Quantifying the Precision of Numerical Abstract Domains

Pascal Sotin

Quantifying the Precision of Numerical Abstract Domains

Pascal Sotin

pascal.sotin@inrialpes.fr
INRIA Grenoble Rhônes-Alpes
655 avenue de l'Europe
38334 Saint Ismier – France

Abstract. In the context of the Abstract Interpretation framework, initiated by Cousot and Cousot to model program static analyses, numerous numerical abstract domains have been proposed. Their number is due to both the intended usage of the domains (properties to prove) and the trade-off between precision and computation efficiency.
In this paper, we propose a way to quantify the precision of abstract values belonging to numerical domains like intervals, octagons or polyhedra. To do so, we rely on the fact that abstract values of that kind describe some volumes that can be measured. The article focuses on the definition and computation of this precision, also providing examples of application.

1 Introduction

Abstract interpretation is a well-known framework for modelling sound static analyses and building static analysers. In this framework, the program semantics, operating on some concrete domain, is over-approximated by an abstract semantics, operating on an abstract domain. The original proposal [CC77], by Cousot and Cousot, was built upon the interval abstract domain, where numerical variables of the program were handled through their possible range of values. This analysis has a good complexity but is imprecise. In [CH78], Cousot and Halbwachs formalise a much more precise abstract domain based on linear constraints, called convex polyhedra abstract domain. Unfortunately, the cost of this analysis is high and the approach does not scale easily. In [Min04], Miné proposes some abstract domains, called weakly relational, having a reasonable trade-off between precision and computation cost. For example, the octagon abstract domain is based on linear constraints with coefficients in $\{-1, 0, 1\}$ and involving at most two variables.

We assume the reader to be familiar with the notion of abstract interpretation and to have a minimal knowledge on the cited numerical abstract domains. Abstract value in these domains have in common to model subsets of $\mathbb{I}^{\mathcal{V}}$, where $\mathcal{V}$ is a set of numerical variables and $\mathbb{I}$ is one of the sets: $\mathbb{R}$, $\mathbb{Q}$, $\mathbb{Z}$.

Motivations. Within the framework of abstract interpretation, all the numerical domains may be compared *qualitatively*. It is known that the polyhedra are more precise than the octagons that are more precise than the intervals. The motivation for this work is to compare *quantitatively* the precision of these domains, relying on the fact that all of them express information on the space $\mathbb{I}^{\mathcal{V}}$. In particular the three domains mentioned before draw shapes in this space, and a human can visually compare the volumes of these shapes. We are aiming at a similar but automated comparison.

With the same motivation, but a different technique, Logozzo et al. [LPL09] compare automatically abstract values of numerical domains. They rely on the fact that these abstract values can be expressed as a (minimal) set of linear constraints. The comparison is based on the cardinality of these sets. In [RKG04] Rountev et al. describe an experiment where they have compared quantitatively class analyses for Java. This comparison is very accurate but not at all automatic. We argue that numerical domains offer the possibility of accurate and automatic measure of the precision of the abstract values. This measure can be used to compare quantitatively different static analyses.

Our proposal is mathematically well-defined, but is in a first place directed toward *practical* implementation and use in static analysers.

Contributions. We define a notion of precision of a numerical abstract value, based on the mathematical notion of *measure*. A measure associates a positive number to an element (of a σ-algebra). This number will be interpreted as the precision of the element, with the following principle: the lower, the most precise. This interpretation is compatible with the qualitative notion of precision in abstract interpretation. We examine a series of measure that could fit our purpose and conclude that the *exponential* measure is well-suited for abstract domains like intervals, octagons, polyhedra.

We describe efficient *algorithms* that compute a precision based on the exponential measure. For some abstract domains and some number of dimensions of the concrete domain, we can reach an exact algorithm based on analytical results. For cases where the exact algorithm is not known, or too expensive, we describe a general Monte-Carlo approximated algorithm.

We propose some applications relying on our notion of precision. We detail in particular a prototype in which we study the possibility to use the relative precision of two distinct abstract domains on a program to choose which one to use. The results of this experiment are promising.

Outline. Section 2 recalls the mathematical foundation of measure theory and defines our notion of precision of an abstract value. Section 3 provides algorithms to compute this precision for the abstract domains of intervals, octagon, polyhedra and more. In Sect. 4, we study the use of our precision for abstract domain selection on a given program. Section 5 lists possible applications together with questions raised by our proposal. Section 6 contains the related works and Sect. 7 a general conclusion.

2 Precision by Measure

In this section, we define the precision of an abstract value using the mathematical notion of measure. In 2.1 we introduce the concept of precision by measure, together with the necessary mathematical definitions. In 2.2 we focus on picking the right measure for numerical abstract domains.

2.1 Definition of the Precision

In this subsection, we recall the mathematical definition of a measure, we list the elements of Abstract Interpretation that we rely on and define formally our notion of precision.

Measures. A measure is a function which domain is a σ-algebra over a given set Ω. The result of this function gives a kind of estimate of the size of the measured object.

Definition 1 (σ-algebra). *A* σ-algebra *over the set Ω is a non-empty collection Σ of subsets of Ω closed under complementation and countable union.*

For any Ω, $\mathcal{P}(\Omega)$ is a σ-algebra. In the following, we will mainly use the σ-algebra $\mathcal{P}(\mathbb{R}^n)$.

Definition 2 (Measure). *A* measure*, μ, is a function from a σ-algebra Σ over a set Ω to $\mathbb{R}_+ \cup \{+\infty\}$, satisfying:*

1. $\mu(\emptyset) = 0$
2. *If (S_i) is a countable sequence of disjoint elements of Σ then:*

$$\mu\left(\biguplus_{S_i} S_i\right) = \sum_{S_i} \mu(S_i)$$

We now present the framework in which we find the elements to measure.

Abstract Interpretation. The framework of Abstract Interpretation, proposed by Cousot and Cousot [CC77], is a formal foundation to describe and compute sound static analyses. These analyses take program as an input and return an over-approximation of the possible behaviours of the program. If these behaviours are never faulty, then none of the real behaviours of the program will be.

