluding 0 for the divisor, which is in general a non-convex property and may be out of the reasoning power of convex abstract domains.

The *Absolute Value* (AV) function is one of the most used functions in mathematics and widely used in numerical computations. The AV function is supported by many modern program languages. E.g., the C99 standard for the C programming language provides the `abs()` and `fabs()` functions to compute the absolute value of an integer number and a floating-point number respectively. However, due to non-convexity, the AV function in the program code is rarely well handled during program analysis. Moreover, the AV function has natural ability to encode disjunctions of linear constraints in a program that account for a large class of non-convex constraints in practice. E.g., $x \leq -1 \vee x \geq 1$ can be encoded as $|x| \geq 1$, while $(x \neq 1 \vee y \neq 2)$ can be encoded as $|x-1| + |y-2| > 0$. Hence, we could exploit the non-convex expressiveness of the AV function to design non-convex abstract domains. Based on this insight, in [7], Chen et al. proposed an abstract domain of linear AV inequalities but which is exponential in complexity and thus has scalability limitations in practice.

In this paper, we propose a new abstract domain, namely the abstract domain of octagonal constraints with absolute value (AVO), to infer relations of the form $\{\pm X \pm Y \leq c, \pm X \pm |Y| \leq d, \pm |X| \pm |Y| \leq e\}$ over each pair of variables X, Y in the program where constants $c, d, e \in \mathbb{R}$ are automatically inferred by the analysis. AVO is more expressive than the classic octagon abstract domain and allows expressing certain non-convex (even unconnected) sets, thanks to the non-convex expressiveness of the AV function. We propose several closure algorithms over AV octagons to offer different time-precision tradeoffs. On this basis, we provide algorithms for domain operations such that the new domain still enjoys the same asymptotic complexity as the octagon domain. In addition, we show how to extend AVO to support strict inequalities over rational or real-valued variables. In other words, after the extension, AVO can additionally infer relations that are of the form $\{\pm X \pm Y < c, \pm X \pm |Y| < d, \pm |X| \pm |Y| < e\}$. Experimental results of our prototype are encouraging on benchmark programs and large embedded C programs; AVO is scalable to large-scale programs and able to find non-convex invariants of interest in practice.

Motivating Example. In Fig. 1, we show a small instructive example adapted from [14] (by replacing the `double` type by `real` type), which is originally extracted from the XTide[1] package that provides tide and current predictions in various formats. It shows a frequently used pattern in implementing a Digital Differential Analyzer algorithm in computer graphics. This example is challenging to analyze as it involves complicated non-convex constraints (due to disjunctions, the usage of the AV function) as well as strict inequalities, and precise reasoning over these constraints is required to prove the absence of the potential risk of division-by-zero errors.

At location ① in Fig. 1, it holds that $(dx \neq 0 \vee dy \neq 0)$ which describes a non-convex set of points that includes all points in $\mathbb{R}^2$ except the origin $(0, 0)$. Using octagonal constraints with absolute value, it can be encoded as $-|dx| - |dy| < 0$. At location ②, it holds that $-|dx| - |dy| < 0 \wedge |dx| - |dy| < 0$ which implies that $-|dy| < 0$ and thus the

[1] http://www.flaterco.com/xtide/

division by dy in the `then` branch will not cause division-by zero error. At location ③, it holds that $-|dx| - |dy| < 0 \wedge -|dx| + |dy| \leq 0$ which implies that $-|dx| < 0$ and thus the division by dx in the `else` branch will not cause division-by zero error. However, if using convex abstract domains such as octagons and polyhedra, $\top$ (no information) will be obtained at ① and thus the division-by-zero false alarms will be issued in both the `then` and `else` branches. Moreover, since the program involves strict inequality tests, we need an abstract domain supporting strict inequalities to do precise reasoning.

```
static void p_line16_primary (...) {
    real dx, dy, x, y, slope;
    ...
    if (dx == 0.0 && dy == 0.0)
        return;
 ① if (fabs(dy) > fabs(dx)) {
     ② slope = dx / dy;
        ...
    } else {
     ③ slope = dy / dx;
        ...
    } }
```

Loc	AV octagons
①	$-\|dx\| - \|dy\| < 0$
②	$-\|dx\| - \|dy\| < 0 \wedge$ $\|dx\| - \|dy\| < 0 \wedge$ $-\|dy\| < 0$
③	$-\|dx\| - \|dy\| < 0 \wedge$ $-\|dx\| + \|dy\| \leq 0 \wedge$ $-\|dx\| < 0$

Fig. 1. Motivating example from [14] which is originally extracted from the XTide package.

The rest of the paper is organized as follows. Section 2 reviews the octagon abstract domain. Section 3 presents a new abstract domain of octagonal constraints with absolute value. Section 4 presents our prototype implementation together with experimental results. Section 5 discusses some related work before Section 6 concludes.

2 The octagon abstract domain

In this section, we give a brief review of the background of the octagon abstract domain and we refer the reader to [24] for details.

2.1 Octagon representation

Let $\mathcal{V} = \{V_1, \ldots, V_n\}$ be a finite set of program variables in a numerical set $\mathbb{I}$ (which can be $\mathbb{Q}$, or $\mathbb{R}$). The octagon abstract domain manipulates a set of so-called *octagonal constraints* (also called Unit Two Variable Per Inequality or UTVPI constraints) that are of the form $\pm V_i \pm V_j \leq c$ where $\pm \in \{-1, 0, +1\}$ and $c \in \mathbb{I}$. From the geometric point of view, the set of points satisfying a conjunction of octagonal constraints forms an *octagon* (the projection of which on a 2D plane parallel to the axes is a 8-sided polygon).

Potential Constraints. An octagonal constraint over $\mathcal{V} = \{V_1, \ldots, V_n\}$ can be reformulated as a so-called potential constraint that is of the form $V'_i - V'_j \leq c$ over

$\mathcal{V}' = \{V'_1, \ldots, V'_{2n}\}$ where V'_{2k-1} represents $+V_k$ and V'_{2k} represents $-V_k$. E.g., the octagonal constraint $V_i + V_j \leq c$ can be encoded as either $V'_{2i-1} - V'_{2j} \leq c$ or $V'_{2j-1} - V'_{2i} \leq c$. Moreover, a unary octagonal constraint such as $V_i \leq c$ (and $-V_i \leq c$) can be encoded as $V'_{2i-1} - V'_{2i} \leq 2c$ (and $V'_{2i} - V'_{2i-1} \leq 2c$). A conjunction of potential constraints can be represented as a directed weighted graph $\mathcal{G}$ with nodes $\mathcal{V}'$ and edges labeled with weights in $\mathbb{I}$. For each constraint $V'_j - V'_i \leq c$ in the constraint conjunction, there will be an edge from V'_i to V'_j labelled with weight c in $\mathcal{G}$.