As mentioned in the introduction, we assume that the reader is familiar with the basics of Abstract Interpretation. We will be mainly interested by three abstract domains: the intervals, the octagons and the polyhedra.

This article manipulates the notions of lattice as semantic domain, partial order ($\sqsubseteq$), least upper bound or join ($\sqcup$), greatest lower bound ($\sqcap$), bottom

element ($\bot$), top element ($\top$), Galois connection between two partially ordered sets $\left(C \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} D\right)$, abstraction ($\alpha$), concretisation ($\gamma$) and widening operator.

In the following, all our semantics domains are lattices equipped with a top and bottom element. These domains encompass powerset domains, intervals, octagons and polyhedra.

Precision of an Abstract Value. In Abstract Interpretation, top is the less precise value (everything may happen) and bottom the most precise value (nothing happen). The precision we define reflects this idea and the number associated to bottom will be low, while the one associated to top will be high: the smaller the better. In addition, the order relation $\sqsubseteq$ can be interpreted as "more precise than". The precision we propose is compatible with the order.

We define the precision of an abstract value by the measure of its concretisation.

Definition 3 (Precision of an abstract value). *Let $\mathcal{P}(\mathbb{R}^n) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} D$ be a Galois connexion and μ a measure on $\mathcal{P}(\mathbb{R}^n)$. The precision of an abstract value $d \in D$ is defined by:*

$$pre_{\mu}(d) = \mu(\gamma(d))$$

So far, the precision benefit from the following properties:

- $x \sqsubseteq y \Rightarrow pre_{\mu}(x) \leq pre_{\mu}(y)$
- if $\gamma(\bot) = \emptyset$ then $pre_{\mu}(\bot) = 0$

2.2 Measures for Numerical Abstract Domains

In this subsection, we examine several measures that might be used to compute the precision in numerical domains.

Simple Measures. A simple way to measure a set is to count the number of its elements.

Definition 4 (Counting measure). *We define the* counting measure *by:*

$$\mu_{\text{count}}(S) = \begin{cases} |S| & \text{if } S \text{ is finite} \\ +\infty & \text{otherwise} \end{cases}$$

This measure will provide useful information on the precision of an abstract value when the associated concrete value has a finite number of points (see Figs. 1(a) and 1(b)). However, it cannot distinguish two abstract values leading to two different infinite sets of points (See Figs. 1(c) and 1(d)).

The counting measure will be well suited to cope with abstract domains leading to finite subsets of concrete values, like the congruence abstract domain combined with intervals, octagons or polyhedra.

An alternative and yet simple measure that can handle abstract values such as the ones depicted in Figs. 1(c) and 1(d) is the n-dimensional volume (e.g. the surface, in 2 dimensions).

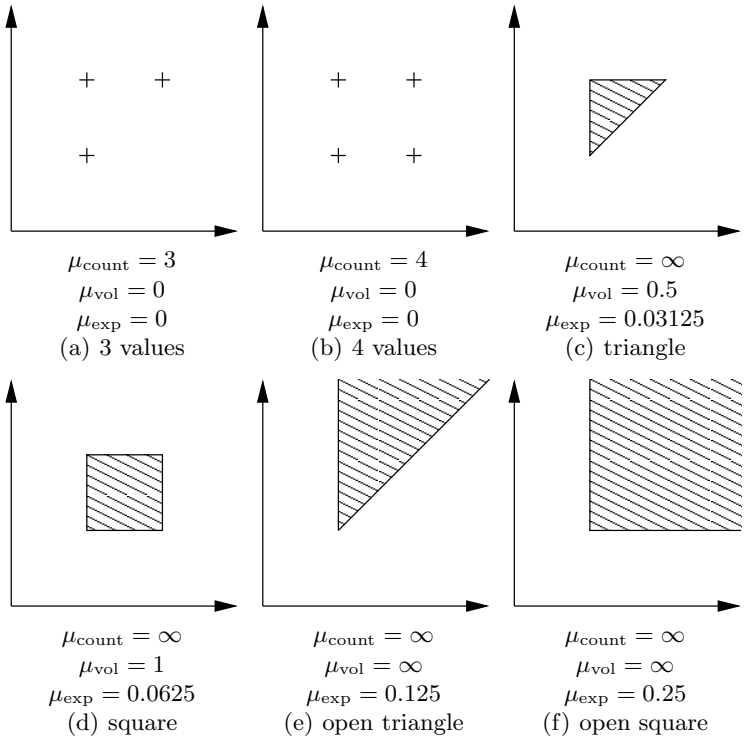


Fig. 1. Measures of various abstract values.

Definition 5 (Volume measure). *We define the* volume measure *by:*

$$\mu_{\text{vol}}(S) = \textit{(hyper-)volume of } S$$

The volume measure could be used to deal with domains like interval or polyhedra, including unions of polyhedra. Unfortunately, this measure (like the counting measure) does not distinguish shapes that are unbounded, which is a common characteristic of the abstract values manipulated by a static analyser.

As depicted in Figs. 1(e) and 1(f), the measures presented so far are not well-suited to deal with the precision of abstract value like intervals or octagons. We now look toward a measure over Ω satisfying the following properties:

- $\mu(\Omega) \neq +\infty$
- $\mu(S) \neq 0$, for some finite measurable set $S \subseteq \Omega$
- The computation of μ must be easy for simple shapes in $\mathbb{R}^n$, like the ones drawn by intervals or octagons.

Exponential Measure. We exploit the *exponential* measure, which fulfils the above conditions. The name of the measure comes from the exponential distri-

bution of probabilities.

$$\mu_{\exp}(S) = P(X \in S), \quad \text{with } X \sim \mathit{Exponential}(\Omega, \lambda)$$

where $\mathit{Exponential}(\Omega, \lambda)$ is a probability distribution extending the exponential distribution of parameter λ on the space Ω.