Difference Bound Matrices. An equivalent but more practical representation for the conjunction of potential constraints C over n variables is to use a *Difference Bound Matrix* (DBM) [12]. A DBM representing C is a $n \times n$ matrix M defined by

$$M_{ij} \stackrel{\text{def}}{=} \inf\{\, c \mid (V_j - V_i \leq c) \in C\}$$

where $\inf(\emptyset) = +\infty$ and n is the number of variables involved in C. For a set of potential constraints described by a DBM M of dimension n, we define the following concretization function $\gamma^{Pot} : \mathsf{DBM} \to \mathcal{P}(\mathcal{V} \to \mathbb{I})$:

$$\gamma^{Pot}(M) \stackrel{\text{def}}{=} \{\, (V_1, \ldots, V_n) \in \mathbb{I}^n \mid \forall i, j, V_j - V_i \leq M_{ij} \,\}.$$

Similarly, for a set of octagonal constraints described by a DBM M of dimension $2n$, we define the following concretization function $\gamma^{Oct} : \mathsf{DBM} \to \mathcal{P}(\mathcal{V} \to \mathbb{I})$:

$$\gamma^{Oct}(M) \stackrel{\text{def}}{=} \{\, (V_1, \ldots, V_n) \in \mathbb{I}^n \mid (V_1, -V_1, \ldots, V_n, -V_n) \in \gamma^{Pot}(M) \,\}.$$

Some octagonal constraints over $\mathcal{V}$ have two different encodings as potential constraints over $\mathcal{V}'$, and thus can be represented by two elements in the DBM. E.g., $V_i + V_j \leq c$ can be described by either $V'_{2i-1} - V'_{2j} \leq c$ (i.e., $M_{(2j)(2i-1)} = c$) or $V'_{2j-1} - V'_{2i} \leq c$ (i.e., $M_{(2i)(2j-1)} = c$). To ensure that elements of such pairs encode equivalent constraints, we define the coherence of a DBM as

$$M \text{ is coherent} \iff \forall i, j,\ M_{ij} = M_{\bar{j}\bar{i}}$$

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

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

Let DBM denote the set of all DBMs. We enrich DBM with a new smallest element, denoted by $\bot^{\mathsf{DBM}}$. Then we get a lattice $(\mathsf{DBM}, \sqsubseteq^{\mathsf{DBM}}, \sqcup^{\mathsf{DBM}}, \sqcap^{\mathsf{DBM}}, \bot^{\mathsf{DBM}}, \top^{\mathsf{DBM}})$ where

$$M \sqsubseteq^{\mathsf{DBM}} N \stackrel{\text{def}}{\iff} \forall i, j, M_{ij} \leq N_{ij} \qquad (M \sqcup^{\mathsf{DBM}} N)_{ij} \stackrel{\text{def}}{=} \max(M_{ij}, N_{ij})$$
$$(\top^{\mathsf{DBM}})_{ij} \stackrel{\text{def}}{=} +\infty \qquad (M \sqcap^{\mathsf{DBM}} N)_{ij} \stackrel{\text{def}}{=} \min(M_{ij}, N_{ij})$$

2.2 Closure

An octagon can still have several distinct representations using coherent DBMs. To compare octagons, we thus construct a normal form on DBMs to represent octagons.

The octagon abstract domain utilizes a so-called *strong closure* of the DBM, as the normal form for a non-empty DBM representing octagons. The strong closure (denoted as $M^\bullet$) of a DBM M encoding octagonal constraints is defined as:

$$M^\bullet \stackrel{\text{def}}{=} \inf_{\sqsubseteq^{\mathsf{DBM}}} \{X^\sharp \in \mathsf{DBM} \mid \gamma^{Oct}(M) = \gamma^{Oct}(X^\sharp)\}$$

The octagon domain uses a modified version of the Floyd-Warshall algorithm to compute $M^\bullet$ (which is firstly proposed by Miné [23] and later improved by Bagnara et al. [3]), which is of cubic-time complexity. Strong closure is a basic operator in the octagon domain. Most abstract operators over octagons can be obtained based on the strong closure of DBMs. We refer the reader to [24] for details.

3 An abstract domain of octagonal constraints with absolute value

In this section, we show how to extend the octagon abstract domain with absolute value.

3.1 Octagonal constraints with absolute value

A constraint is said to be an *AV octagonal constraint* if it is of the following forms:

- octagonal constraints: $\pm V_i \pm V_j \leq a$
- constraints with absolute value of one variable per inequality: $\pm V_i \pm |V_j| \leq b$
- constraints with absolute value of two variables per inequality: $\pm|V_i| \pm |V_j| \leq c$

where $\pm \in \{-1, 0, +1\}$ and $a, b, c \in \mathbb{I} \cup \{+\infty\}$. From the geometric point of view, we call *AV octagon* the geometric shape of the set of points satisfying a conjunction of AV octagonal constraints. Now, we will design a new abstract domain, namely AVO, to infer AV octagonal constraints among program variables $\mathcal{V} = \{V_1, \ldots, V_n\}$.

According to Theorem 1 in [7], it is easy to derive the following theorem.

Theorem 1. *Let e be an arbitrary expression that does not involve variable X. Then*

$$|X| + e \leq c \iff \begin{cases} X + e \leq c \\ -X + e \leq c \end{cases}$$

A direct consequence of Theorem 1 is that those constraints with positive coefficients on the AV term are redundant with other AV octagonal constraints and do not bring additional expressiveness. Hence, in the domain representation of AVO, we only need to encode AV octagonal constraints of the following forms:

- $\pm V_i \pm V_j \leq a$
- $\pm V_i - |V_j| \leq b$
- $-|V_i| - |V_j| \leq c$

For example, to describe a planar AV octagon over program variables x, y, we only need to consider at most 15 AV octagonal constraints, which are listed in Fig. 4(a).