The exponential distribution is defined for $\Omega = \mathbb{R}_+$ by a probability density function $f(x, \lambda) = \lambda e^{-\lambda x}$, where $\lambda > 0$ is a parameter. We recall that a probability density function is a function such that $P(a \leq X \leq b) = \int_a^b f(x)dx$. Common extensions of this distribution handle negative values ($\Omega = \mathbb{R}$) and several random variables ($\Omega = \mathbb{R}^n$). See Table 1 for an idea of how the extension impacts the measure.

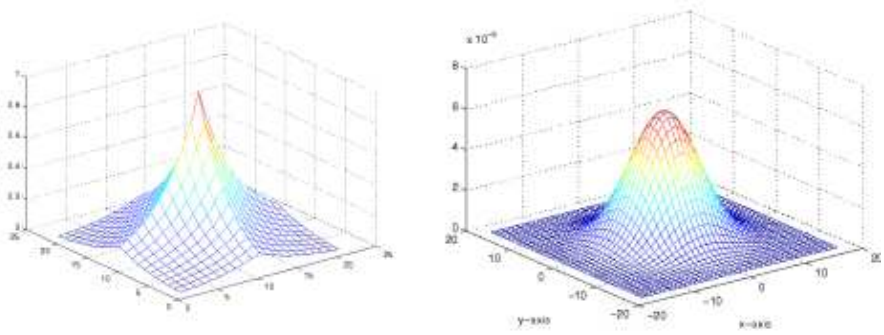
Space	Exponential random variables	$\mu(S) =$
$\mathbb{R}_+$	X	$P(X \in S)$
$\mathbb{R}$	X	$\frac{1}{2}\left(P(X \in S) + P(-X \in S)\right)$
$\mathbb{R}_+^2$	X, Y	$P(\langle X, Y\rangle \in S)$
$\mathbb{R}^2$	X, Y	$\frac{1}{4}\begin{pmatrix} P(\langle X, Y\rangle \in S) + P(\langle X, -Y\rangle \in S) \\ +P(\langle -X, Y\rangle \in S) + P(\langle -X, -Y\rangle \in S) \end{pmatrix}$

Table 1. Examples of generic exponential measures.

Many measures, built upon continuous probability distributions could deal with infinite shapes. For example, we depict in Fig. 2 the probability density function of a normal distribution and of an exponential distribution on $\mathbb{R}^2$. The measure of a shape in $\mathbb{R}^2$ is the integral of such functions, i.e. the volume between this shape and the curve. The exponential density function have the nice property to be easy to integrate.

The *median* is a value for which, when randomly picking points according to the distribution, points tend to be equally distributed above and under. The median of the exponential distribution is given by $\ln(2)/\lambda$. Passing from a one-variable distribution to its associated measure, the notion of median indicates the point which splits the line in two half-lines measuring each 0.5. For this reason, setting the median is more intuitive than setting directly the value of λ, yet it is equivalent.

The existence of the λ parameter is both a weakness and a strength for the exponential measure, w.r.t. its use in abstract domains. The weakness comes from the fact that it will under-valuate differences that arise far above this median point and over-valuate differences that arise close to the origin. The strength of the existence of this λ parameter, apart from allowing finite measure of infinite objects is to allow the user to focus on a given part of the plan, according to the expected range of activity of the program. In addition, the user can easily recenter the measure on a point different from the origin.



(a) Extended exponential (b) Multivariate normal

Fig. 2. Probability distributions on $\mathbb{R}^2$.

3 Algorithms

In this Section, we are interested in a computable version of the exponential measure for the abstract values: intervals, octagons and polyhedra. We present exact algorithms for nD intervals and 2D octagons, and Monte-Carlo algorithms for nD octagons and nD polyhedra (nD abstract values in general).

3.1 Analytical Measure of Intervals

In one dimension, an interval is a segment. In two dimensions, i.e. two variables programs, an abstract state is a pair of intervals reproducing the shape of a rectangle in the space. In three dimensions, we obtain a cuboid, whose faces are rectangles. In higher dimensions, we obtain what we could call a hyper-cuboid, whose faces are structures similar to those that can be found in lower dimensions. Furthermore, this structure can be unbounded on some sides (e.g. $]-\infty, 5]$).

The measure exponential of such abstract values is provided by:

$$\mu(\langle [a_1, b_1], \ldots, [a_n, b_n] \rangle) = \int_{x_1=a_1}^{b_1} \cdots \int_{x_n=a_n}^{b_n} \lambda e^{-\lambda.(\sum_{i=1}^{n} x_i)}$$

For example, in two dimensions, the formula can be derived into:

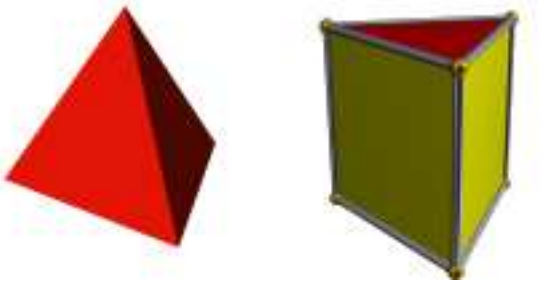
$$\mu(\langle [a_1, b_1], [a_2, b_2] \rangle) = \frac{1}{\lambda} \begin{pmatrix} \exp_\lambda (a_1 + a_2) - \exp_\lambda (b_1 + a_2) \\ - \exp_\lambda (a_1 + b_2) + \exp_\lambda (b_1 + b_2) \end{pmatrix}$$

where $\exp_\lambda (x) = \lambda e^{-\lambda x}$ is the probability density function given before. Note that all the primitives operations contained in the formula can be efficiently implemented (including the exponentiation).

These measures hold when all intervals belong to $\mathbb{R}_+$. When it is not the case, we first need to split the cuboids along the axes. For example, in two dimensions, we split the rectangle in four (possibly empty) rectangles, measuring them as if they were in $\mathbb{R}_+^2$. The measure becomes $\frac{1}{4}$ of the sum of all measures of the new rectangles.

3.2 Analytical Measure of Octagons

When dealing with octagons, the measure has to be computed by removing the corners of the cuboid. In two dimensions, an octagon is built by removing some triangles from a rectangle. In more than two dimensions, we need to figure what a corner means in terms of geometry. For example, in three dimensions, the number of corners to remove is not 8, the number of vertices of the cuboid, but 12, the number of edges of the cuboid. In other words, the pieces to be removed are not tetrahedra but triangular prisms, made of two right triangles joined by three rectangles (see Fig. 3). The problem caused by these prisms does not appear in two dimensions. In three or more dimensions, this prisms can intersect with each other, and there is a risk of (dis-)counting the same volume twice. We did not solved analytically the problem of removing the corners for dimensions higher than 2. However, the Monte-Carlo technique presented in the following may be used instead.



(a) Tetrahedron (b) Triangular prism

Fig. 3. About the corners of a three-dimension octagon.

In the same way that interval abstract values can be split into disjoint interval abstract values, whose union gives back the exact original shape, the octagons can also be split in octagons by the axes.

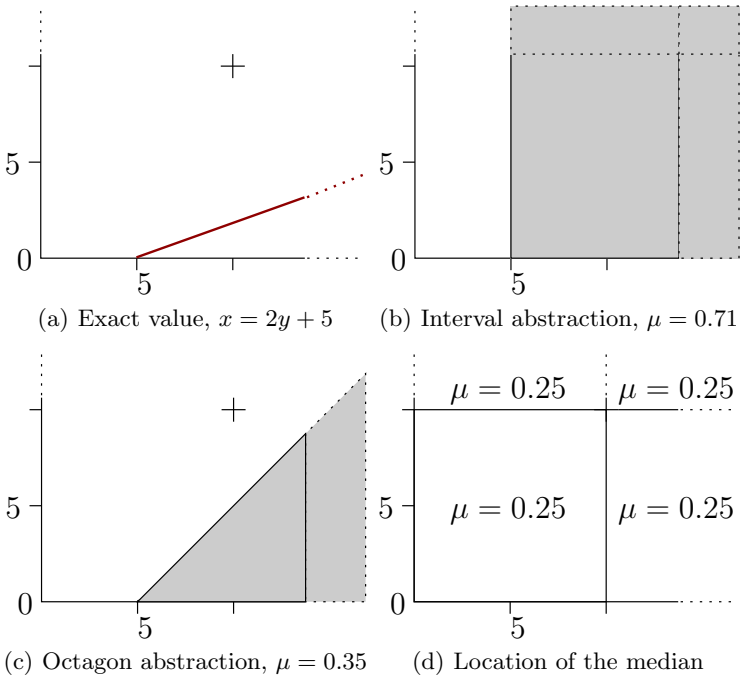
3.3 Approximate Measure of Polyhedra (or any other Domain)

When dealing with polyhedra, the measure by integration of the exponential function becomes far too complex, and we need approximated algorithms. The algorithm can be split in two phases. The first phase consist in obtaining a predicate of membership from the polyhedra: the abstract value itself, under its linear constraints form, can be used as predicate. The second phase is a Monte-Carlo algorithm [Fis96] where points are picked randomly, in conformance with the exponential distribution. The rate of membership of those points is an estimator of the volume. The random picking can be limited to the bounding box of the polyhedra and the rate is then adjusted knowing the exact measure of this box.

3.4 Implementation

We implemented the measure for two dimensional intervals and octagons. On Fig. 4, we show a concrete value in $\mathbb{R}_+[2]$, the set S of pairs $\langle x, y\rangle$ such that $x = 2y + 5$. This set is a line on the plan and so its measure by μ is 0 (see Fig. 4(a)). If we abstract with an interval abstraction, we get the following abstract value: $\langle x \in [5, +\infty[, y \in [0, +\infty[\rangle$, whose measure is 0.71 (see Fig. 4(b)). If we abstract with an octagon abstraction we get the following constraints: $\{0 \leq y, 5 \leq x, y + 5 \leq x\}$ whose measure is 0.35 (see Fig. 4(c)).

On the example shown in Fig. 4, all measures are done with 10 as median value, so $\lambda = \ln(2)/10$. In two dimensions, the plan is split in four quarters measuring each 0.25 (see Fig. 4(d)).



(a) Exact value, $x = 2y + 5$ (b) Interval abstraction, $\mu = 0.71$

(c) Octagon abstraction, $\mu = 0.35$ (d) Location of the median

Fig. 4. Precision through exponential measure.

4 Experiment

In the following, we present a prototype that tries to predict the precision ratio between an interval analysis and an octagon analysis. The principle of the experiment is described in 4.1 and the prototype implementation and its results are detailed in 4.2.

4.1 Precision Prediction

For a given program, distinct analyses will lead to distinct results, each of them having its own computation cost. Computation expensive analyses are supposed to be more precise than cheap ones. With our notion of precision, we can answer the question "how much precise" is the most expensive analysis, by comparing its quantitative precision with the cheap analysis one. Even better would be the capability to predict this difference: that is what we experiment in this section.

The concrete semantics of our program is expressed on the concrete domain:

$$\mathcal{L} \rightarrow \mathcal{P}(\mathbb{R}^{\mathcal{V}})$$

where $\mathcal{L}$ is a set of control points and $\mathcal{V}$ a set of variables (all being defined on $\mathbb{R}$). Let $\dot{\subseteq}$ be the pointwise extension w.r.t. the control points of the order relation $\subseteq$ on $\mathcal{P}(\mathbb{R}^{\mathcal{V}})$.

Let E be the set of reachable states of the program. The interval and octagon analyses compute sets that over-approximate E. We write $\mathbf{I}$ for the interval analysis result and $\mathbf{O}$ for the octagon analysis result. The lattice of octagon is qualitatively more precise than the lattice of intervals (existence of a Galois connexion). However, as the analysers may use non-optimal operators, either for termination (widening) or for efficiency reasons, the precision of the corresponding analyses are qualitatively unordered (we cannot prove $\mathbf{O} \dot{\subseteq} \mathbf{I}$). With the algorithms we presented in the previous section, it is possible to compute and compare, for each control point, the quantitative precisions $pre_{\mu_{\exp}}(\mathbf{I})$ and $pre_{\mu_{\exp}}(\mathbf{O})$.