Due to the non-convexity expressiveness of the AV function, an AV octagon is *non-convex* in general, but its intersection with each orthant in $\mathbb{R}^n$ gives a (possibly empty)

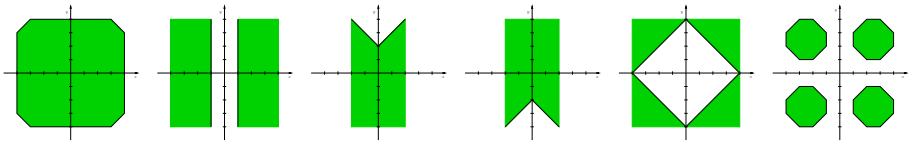

Fig. 2. The geometric shape of AV octagonal constraints. (a) depicts an octagon with constraint set $C = \{x \leq 4, -x \leq 4, y \leq 4, -y \leq 4, x + y \leq 7, x - y \leq 7, -x + y \leq 7, -x - y \leq 7\}$; (b) depicts $-|x| \leq -1$; (c) depicts $-|x| + y \leq 2$; (d) depicts $-|x| - y \leq 2$; (e) depicts $-|x| - |y| \leq -4$; (f) depicts an AV octagon with constraint set $C' = C \cup \{-|x| \leq -1, -|y| \leq -1, -|x| + y \leq 2, -|x| - y \leq 2, x - |y| \leq 2, -x - |y| \leq 2, -|x| - |y| \leq -4\}$.

octagon. Fig. 2 shows typical geometric shape of AV octagonal constraints. In particular, Fig. 2(a) shows a typical shape of octagons, while Fig. 2(f) shows an example of an AV octagon that is non-convex and even unconnected.

Expressiveness lifting. Note that in the AVO domain representation, the AV function $|\cdot|$ applies to only (single) variables rather than expressions. E.g., consider the relation $y = ||x| - 1| + 1$ which encodes a piecewise linear function with more than two pieces, whose plot is shown in Fig. 3. The AVO domain cannot express directly this piecewise linear function (in the space of x, y), since $|\cdot|$ applies to an expression $|x| - 1$. Indeed, in Fig. 3 the region in the orthant where both x and y are positive is not an octagon.

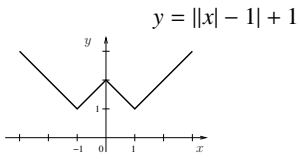

Fig. 3. A piecewise linear function with nested AV functions.

In order to express such complicated relations, we follow the same strategy as in [7]. We introduce new auxiliary variables to denote those expressions that appear inside the AV function. E.g., we could introduce an auxiliary variable ν to denote the value of the expression $|x| - 1$. Then using AVO domain elements in the space with higher dimension (involving 3 variables: x, y, ν), such as $\{y = |\nu| + 1, \nu = |x| - 1\}$, we could express complicated relations in the space over lower dimension (involving 2 variables: x, y), such as $y = ||x| - 1| + 1$. Note that due to the octagonal shape, the expression inside the AV function can only be

$$e ::= \quad \pm X \pm c \;\mid\; \pm|e| \pm c$$

where c is a constant and X is a variable.

3.2 Extending difference-bound matrices

Now, we show how to encode AV octagonal constraints using DBMs. Similarly to octagonal constraints, an AV octagonal constraint over $\{V_1, \ldots, V_n\}$ can be reformulated as a potential constraint of the form $V''_i - V''_j \leq c$ over $\{V''_1, \ldots, V''_{4n}\}$ where

- V''_{4k-3} represents $+V_k$,
- V''_{4k-2} represents $-V_k$,
- V''_{4k-1} represents $|V_k|$,
- V''_{4k} represents $-|V_k|$.

As an example, in Fig. 4, we show a general set of constraints for a planar AV octagon (left) and its DBM representation (right).

$$
\begin{array}{rrl}
x & & \leq a_1 \\
-x & & \leq a_2 \\
 & y & \leq a_3 \\
 & -y & \leq a_4 \\
x & +y & \leq a_5 \\
x & -y & \leq a_6 \\
-x & +y & \leq a_7 \\
-x & -y & \leq a_8
\end{array}
\qquad
\begin{array}{rrl}
-|x| & & \leq b_1 \\
 & -|y| & \leq b_2 \\
-|x| & +y & \leq b_3 \\
-|x| & -y & \leq b_4 \\
x & -|y| & \leq b_5 \\
-x & -|y| & \leq b_6 \\
\hline
-|x| & -|y| & \leq c_1
\end{array}
$$

(a)

	x	$-x$	$\lvert x\rvert$	$-\lvert x\rvert$	y	$-y$	$\lvert y\rvert$	$-\lvert y\rvert$
x	0	$2a_2$						
$-x$	$2a_1$	0						
$\lvert x\rvert$			0	$2b_1$				
$-\lvert x\rvert$				0				
y	a_6	a_8		b_4	0	$2a_4$		
$-y$	a_5	a_7		b_3	$2a_3$	0		
$\lvert y\rvert$	b_5	b_6		c_1			0	$2b_2$
$-\lvert y\rvert$								0

(b)

Fig. 4. DBMs for AV octagons. (a) shows a constraint set for a planar AV octagon; (b) shows a DBM to encode the constraints.

For a set of AV octagonal constraints described by a DBM M of dimension $4n$, we define the following concretization function $\gamma^{AVO} : \mathsf{DBM} \to \mathcal{P}(\mathcal{V} \to \mathbb{I})$:

$$\gamma^{AVO}(M) \stackrel{\text{def}}{=} \left\{ (V_1, \ldots, V_n) \in \mathbb{I}^n \mid (V_1, -V_1, |V_1|, -|V_1|, \ldots, V_n, -V_n, |V_n|, -|V_n|) \in \gamma^{Pot}(M) \right\}.$$

Some AV octagonal constraints have two different encodings as potential constraints in $\mathcal{V}''$, and can be represented by two elements in the DBM. E.g., $-|V_i| - |V_j| \leq c$ can be described by either $V''_{4j} - V''_{4i-1} \leq c$ (i.e., $M_{(4i-1)(4j)} = c$) or $V''_{4i} - V''_{4j-1} \leq c$ (i.e., $M_{(4j-1)(4i)} = c$). In addition, according to the specific property over AV constraints shown in Theorem 1, DBMs encoding AV octagons have another restriction, i.e.,

$$e + V_i \leq c_1 \wedge e - V_i \leq c_2 \implies e + |V_i| \leq \max(c_1, c_2) \tag{1}$$

where $e \in \{\pm V_j, \pm|V_j|\}$. To this end, we define the AV coherence of a DBM as

$$M \text{ is } \textit{AV coherent} \iff \begin{cases} \forall i, j,\ M_{ij} = M_{\bar{j}\bar{i}} \\ \forall j, k,\ M_{(4k)j} = \max(M_{(4k-3)j}, M_{(4k-2)j}) & \text{if } j \neq 4k \\ \forall i, k,\ M_{i(4k-1)} = \max(M_{i(4k-2)}, M_{i(4k-3)}) & \text{if } i \neq 4k-1 \end{cases}$$

The first condition is similar to the coherence condition for DBMs that encode octagons. The second condition is due to the restriction (1) over the $-|V_k|$ row, while the third condition is due to the restriction (1) over the $|V_k|$ column.