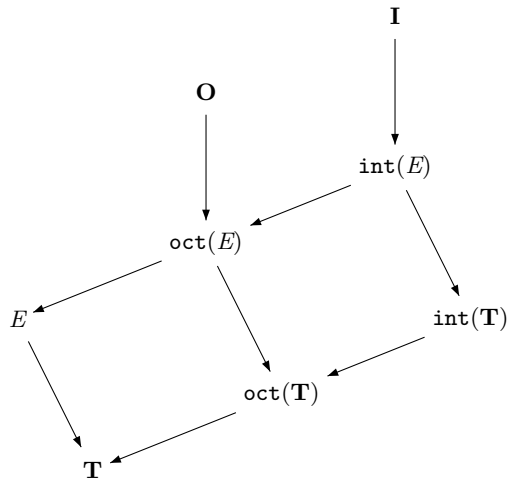
The difference between these precisions is valuable, but requires to run both analyses. We now try to predict this difference from a set of executions. If we execute the program several times with random inputs, and collect the visited states, it gives us a set that under-approximates the set of reachable states. We write $\mathbf{T}$ for a such a collection of states. To exploit $\mathbf{T}$, we introduce the notion of bounding rectangle or bounding octagon of a set S, respectively denoted by $\texttt{int}(S)$ and $\texttt{oct}(S)$. The abstract bounding for an abstract domain is defined by $\gamma \circ \alpha$.

The idea is to compute $\mathbf{T}$, $\texttt{int}(\mathbf{T})$, $\texttt{oct}(\mathbf{T})$ and given the relative difference between $\mu_{\exp}(\texttt{int}(\mathbf{T}))$ and $\mu_{\exp}(\texttt{oct}(\mathbf{T}))$, decide whether it is worthiest to compute $\mathbf{I}$ or $\mathbf{O}$. The prototype presented in the following computes all these five sets in order to experimentally figure out the cases when the prediction turns out to be good and when not. Figure 5 depicts the qualitative ordering on the defined sets.

4.2 Prototype

The prototype, developed in OCaml, computes $\mathbf{T}$, $\mathbf{I}$, $\mathbf{O}$ and $\texttt{int}(\mathbf{T})$, $\texttt{oct}(\mathbf{T})$ for a program in input. The interval analysis source code is from Pichardie [Pic05] (certified correct with the prover Coq), the octagon analysis source code is from Miné [Min04]. A simulation module has been developed to generate random

Fig. 5. $\dot{\subseteq}$ ordering.



values for the variables and to register the accessed points. A simple graphical 2D view of these accessed points is provided. The prototype only handles programs with two variables. This is sufficient for visualisation and abstract values can be measured with the analytical algorithms described before, with no need to implement Monte-Carlo for nD octagons.

The measures presented on Fig 4 can be computed by the prototype. Figure 4(a) depict the concrete values: two positive variables x and y such that $x = 2y + 5$. To create these values and their abstractions, we run the prototype on the following program:

```
if y>=0 {
x = 2*y+5; // (1) Measure (int vs oct) = 0.71 <> 0.35;
} else {};
```

The result of the analysis is here partially given as a comment. We kept the source code and the information added by the analyser comparing the respective measures of Pichardie's intervals and Miné's octagons at control point `(1)`.

The second example we present is from [Min04, ex251]. Table 2 and Fig. 6 display the prototype textual and graphical outputs for this program. The graphics uses an exponential inverse scale which counter-balances the fact that points are picked according to an exponential distribution. This choice is a practical convenience which avoids to use a regular truncated scale, where most of the points would have been stacked near 0 and where $+\infty$ could not have been depicted. The program is analysed in positive variables mode, i.e. measures are made only for the positive corner of the plan and test sets are picked in this same plan (a full space mode is also available). The median is set to 10 (see Fig. 4(d)).

Here are some comments on the example, made per program points.

`(0)` At the beginning of the program, all values are possible. This is shown in the textual output by two unbounded intervals and an octagon with no constraints. The visualisation of the tests looks like having no particular distribution. Indeed, the exponential random number generation comes from a twisted uniform distribution, with the exponential scale of the display, we get back the uniform distribution.

`(2)` The sequence `x = ?; while (x < 0 or x > 10) { x = ?; };` simulates the instruction $x \leftarrow [0, 10]$, only handled by Miné's analyser. Exiting the loop, the measures for interval and octagon are exactly the same, and the runs (presented below the code) predicted well this similarity.

`(4,5)` Conditional `x <= y` splits the points coming out of the control point `(2)` in two parts, which is easily noticeable on the graphical output. This test enforces the relationship $x \leq y$ between the two variables. The measure for interval and octagon differs for these two points. The measure of the experiments also differs, but in a smaller proportion.

`(6)` Assignment, `y = x` enforces a strong relation between the two variables, which is not well captured by the interval abstract domain. The measure for the octagons, in the effective analysis as well as in the experiment, drops to 0, while the interval over-approximation is indicated by the non-null measure.

`(3)` After the join, the octagon measure is about half the measure of the intervals. The measures render in this way the fact that this fragment of code build a kind of triangle, whose size is over-approximated by the interval analysis to the size of its rectangular hull.

In this example, the precision predictions turn out to be close to reality. We provide a second example, where the random testing does not succeed as well, but even then, interesting conclusions can be drawn.

The second example, whose prototype textual output can be seen in Table 3, is also taken from [Min04, ex254]. It involves a loop whose number of iterations is defined by a parameter `n`. The settings for the prototype are the same as in the first example: the median of the exponential distribution is set to 10 and we only measure the positive part of the plan. The loop invariant includes a linear relationship between `x` and `n` that can be discovered by the octagon domain, but cannot be expressed by a value of the interval domain. The control point for which the difference is the most obvious is `(1)`. In fact, this control point is unreachable, and the octagon domain is able to infer this property. The experiments do not manage to predict this difference because obviously no run came at control point `(1)`. However, the difference of precision in `(5)` between the interval analysis and the octagon analysis, that allow the octagon analysis to conclude the non-reachability of control point `(1)` is detected by the executions.