3.3 Conversions between octagons and AV octagons

The intersection of an AV octagon with each orthant gives an octagon. Based on this insight, we now present operators for conversions between octagons and AV octagons. Let $u = (1, \ldots, 1)^T$ be the unit vector, and $S^n = \{s \in \mathbb{R}^n \mid |s| = u\}$. We define an operator $C = \mathsf{S2Cons}(s)$ to derive a conjunction C of sign constraints from $s \in S^n$, such that

$$C_i \stackrel{\text{def}}{=} \begin{cases} x_i \geq 0 & \text{if } s_i = 1 \\ x_i \leq 0 & \text{if } s_i = -1 \end{cases}$$

First, we define an operator $\mathsf{AVO2Oct}(M, s)$ to convert an AV octagon (described by M that is a DBM of dimension $4n$) into an octagon (described by N that is a DBM of dimension $2n$) with respect to a given orthant (described by $s \in S^n$), as:

$$N \stackrel{\text{def}}{=} \mathsf{AVO2Oct}(M, s)$$

such that $\gamma^{Oct}(N)$ equals to the solution set of the conjunction of $\mathsf{S2Cons}(s)$ with the constraint set corresponding to M. From the algorithmic view, N can be easily obtained from M, by considering the sign of each variable defined in s. E.g.,

$$N_{(2k-1)(2k)} = \begin{cases} M_{(4k-3)(4k-2)} & \text{if } s_k = -1 \\ \min(M_{(4k-3)(4k-2)}, M_{(4k-1)(4k)}) & \text{if } s_k = 1 \end{cases}$$

where $M_{(4k-3)(4k-2)}$ and $M_{(4k-1)(4k)}$ denote the upper bounds for $(-V_k) - V_k$ and $(-|V_k|) - |V_k|$ respectively. If $V_k \geq 0$, we know the upper bound for $(-V_k) - V_k$ (denoted by $N_{(2k-1)(2k)}$ in the DBM representation of octagons) will be $\min(M_{(4k-3)(4k-2)}, M_{(4k-1)(4k)})$.

Note that an octagon itself is an AV octagon. However, if we know the orthant that an octagon lies in, we could deduct additionally upper bounds for AV expressions (such as $-|X| - |Y|$), to saturate the DBM. To this end, we define an operator $\mathsf{Oct2AVO}(N, s)$ to convert an octagon (N) in a given orthant ($s \in S^n$) into an AV octagon (M), as:

$$M \stackrel{\text{def}}{=} \mathsf{Oct2AVO}(N, s)$$

such that the solution set of the conjunction of the constraint set corresponding to M with $\mathsf{S2Cons}(s)$ is equivalent to $\gamma^{Oct}(N)$.

3.4 Closure algorithms

To obtain a unique representation for a non-empty AV octagon, we define the so-called *AV strong closure* $M^{|\bullet|}$ for a DBM encoding a non-empty AV octagon, as

$$M^{|\bullet|} \stackrel{\text{def}}{=} \inf_{\sqsubseteq^{\mathsf{DBM}}} \{X^\sharp \in \mathsf{DBM} \mid \gamma^{AVO}(M) = \gamma^{AVO}(X^\sharp)\}$$

Strong closure by enumerating the signs of all n variables. We provide an approach to compute the AV strong closure $M^{|\bullet|}$ by enumerating the signs of all n variables:

$$\mathsf{AVOStrClo}(M) \stackrel{\text{def}}{=} \bigsqcup_{s \in S^n}^{\mathsf{DBM}} \{M' \in \mathsf{DBM} \mid M' = \mathsf{Oct2AVO}(N^\bullet, s), N = \mathsf{AVO2Oct}(M, s)\}$$

The intuition is as follows. The intersection of an AV octagon M with each orthant s gives an octagon N. Hence, we could enumerate all orthants and in each orthant we compute the AV strong closure via the regular strong closure of the octagon domain. It

```
1  DBM^{4n×4n} WeakCloVia3Sign(M : DBM^{4n×4n}){
2    M', M'', M'^{|•|} : DBM^{12×12};
3    N : DBM^{6×6};
4    for k ← 1 to n
5      for i ← 1 to n
6        for j ← 1 to n {
7          M' ← M/{V_k, V_i, V_j};
8          M'^{|•|} ← ⊔^{DBM}_{s∈S^3}{M'' ∈ DBM | M'' = Oct2AVO(N^•, s), N = AVO2Oct(M', s)};
9          M/{V_k, V_i, V_j} ← M'^{|•|}; }
10   for i ← 1 to 4n
11     if (M_ii < 0) return ⊥^{DBM};   else M_ii ← 0;
12   return M; }
```

Fig. 5. The weak closure algorithm by enumerating the signs of 3 variables in each step. $M/\{V_k, V_i, V_j\}$ denotes the sub-matrix of M consisting of the rows and columns corresponding to variables in $\{V_k, V_i, V_j\}$.

is not hard to see that $\mathsf{AVOStrClo}(M) = M^{|\bullet|}$. However, the time complexity of this approach is $O(2^n \times n^3)$. At the moment, we do not know whether the problem of computing the AV strong closure for AV octagons is NP-hard or not.

To offer different time-precision tradeoffs, we now propose two approaches that are of cubic time complexity to compute weak closures M° (such that $M^{|\bullet|} \sqsubseteq^{\mathsf{DBM}} M^\circ$) for AV octagons. Note that the key behind the closure algorithm is to combine the constraints over (V_i, V_k) and those over (V_k, V_j) to tighten the constraints over (V_i, V_j), by constraint propagation through the intermediate variable V_k. Based on this insight, we first propose a weak closure algorithm WeakCloVia3Sign() by enumerating the signs of 3 variables $\{V_i, V_k, V_j\}$ each time to perform constraint propagation. Then we propose a cheaper weak closure algorithm WeakCloVia1Sign() by enumerating only the sign of the intermediate variable V_k each time to perform constraint propagation.