The precision prediction, in these simple yet interesting cases, provide accurate results if we set aside probabilistically unreachable points. This problem is shared by most of the random approaches to program verification. The cost of the prediction is low. Random executions are stopped after a given delay. The computation of the precision in two dimensions is negligible in time. The convex

```
            // ((0)) Measure (int vs oct) = 1.00 <> 1.00;
            // interval shape = y = [-oo,+oo]  x = [-oo,+oo]  ;
            // octogon shape = {  }
x = ?; while (x < 0 or x > 10) { x = ?; };
            // ((2)) Measure (int vs oct) = 0.50 <> 0.50;
            // interval shape = y = [-oo,+oo]  x = [ 0 , 10]  ;
            // octogon shape = { 0 <= x <= 10 }
if x <= y {
            // ((5)) Measure (int vs oct) = 0.50 <> 0.38;
            // interval shape = y = [ 0 ,+oo]  x = [ 0 , 10]  ;
    // octogon shape = { 0 <= x <= 10, y >= 0, y-x >= 0, y+x >= 0 }
  y = x;
            // ((6)) Measure (int vs oct) = 0.25 <> 0.00;
            // interval shape = y = [ 0 , 10]  x = [ 0 , 10]  ;
    // octogon shape = { 0 <= x <= 10, 0 <= y <= 10,
    //                   y-x = 0, 0 <= y+x <= 20 }
} else {
            // ((4)) Measure (int vs oct) = 0.23 <> 0.10;
            // interval shape = y = [-oo, 9 ]  x = [ 0 , 10]  ;
    // octogon shape = { 0 <= x <= 10, y <= 9, y-x <= -1, y+x <= 19 }
};
            // ((3)) Measure (int vs oct) = 0.25 <> 0.12;
            // interval shape = y = [-oo, 10]  x = [ 0 , 10]  ;
    // octogon shape = { 0 <= x <= 10, y <= 10, y-x <= 0, y+x <= 20 }
Results of the runs:
2 variables, 9 program points
pt (0) -> int vs oct = 0.95 <> 0.92; distinct hits = 50
pt (2) -> int vs oct = 0.49 <> 0.48; distinct hits = 50
pt (5) -> int vs oct = 0.43 <> 0.35; distinct hits = 39
pt (6) -> int vs oct = 0.24 <> 0.00; distinct hits = 39
pt (4) -> int vs oct = 0.13 <> 0.10; distinct hits = 11
pt (3) -> int vs oct = 0.24 <> 0.12; distinct hits = 50
```

Table 2. Example 2.5.1, text output.

```
x = 0;
while x <= n {
            // (5) Measure (int vs oct) = 1.00 <> 0.50;
            // interval shape = n = [ 0 ,+oo]  x = [ 0 ,+oo]  ;
            // octogon shape = { x >= 0, n >= 0, n-x >= 0, n+x >= 0 }
  if (x < 0 or n < x) {
            // (1) Measure (int vs oct) = 0.93 <> 0.00;
            // interval shape = n = [ 0 ,+oo]  x = [ 1 ,+oo]  ;
            // octogon shape = (empty)
  } else {
            // (2) Measure (int vs oct) = 1.00 <> 0.50;
            // interval shape = n = [ 0 ,+oo]  x = [ 0 ,+oo]  ;
            // octogon shape = { x >= 0, n >= 0, n-x >= 0, n+x >= 0 }
  };
            // (4) Measure (int vs oct) = 1.00 <> 0.50;
            // interval shape = n = [ 0 ,+oo]  x = [ 0 ,+oo]  ;
            // octogon shape = { x >= 0, n >= 0, n-x >= 0, n+x >= 0 }
  x = x + 1;
            // (3) Measure (int vs oct) = 0.93 <> 0.47;
            // interval shape = n = [ 0 ,+oo]  x = [ 1 ,+oo]  ;
            // octogon shape = { x >= 1, n >= 0, n-x >= -1, n+x >= 1 }
};
            // (6) Measure (int vs oct) = 1.00 <> 0.47;
            // interval shape = n = [-oo,+oo]  x = [ 0 ,+oo]  ;
            // octogon shape = { x >= 0, n-x <= -1 }
Results of the runs:
2 variables, 8 program points
pt 5 -> int vs oct = 0.96 <> 0.49; distinct hits = 399
pt 1 unreached
pt 2 -> int vs oct = 0.96 <> 0.49; distinct hits = 386
pt 4 -> int vs oct = 0.96 <> 0.49; distinct hits = 386
pt 3 -> int vs oct = 0.90 <> 0.46; distinct hits = 381
pt 6 -> int vs oct = 0.83 <> 0.03; distinct hits = 26
```

Table 3. Example 2.5.4, text output.

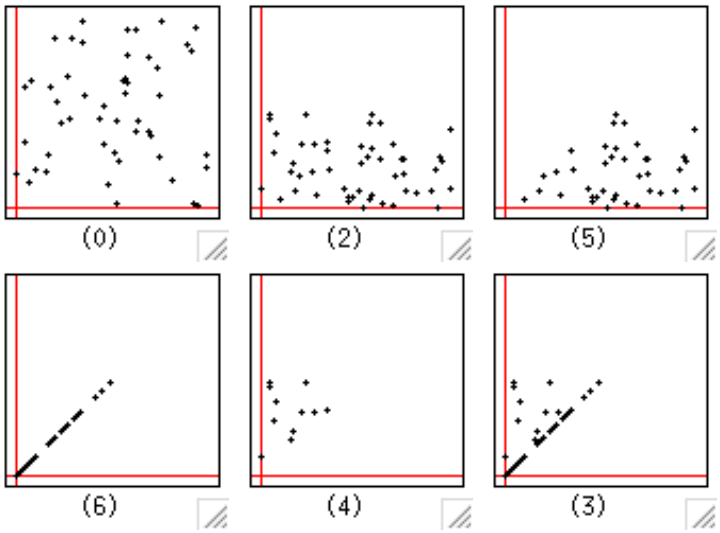


Fig. 6. Example 2.5.1, visual outputs.

hull computation is in nk for the intervals and nk^2 for the octagons, where n is the number of points and k the number of constraint of the structure.