Weak closure by enumerating the signs of 3 variables each time. In Fig. 5, we show the WeakCloVia3Sign() algorithm. In the loop body, we compute the AV strong closure among three variables V_i, V_k, V_j (by enumerating 8 orthants due to the signs of 3 variables), and then update the tightened constraints over V_i, V_k, V_j in the original DBM. Note that WeakCloVia3Sign() gives AV strong closure for AV octagons involving only 3 variables. However, in general, WeakCloVia3Sign() does not guarantee to result in the AV strong closure for more than 3 variables.

Weak closure by enumerating the sign of one variable each time. In Fig. 7, we show the WeakCloVia1Sign() algorithm. Rather than enumerating the signs of 3 variables, in the loop body of WeakCloVia1Sign() we enumerate only the sign of the intermediate variable V_k. For each case of the sign of V_k, we call TightenIJviaK() which is shown in Fig. 6 to tighten the constraints over $\{\pm V_i, \pm|V_i|, \pm V_j, \pm|V_j|\}$ by combining the constraints over $\{\pm V_i, \pm|V_i|, \pm V_k\}$ and those over $\{\pm V_k, \pm|V_j|, \pm V_j\}$. We now explain how TightenIJviaK() works by considering the case where $V_k \geq 0$. When $V_k \geq 0$, we have $|V_k| = V_k$. Hence, it holds that $V'' - |V_k| \leq c \implies V'' - V_k \leq c$ where $V'' \in \{0, \pm V_i, \pm|V_i|, \pm V_j, \pm|V_j|\}$. Then, we use $V'' - |V_k| \leq c$ to tighten the upper bound

```
1  DBM^{12×12} TightenIJviaK(M : DBM^{12×12}, Kpositive : bool){
2    M', M'^{|•|} : DBM^{12×12};
3    M' ← M; k ← 1; i ← 2; j ← 3;          // Let V'' = {0, ±V_i, ±|V_i|, ±V_j, ±|V_j|}
4    if (Kpositive == true){  // V'' − |V_k| ≤ c ⟹ V'' − V_k ≤ c where V'' ∈ V''
5      M'_{(4k−3)(4k−2)} ← min(0, M'_{(4k−3)(4k−2)}, M'_{(4k−1)(4k)});
6      for n ← 0 to 3 {
7        M'_{(4k−3)(4i−n)} ← min(M'_{(4k−3)(4i−n)}, M'_{(4k−1)(4i−n)});
8        M'_{(4j−n)(4k−2)} ← min(M'_{(4j−n)(4k−2)}, M'_{(4j−n)(4k)}); } }
9    else{                    // V'' − |V_k| ≤ c ⟹ V'' + V_k ≤ c where V'' ∈ V''
10     M'_{(4k−2)(4k−3)} ← min(0, M'_{(4k−2)(4k−3)}, M'_{(4k−1)(4k)});
11     for n ← 0 to 3 {
12       M'_{(4k−2)(4i−n)} ← min(M'_{(4k−2)(4i−n)}, M'_{(4k−1)(4i−n)});
13       M'_{(4j−n)(4k−3)} ← min(M'_{(4j−n)(4k−3)}, M'_{(4j−n)(4k)}); } }
14   for n ← (4 ∗ i − 3) to 4 ∗ j
15     for m ← (4 ∗ i − 3) to 4 ∗ j
16       M'_{nm} ← min(M'_{nm}, M'_{n(4k−3)} + M'_{(4k−3)m}, M'_{n(4k−2)} + M'_{(4k−2)m});
17         // V_k − V''_n ≤ c ∧ V''_m − V_k ≤ d ⟹ V''_m − V''_n ≤ c + d where V''_n, V''_m ∈ V'' \ {0}
18   return M'; }
```

Fig. 6. A algorithm to tighten AV constraints between V_i and V_j through V_k.

for $V'' - V_k$. E.g., if we have $-V_k \leq c_1$ and $-|V_k| \leq c_2$ in the input DBM, we can derive a upper bound for $-V_k$ as $-V_k \leq \min(0, c_1, c_2)$, which corresponds to line 5 in TightenIJviaK(). After line 14, the information over the rows and columns corresponding to $\pm|V_k|$ in the DBM becomes redundant. Hence, from line 14 to line 16, we only need to consider the propagation through $\pm V_k$ (without need through $\pm|V_k|$). Overall, WeakCloVia1Sign() is less precise but cheaper than WeakCloVia3Sign().

```
1  DBM^{4n×4n} WeakCloVia1Sign(M : DBM^{4n×4n}){
2    M', M'^{|•|}, N, N' : DBM^{12×12};
3    for k ← 1 to n
4      for i ← 1 to n
5        for j ← 1 to n {
6          M' ← M/{V_k, V_i, V_j};
7          N ← TightenIJviaK(M', true);     // when V_k ≥ 0
8          N' ← TightenIJviaK(M', false);   // when V_k ≤ 0
9          M'^{|•|} ← N ⊔^{DBM} N';
10         M/{V_i, V_j} ← M'^{|•|}/{V_i, V_j}; }
11   for i ← 1 to 4n
12     for j ← 1 to 4n
13       M_{ij} ← min(M_{ij}, (M_{ī} + M_{j̄})/2);
14   for i ← 1 to 4n
15     if (M_{ii} < 0) then return ⊥^{DBM}; else M_{ii} ← 0;
16   return M; }
```

Fig. 7. The weak closure algorithm by enumerating the sign of one variable in each step.

The initial constraint set

$y \leq 24$	$-\lvert y\rvert + x \leq 10$	$-s - \lvert x\rvert \leq 36$
$-\lvert s\rvert - z \leq 8$	$-z - y \leq 84$	$s + y \leq 80$

Common constraints found by 3 closure algorithms

$s - z \leq 164$	$y + x \leq 58$	$y - z \leq 132$
$-z \leq 108$	$x - \lvert z\rvert \leq 94$	

AV strong closure

$-\lvert x\rvert - z$	$\leq$	86
$x - z$	$\leq$	112

WeakCloVia3Sign

$-\lvert x\rvert - z$	$\leq$	86
$x - z$	$\leq$	142

WeakCloVia1Sign

$-\lvert x\rvert - z$	$\leq$	108
$x - z$	$\leq$	142

Fig. 8. An example of applying 3 closure algorithms on the same initial constraint set.

Example 1. In Fig. 8, we apply the above 3 closure algorithms on the same initial set of constraints. The AV strong closure finds $x - z \leq 112$ while WeakCloVia3Sign() and WeakCloVia1Sign() are less precise and can only find $x - z \leq 142$. Moreover, WeakCloVia1Sign() gives less precise result $-|x| - z \leq 108$ than WeakCloVia3Sign() which can find $-|x| - z \leq 86$.

3.5 Other domain operations

Closure is a basic operator in the AVO domain. Most abstract operators over AV octagons can be obtained following similar ideas as those over octagons by replacing strong closure with AV strong closure (if necessary). In practice, since our AV strong closure is of exponential-time complexity, we use weak closure instead. When we use weak closure, all the AVO domain operations can be $O(n^3)$ in the worst case. However, we do not have a normal form for AV octagons when using weak closure, and most domain operations are not guaranteed to be the best abstraction. E.g., for the inclusion test, we have $\gamma^{AVO}(M) \subseteq \gamma^{AVO}(N) \iff M^{|\bullet|} \sqsubseteq^{\mathsf{DBM}} N$ when using AV strong closure. If we use any of our weak closures, denoted as $M^{|\circ|}$, it holds that $M^{|\circ|} \sqsubseteq^{\mathsf{DBM}} N \implies \gamma^{AVO}(M) \subseteq \gamma^{AVO}(N)$ but it may not hold that $\gamma^{AVO}(M) \subseteq \gamma^{AVO}(N) \implies M^{|\circ|} \sqsubseteq^{\mathsf{DBM}} N$.

For test transfer functions, first, constraints in the tests are abstracted into AV octagonal constraints, following similar ideas as abstracting arbitrary constraints into octagonal constraints [24]. Moreover, we employ AVO join operation to try to encode disjunctive constraints in tests as conjunctive AV octagonal constraints. E.g., consider the condition that holds at ① in Fig. 1, i.e., $|dx| \neq 0 \vee |dy| \neq 0$. The disequality $|dx| \neq 0$ which itself can be rewritten as a disjunction $dx < 0 \vee -dx < 0$, can be encoded as $-|dx| < 0$ by the AVO join operation. Then, $-|dx| < 0 \vee -|dy| < 0$ can be further encoded as $-|dx| - |dy| < 0$ by the AVO join operation. Hence, even when the original condition test does not involve AV, AV may be introduced during constraint abstraction. After the process of constraint abstraction, the AV octagonal constraints derived from the tests are then used to tighten the current AVO abstract element.

For assignment transfer functions, we allow the right-hand side expression to involve AV, such as $x := \pm|y| \pm c$. However, we can simply transform assignments with AV into conditional branches with assignments that do not involve AV. E.g., the assignment $x := a * |e| + c$ where a, e, c are expressions, can be transformed into: `if` $(e \geq 0)$ `then` $x := a * e + c$; `else` $x := -a * e + c$; `fi`.

3.6 Supporting strict inequalities

In practice, strict inequalities (such as $|x| + |y| > 0$) may appear in branch conditions of a program. To this end, we extend the AVO domain to support strict inequalities. In the domain representation, we maintain a boolean matrix S of the same size as the DBM M that encodes an AV octagon, such that

$$S_{ij} \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } V''_j - V''_i < M_{ij} \\ 1 & \text{if } V''_j - V''_i \leq M_{ij} \end{cases}$$

We define the order over pairs (m, s)'s where $m \in \mathbb{I}$ and $s \in \{0, 1\}$, as

$$(m, s) \sqsubseteq (m', s') \iff (m < m') \vee (m = m' \wedge s \leq s')$$

Note that $\sqsubseteq$ is a total order on $(\mathbb{I}, \mathsf{bool})$, i.e., at least one of $(m, s) \sqsubseteq (m', s')$ and $(m', s') \sqsubseteq (m, s)$ holds. Let DBMS denote the set of all pairs of DBMs and boolean matrices. A lattice over DBMS can be obtained by "lifting" the operations from DBM and the boolean matrices element-wise. In addition, we define the addition over DBMS as:

$$(M_{ik}, S_{ik}) + (M_{kj}, S_{kj}) \stackrel{\text{def}}{=} (M_{ik} + M_{kj}, S_{ik} \& S_{kj})$$

Then in the abstract domain supporting strict inequalities, all domain operations can be adapted from the domain that supports only non-strict inequalities by replacing operations over DBM with operations over DBMS. E.g., in the AVO domain supporting strict inequalities, the emptiness test is to check whether it holds that $\exists i, M_{ii} < 0 \vee (S_{ii} = 0 \wedge M_{ii} = 0)$. Whereas, in the regular AVO domain, we only need to check whether it holds that $\exists i, M_{ii} < 0$.

4 Implementation and experimental results

We have implemented the AVO domain in the APRON abstract domain library [19].

4.1 Experimental comparison of three closure algorithms

We first compare in precision and efficiency the three closure algorithms proposed in Sect.3.4 for AV octagons. We conduct our experiments on randomly generated DBMs (of dimension $4n$ but partially initialized) over different numbers of variables (n). The experimental result is shown in Fig .9. "♯cases" gives the number of test cases for each such number n. "str" denotes the AV strong closure algorithm, "wk3s" denotes WeakCloVia3Sign(), and "wk1s" denotes WeakCloVia1Sign(). "%same_results" shows the percentage of test cases where the two compared algorithms give the same resulting DBMs. The column "%different_elements" presents the average percentage of the number of different elements in the resulting DBMs to the size of the DBMs, when the two compared algorithms produce different resulting DBMs.

From the result, we can see that WeakCloVia1Sign() is much more efficient than the other two closure algorithms. For those test cases where the two compared algorithms produce different resulting DBMs, the percentage of the number of different elements

♯vars	♯cases	average time			%same_results			%different_elements		
		str	wk3s	wk1s	str= wk3s	str= wk1s	wk3s= wk1s	str≠ wk3s	str≠ wk1s	wk3s ≠ wk1s
4	10000	2.4ms	7ms	0.19ms	94%	94%	99%	0.94%	0.79%	0.78%
8	1000	380ms	160ms	20ms	36%	28%	74%	0.83%	1.5%	0.26%
10	1000	5.7s	410ms	53ms	10%	5.3%	51%	1.1%	2.1%	0.18%

Fig. 9. An experimental comparison of 3 closure algorithms on randomly generated DBMs.

in the resulting DBMs is very low. In other words, the two different resulting DBMs are mostly the same except for very few elements. During our experiments, at the moment, we found no test case for which weak closures give $+\infty$ for an element where the strong closure gives a finite constant in the resulting DBMs.