A problem would appear if we apply naively the same method for extending the prototype to the polyhedra abstract domain. Indeed, the set of point is random and would generate a convex hull with a huge number of constraints (w.r.t. a value obtained by static analysis). Legay et al. [CLW07] propose a method, based on automata, that build a membership predicate without computing the convex hull. Such a predicate is sufficient for the Monte-Carlo algorithm presented before.

5 Opened Perspectives

This section contains some motivating fields of application, in 5.1. It also lists, in 5.2, interesting questions that arise when applying our precision in a broader context. We mention researches that might be part of the answers.

5.1 Field of Application

This subsection present ideas exploiting our notion of precision for static analysis purpose. Most of the applications we considered used the precision as a tool to compare two different abstract values. The framework allows this comparisons within the abstract domain layer, at analysis level or even between distinct analysers.

Keeping Track of Precision Loss. Analysers, within iterative or conditional structures, use the least upper bound operator, that introduces imprecision. Furthermore, to enforce the termination of the Kleene iteration, they rely on the widening operator, that also causes imprecision. Approximations may also appear with efficient but non-optimal operators or with linear approximation of non-linear properties.

The idea is to compute the imprecision added at each point and to keep track of it in order to inform the user. When the property is supposed to hold but cannot be proved, the information on where comes from the imprecision can be valuable to tune the analyser.

For example, the join imprecision can be computed this way:

$$pre_\mu(a \sqcup b) - \left(pre_\mu(a) + pre_\mu(b) - pre_\mu(a \sqcap b)\right).$$

This value reflects the quantity of false positive added when joining the abstract values a and b.

Disjunctive Abstract Domains. Some abstract domains are expressed as disjunctions of values from other abstract domains (e.g. [PC06]). In an analysis that manipulate sets of polyhedra, the analyser maintains these sets to a reasonable size by applying fusions of polyhedra.

The (classic) idea is to select the polyhedra to be joined by a minimal loss of precision, as a quantitative alternative to more qualitative or heuristic approaches. An analyser that refines its abstract values by partitioning them faces the same kind of problem.

Domain Selection Heuristic. Several domains may be dedicated to prove the same kind of properties, but at different trade-off between precision and computation cost. They may be grouped in libraries of numerical abstract domain, like the academic state-of-art APRON [JM09]. Their common interface is favourable to dynamic domain selection and comparison.

The way we measure the precision of an abstract value, through the measure of its concretisation allows to compare abstract values from distinct abstract domains. The idea, experimented on simple domains in Sect. 4, is then to use this comparison to decide whether the gain of precision is worth the complexity increase.

In addition, not all abstract domains are linked by a qualitative relation of precision. The zonotope abstract domain, added to APRON by Ghorbal, Goubault and Putot [GGP09], is qualitatively neither more nor less precise than the octagon abstract domain. Our proposal may be a tool to compare them.

5.2 Opened Questions

This subsection examines problems that might appear in some applications relying on our precision. These problems may be more or less critical depending on the intended use of the precision. To the best of our knowledge, these problems are neither clearly solved nor impossible to solve.

Degenerated Polyhedra. In static analysis, when the number of dimension of a polyhedra is high, it is probable that the polyhedra will not be fully dimensional, i.e. some equalities make it flat somewhere. In this case, its measure is 0, and the comparison of two 0-precision polyhedra provides no information. In such cases, the difference should be made at a sublevel, as in a lexicographic ordering.

Solutions may come from [JR06], where Jahier and Raymond are interested by randomly picking a point in a polyhedra (for random testing purpose). However, they do not have efficient solution to randomly pick in a degenerated polyhedra according to the uniform distribution. In [HMG06], Halbwachs, Merchat and Gonnord face the problem of reducing the number of dimensions of a polyhedra. Their solutions might also be solutions for the problem we mentioned.

Measure Defined by Probability Distribution. In this paper, we favoured the analytical and exact measure of precision. We mentioned the Monte-Carlo alternative but did not implemented it. The Monte-Carlo approach has for drawback that the result of the measure is non-deterministic, which is looked with suspicion in the domain of static analysis. The advantage of the approach is to let a huge freedom in the choice of the measure, given that all you need is a probability distribution to generate values for your variables.

In static analysis, non-determinism in result is a pretty bad option. A way to cope with it, without ad-hoc tricks on pseudo-random generation, is to turn this non-deterministic result in non-deterministic execution time, for applications with backtracking. Another approach, defended by Sumit Gulvani in his PhD thesis [Gul05], is to trade soundness for probabilistic soundness, ensuring that the odds of being unsound will be (arbitrarily) small.

We now consider the advantages provided by more flexibility in the measure. When choosing the measure, the user can tell what are his or her interests. We can imagine combining a uniform distribution on a given interval with a distribution handling infinite shapes for the remainder of the space. It is intuitive for the user to understand that what will be most valued in the measure will simply be the most probable part of the space. Another benefit of a wider choice of probability distribution is the possibility to pick a distribution stable by rotation (exponential is not), which could be a way do deal with the degenerated polyhedra problem mentioned before.

6 Related Works

The goal of quantifying the loss of precision due to an abstract domain is shared by Logozzo, Popeea and Laviron [LPL09]. They do not rely on the notion of measure (on the concrete domain) but on the notion of (pseudo-)distance in a lattice (the abstract one). Like us, this mathematical foundation allow them to pick an instance well-suited for the abstract domains they deal with. They argue that the pseudo-distance called *affinity*, already used by Popeea and Chin in [PC06] for disjunctive abstract domain, is well-adapted to numerical abstract

lattices. This pseudo-distance relies on the minimal number of linear constraints needed to encode an abstract value. In their framework, $[0, 3]$ is more precise than $[0, 2]$, thanks to constraint entailment, but $[0, 2]$ is as precise as $[1, 10]$. Our quantification is finner since it notices that $[0, 2]$ is "shorter" than $[1, 10]$.