4.2 Experiments on NECLA division-by-zero benchmarks

We have conducted experiments using the Interproc [20] static analyzer on the NECLA Benchmarks: Division-by-zero False Alarms [14]. The benchmark set is extracted from source code of several open-source projects. These programs illustrate commonly used techniques that programmers use to protect a division-by-zero (e.g., by using the AV function), and are challenging for analysis since they involve non-convex constraints (e.g., disjunctions, constraints involving the AV function) and strict inequalities.

Fig. 10 shows the comparison of invariants inferred by AVO (using the weak closure algorithm WeakCloVia1Sign) with those by the octagon domain [24] and by the donut domain [14] (the main idea of which is to represent concrete object by the so-called hole that is the set minus of two convex sets). The `motiv` program corresponds to the motivating example (shown in Fig. 1) with its two branches. The column "WCfS" gives the weakest condition to prove the absence of the division-by-zero error in the program. The results given in the column "donut domain" are taken from [14] (using boxes to encode holes). From Fig. 10, we can see that the octagon domain fails to prove the absence of division-by-zero error for all programs since it cannot express non-convex properties nor strict inequalities. Our AVO domain succeeds to prove the absence of the division-by-zero errors for all programs including *xcor* on which the donut domain fails (due to its default heuristic for choosing holes).

program	WCfS	donut domain		octagons		AV octagons	
		invariants	♯false alarms	invariants	♯false alarms	invariants	♯false alarms
motiv(if)	$dy \neq 0$	$dy \neq 0$	0	$dy \in [-\infty, +\infty]$	1	$\|dy\| > 0$	0
motiv(else)	$dx \neq 0$	$dx \neq 0$	0	$dx \in [-\infty, +\infty]$	1	$\|dx\| > 0$	0
gpc	$den \neq 0$	$den \notin [-0.1, 0.1]$	0	$den \in [-\infty, +\infty]$	1	$\|den\| > 0.1$	0
goc	$d \neq 0$	$d \notin [-0.09, 0.09]$	0	$d \in [-\infty, +\infty]$	1	$\|d\| \geq 0.1$	0
x2	$Dx \neq 0$	$Dx \neq 0$	0	$Dx \in [-\infty, +\infty]$	1	$\|Dx\| > 0$	0
xcor	$usemax \neq 0$	$usemax \notin [1, 10]$	1	$usemax \geq 0$	1	$usemax > 0$	0

Fig. 10. Experimental results on NECLA division-by-zero benchmarks.

code id	size (KLOC)	octagons			AV octagons			result comparison		
		time (s)	♯alarm	♯iter.	time (s)	♯alarm	♯iter.	♯ alarm reduction	time increase	♯ iter. reduction
*P*1	154	6216	881	110	7687	881	110	0	23.66%	0
*P*2	186	6460	1114	116	7854	1114	115	0	21.58%	1
*P*3	103	1112	403	25	2123	403	25	0	90.92%	0
*P*4	493	17195	4912	158	38180	4912	158	0	122.04%	0
*P*5	661	18949	7075	105	43660	7070	104	5	130.41%	1
*P*6	616	34639	8192	118	70541	8180	108	12	103.65%	10
*P*7	2428	99853	10980	317	217506	10959	317	21	117.83%	0
*P*8	3	517	0	19	581	0	19	0	12.38%	0
*P*9	18	534	16	27	670	16	27	0	25.47%	0
*P*10	26	1065	102	42	1133	102	42	0	6.38%	0

Fig. 11. Experimental results using ASTRÉE on large embedded C codes.

4.3 Experiments on ASTRÉE

We have also evaluated the scalability of AVO when analyzing large realistic programs, by integrating it into the ASTRÉE analyzer [4] and analyzing its dedicated benchmarks: a set of large embedded industrial C codes performing much integer and float computation. ASTRÉE contains many abstract domains, including octagons and disjunctive domains (such as trace partitioning and decision diagrams) and domains specialized for the analyzed benchmarks; It is carefully tuned to give few alarms and remain efficient. Hence, we did not expect the AVO domain to bring a notable increase in precision (by simply replacing octagons with AVO, a single program featured a reduction of 4 alarms). For a more fair comparison, we evaluated how AVO could replace, by its natural ability to represent disjunctions, the dedicated disjunctive domains in ASTRÉE. We disabled these disjunctive domains and ran analyses with the regular octagon domain and with AVO. Following the experiments from Fig. 9, we chose to use the more scalable weak closure WeakCloVia1Sign for these large analyses. The results are shown in Fig. 11. The last columns give the number of alarms removed by using AVO and the increase in analysis time. We observe three instances of alarm reductions and an increase of up to +130% of analysis time at worst. Additionally, the majority of codes are composed of a single large synchronous loop running 10^6 iterations, and we provide for those the number of abstract iterations needed to reach a fixpoint. Our experiments show that using the more precise AVO domain can slightly increase the convergence rate and never decrease it. Overall, our results show that, although it cannot compete with domains specifically tailored to analyze a code family, AVO nevertheless brings modest improvements in precision, and keeps the analysis time in the same order of magnitude.

5 Related work

In abstract interpretation, most existing numerical abstract domains can only express convex sets, such as the classical convex polyhedra domain [11] together with all its

subdomains (including octagons [24], two variables per inequality (TVPI) [27], template polyhedra [26], subpolyhedra [21], etc.)

Until now, only a few numerical abstract domains natively allow representing non-convex sets, e.g., congruences [15], max-plus polyhedra [2], interval linear abstract domains [5, 6] and quadratic templates [1]. To enhance numerical abstract domain with non-convex expressiveness, some work makes use of BDDs [17, 18] while some makes use of mathematical functions that could express non-convex properties such as max [16] and the absolute value function [7]. The donut domain [14] utilizes the set difference of two convex sets to express non-convex properties. Recently, [13] studies the impact of using non-lattice abstract domains (including non-convex numerical abstract domains) and proposes general remedies for precision and termination.

The AVO domain that we introduce in this paper is closest to the abstract domain of linear AV inequalities [7] which can infer general linear AV constraints but is of exponential complexity. The AVO domain enjoys abstract operators in cubic time complexity and quadratic memory complexity. Moreover, the AVO domain supports strict inequalities. [25] presents an abstract domain extending DBMs (encoding potential constraints) with disequality constraints of the form "$x \neq y$" or "$x \neq 0$", rather than extending the octagon domain. Moreover, disequalities are different from strict inequalities in that a disequality is a disjunction of two strict inequalities, while in this paper we consider the conjunction of strict inequalities. The pentagon domain [22] also chooses on purpose to perform the closure in an incomplete (but sound) way, to improve the efficiency in practice at the cost of precision. Our purpose to have weak closure in this paper is similar, but to low down the complexity due to absolute value.

6 Conclusion

In this paper, we present an analysis to discover octagonal (or UTVPI) relations among the values and the absolute values of variables of a program ($\pm X \pm Y \leq c, \pm X \pm |Y| \leq d, \pm|X| \pm |Y| \leq e$) , which generalizes the octagon abstract domain ($\pm X \pm Y \leq c$) [24]. The analysis explores the absolute value function as a mean to describe non-convex behaviors in the program. First, we present a representation to encode AV octagons via DBMs. Then we propose 3 closure algorithms for AV octagons to offer different time precision tradeoffs. On this basis, we provide algorithms for domain operations such that the new domain still enjoys the cubic time complexity, as octagons. In addition, we present an approach to extend AVO to support strict inequalities over rational or real-valued variables, which also fits for octagons. Experimental results are encouraging on benchmark programs and large embedded C programs: AVO is scalable and able to find useful non-convex invariants, without too much overhead compared with octagons.

It remains for future work to consider the domain of AV integer octagonal constraints (i.e., AV octagonal constraints with integers as constant terms), wherein the key is to have a tight closure algorithm for AV integer octagonal constraints.

References

1. A. Adjé, S. Gaubert, and E. Goubault. Coupling policy iteration with semi-definite relaxation to compute accurate numerical invariants in static analysis. In *ESOP*, volume 6012 of *LNCS*,

pages 23–42. Springer, 2010.
2. X. Allamigeon, S. Gaubert, and E. Goubault. Inferring min and max invariants using max-plus polyhedra. In *SAS*, volume 5079 of *LNCS*, pages 189–204. Springer Verlag, 2008.
3. R. Bagnara, P. M. Hill, E. Mazzi, and E. Zaffanella. Widening operators for weakly-relational numeric abstractions. In *SAS*, LNCS, pages 3–18. Springer, 2005.
4. B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. A static analyzer for large safety-critical software. In *PLDI*, pages 196–207. ACM Press, 2003.
5. L. Chen, A. Miné, J. Wang, and P. Cousot. Interval polyhedra: An abstract domain to infer interval linear relationships. In *SAS*, volume 5673 of *LNCS*, pages 309–325. Springer, 2009.
6. L. Chen, A. Miné, J. Wang, and P. Cousot. An abstract domain to discover interval linear equalities. In *VMCAI*, volume 5944 of *LNCS*, pages 112–128. Springer, 2010.
7. L. Chen, A. Miné, J. Wang, and P. Cousot. Linear absolute value relation analysis. In *ESOP*, volume 6602 of *LNCS*, pages 156–175. Springer, 2011.
8. P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In *Proc. of the 2nd International Symposium on Programming*, pages 106–130. Dunod, Paris, 1976.
9. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, pages 238–252. ACM Press, 1977.
10. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *POPL*, pages 269–282. ACM Press, 1979.
11. P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *POPL*, pages 84–96. ACM Press, 1978.
12. D.L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Automatic Verification Methods for Finite State Systems*, volume 407 of *LNCS*, pages 197–212. Springer, 1989.
13. G. Gange, J. A. Navas, P. Schachte, H. Søndergaard, and P. J. Stuckey. Abstract interpretation over non-lattice abstract domains. In *SAS*, volume 7935 of *LNCS*, pages 6–24. Springer, 2013.
14. K. Ghorbal, F. Ivancic, G. Balakrishnan, N. Maeda, and A. Gupta. Donut domains: Efficient non-convex domains for abstract interpretation. In *VMCAI*, volume 7148 of *LNCS*, pages 235–250. Springer, 2012.
15. P. Granger. Static analysis of arithmetical congruences. *International Journal of Computer Mathematics*, pages 165–199, 1989.
16. B. S. Gulavani and S. Gulwani. A numerical abstract domain based on expression abstraction and max operator with application in timing analysis. In *CAV*, volume 5123 of *LNCS*, pages 370–384. Springer, 2008.
17. A. Gurfinkel and S. Chaki. Boxes: A symbolic abstract domain of boxes. In *SAS*, volume 6337 of *LNCS*, pages 287–303. Springer, 2010.
18. J. M. Howe, A. King, and C. Lawrence-Jones. Quadtrees as an abstract domain. *Electr. Notes Theor. Comput. Sci.*, 267(1):89–100, 2010.
19. B. Jeannet and A. Miné. Apron: A library of numerical abstract domains for static analysis. In *CAV*, volume 5643 of *LNCS*, pages 661–667. Springer, 2009.
20. G. Lalire, M. Argoud, and B. Jeannet. Interproc. http://pop-art.inrialpes.fr/people/bjeannet/bjeannet-forge/interproc/.
21. V. Laviron and F. Logozzo. Subpolyhedra: A (more) scalable approach to infer linear inequalities. In *VMCAI*, volume 5403 of *LNCS*, pages 229–244. Springer, 2009.
22. F. Logozzo and M. Fähndrich. Pentagons: A weakly relational abstract domain for the efficient validation of array accesses. *Sci. Comput. Program.*, 75(9):796–807, 2010.
23. A. Miné. The octagon abstract domain. In *Proc. of the Workshop on Analysis, Slicing, and Transformation (AST'01)*, pages 310–319. IEEE CS Press, 2001.

24. A. Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation*, 19(1):31–100, 2006.
25. M. Péron and N. Halbwachs. An abstract domain extending difference-bound matrices with disequality constraints. In *VMCAI*, volume 4349 of *LNCS*, pages 268–282. Springer, 2007.
26. S. Sankaranarayanan, H. Sipma, and Z. Manna. Scalable analysis of linear systems using mathematical programming. In *VMCAI*, volume 3385 of *LNCS*, pages 25–41. Springer, 2005.
27. A. Simon, A. King, and J. M. Howe. Two Variables per Linear Inequality as an Abstract Domain. In *LOPSTR*, volume 2664 of *LNCS*, pages 71–89. Springer, 2003.