In [DSW08], Di Pierro, Wiklicky and Sotin investigate the notion of precision in a framework where the program probabilistic semantics, as well as abstraction and concretisation functions are modelled by linear operators. In this context, the notion of precision derives from the norm of a particular linear operator. However, the method used to compute this precision relies on the finiteness of the abstract domain, so it cannot address the problems (in $\mathbb{R}^n$) we deal with.

The counting measure, that we qualified of "simple" may be used for finite concrete domains. In [RKG04], Rountev, Kagan and Gibas uses the cardinalities to compare a set of static class analyses for Java. For a given program, the result of each analysis is compared with the exact value. $|r^\sharp|/|r|$ gives the percentage of false positives, with r being the exact set of valid properties and $r^\sharp$ the concretisation of the result of the analysis. The problem with this approach is the fact that the exact result is not computable, so at some point, a set of proofs of feasibility or infeasibility has to be done manually. The proposition is valuable when trying to demonstrate the superiority of some analysis w.r.t. some competitor. However, the method cannot be made fully automatic, and furthermore, even the proofs by hand would not be tractable with abstract domains containing as many properties as the numerical domains.

The quantitative notions of precision have to relate with the qualitative one. Details on both qualitative and quantitative notions of precision for abstract domains can be found in Pascal Sotin's PhD thesis [Sot08, p26]. In [Cou02], Patrick Cousot builds a hierarchy of concrete semantics (e.g. traces, small-step, denotational). It implies a qualitative relation of precision, not at domain level but at program semantics level. In [Sch06], David Schmidt details the different notion of *completeness* appearing in static analyses, which can be seen as the qualitative notion of maximal precision.

7 Conclusion

In this paper, we defined a way to evaluate quantitatively the precision within any numerical abstract domain. The precision of an abstract value is defined as the mathematical measure of its concretisation. We argue that the exponential measure is well-suited for convex abstract domains, like intervals, octagons, polyhedra, zonotopes. We provide algorithms to effectively compute this precision. We provide some exact algorithms obtained analytically and a general Monte-Carlo-based approximated algorithm.

We believe that our precision measure is a tool having many potential applications in static analysis and being intuitive for its users. We studied the idea of using the precision together with random testing to guide the selection of an adequate abstract domain. We developed a prototype and commented its results.

We also suggest other applications and list interesting problems that some of these applications will have to face.

References

[CC77] Patrick Cousot and Radhia Cousot. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *4th Symposium on Principles of Programming Languages (POPL'77)*, pages 238–252, january 1977.

[CH78] Patrick Cousot and Nicolas Halbwachs. Automatic Discovery of Linear Restraints Among Variables of a Program. In *Proceedings of POPL'78*, pages 84–96, 1978.

[CLW07] Franois Cantin, Axel Legay, and Pierre Wolper. Computing Convex hulls by automata iterations. In *AUTOMATHA*, 2007.

[Cou02] Patrick Cousot. Constructive Design of a Hierarchy of Semantics of a Transition System by Abstract Interpretation. *Theoretical Computer Science*, 277(1–2):47–103, 2002.

[DSW08] Alessandra Di Pierro, Pascal Sotin, and Herbert Wiklicky. Relational Analysis and Precision via Probabilistic Abstract Interpretation. In *QAPL08, Quantitative Aspects of Programming Languages*, 2008.

[Fis96] George S. Fishman. *Monte Carlo: Concepts, Algorithms and Applications*. Springer, 1996.

[GGP09] Khalil Ghorbal, Eric Goubault, and Sylvie Putot. The Zonotope Abstract Domain Taylor1+. In *Computer Aided Verification (CAV'09)*, volume 5643 of *LNCS*, July 2009.

[Gul05] Sumit Gulvani. *Program Analysis using Random Interpretation*. PhD thesis, UC-Berkeley, 2005.

[HMG06] Nicolas Halbwachs, David Merchat, and Laure Gonnord. Some Ways to Reduce the Space Dimension in Polyhedra Computations. *Form. Methods Syst. Des.*, 29(1):79–95, 2006.

[JM09] Bertrand Jeannet and Antoine Miné. APRON: A library of numerical abstract domains for static analysis. In *Computer Aided Verification (CAV'09)*, volume 5643 of *LNCS*, pages 661–667, July 2009.

[JR06] Erwan Jahier and Pascal Raymond. Generating random values using Binary Decision Diagrams and Convex Polyhedra. In *Workshop on Constraints in Software Testing, Verification and Analysis (CSTVA'06)*, September 2006.

[LPL09] Francesco Logozzo, Corneliu Popeea, and Vincent Laviron. Towards a Quantitative Estimation of Abstract Interpretations. In *Workshop on Quantitative Analysis of Software*. Microsoft, June 2009.

[Min04] Antoine Miné. *Weakly Relational Numerical Abstract Domains*. PhD thesis, École Normale Supérieure, Paris, 2004.

[PC06] Corneliu Popeea and Wei-Ngan Chin. Inferring Disjunctive Postconditions. In *11th Asian Computing Science Conference (ASIAN'06)*, Tokyo, December 2006.

[Pic05] David Pichardie. *Interprétation abstraite en logique intuitionniste : extraction d'analyseurs Java certifiés*. PhD thesis, Université Rennes 1, 2005.

[RKG04] Atanas Rountev, Scott Kagan, and Michael Gibas. Evaluating the Imprecision of Static Analysis. In *ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*, pages 14–16, 2004.

[Sch06] David A. Schmidt. Comparing Completeness Properties of Static Analyses and their Logics. In N. Kobayashi, editor, *Asian Programming Languages and Systems Symposium (APLAS'06)*, volume 4279 of *LNCS*, pages 183–199. Springer, 2006.

[Sot08] Pascal Sotin. *Aspects quantitatifs de l'analyse de programmes / Quantitative Aspects of Static Analysis*. PhD thesis, Université de Rennes 1, december 2008.