The Zonotope Abstract Domain Taylor1+[⋆]

Khalil Ghorbal, Eric Goubault, and Sylvie Putot

CEA, LIST, Modelisation and Analysis of Systems in Interaction,
F-91191 Gif-sur-Yvette Cedex, France
`firstname.surname@cea.fr`

1 Introduction

Static analysis by abstract interpretation [1] aims at automatically inferring properties on the behaviour of programs. We focus here on a specific kind of numerical invariants: the set of values taken by numerical variables, with a real numbers semantics, at each control point of a program.

We present an implementation called Taylor1+, interfaced with the APRON library [2], of an abstract domain using affine forms [3], defined by E. Goubault and S. Putot in [4, 5].

Contributions and organisation of the paper. We recap, in Section 2 the semantics of the main operations implemented, both arithmetic and order-theoretic. We then explain in Section 3 how this real number semantics is implemented using finite precision arithmetic. We finally present, in Section 4, experimental results, which we compare to the results obtained with other domains of APRON, such as intervals [1], octagons [6] and polyhedra [7] abstract domains.

Related work. Geometrically, the representation of the abstract values (the joint range of all variables) in our domain is a center-symmetric polytope called zonotope. Zonotopes were successfully applied elsewhere, such as for reachability analysis in the model-checking of hybrid systems [8] or collision detection [9].

2 Abstract Domain Based on Affine Forms

Affine arithmetic [3] is an extension of Interval Arithmetic that keeps track of affine relations between values of variables. An affine form expresses a set of values as a central value plus a sequence of deviation terms over symbolic symbols, called noise symbols. Formally, the affine form, $\hat{x}$, describing the values that program variable x can take, is :

$$\hat{x} = \alpha_0^x + \sum_{i=1}^{n} \alpha_i^x \epsilon_i$$

where the real coefficients $(\alpha_i^x)_{1 \le i \le n}$ are the partial deviations and the noise symbols $(\epsilon_i)_{1 \le i \le n}$ have their unknown values within $[-1, 1]$. Its interval concretisation is

⋆ This work was partially funded by the ANR project ASOPT.

$$\gamma(\hat{x}) = \left[\alpha_0^x - \sum_{i=1}^{n} |\alpha_i^x|, \alpha_0^x + \sum_{i=1}^{n} |\alpha_i^x|\right]$$

These noise symbols are introduced dynamically: *ı)* for each new input whose value is given in an interval, or *ıı)* when non linear operations are achieved (see section 2.1). For instance, the affine form abstraction of a new program variable x known to lie in $[a, b]$, is $\hat{x} = \frac{1}{2}(a+b) + \frac{1}{2}(b-a)\epsilon_{n+1}$, where ϵ_{n+1} is a fresh noise symbol, i.e. is not used by any existing affine form.

Abstract operations must be *sound*, that is, in our case, *ı)* give guaranteed range over-approximations of the variables at the current control point of the program, and *ıı)* give guaranteed range over-approximations for all other expressions on these variables that we might want to evaluate later (see [5]).

2.1 Arithmetic Operations

If performed with real number coefficients, affine arithmetic is exact on linear operations : addition and subtraction operations are defined componentwise. For non-linear unary operations, such as square root and inverse, our implementation relies on Taylor forms of first order with rigorous error bounds for the error term. For non-linear binary operations, an approximated affine form is computed, and the remaining non-linear term is bounded, then assigned to a fresh noise symbol. For instance, the multiplication of two affine forms $\hat{x} = \alpha_0^x + \sum_{i=1}^{n} \alpha_i^x \epsilon_i$ and $\hat{y} = \alpha_0^y + \sum_{i=1}^{n} \alpha_i^y \epsilon_i$ is :

$$\hat{x} \times \hat{y} = \alpha_0^x \alpha_0^y + \sum_{i=1}^{n} (\alpha_i^x \alpha_0^y + \alpha_i^y \alpha_0^x)\epsilon_i + \sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i^x \alpha_j^y \epsilon_i \epsilon_j \ . \tag{1}$$

We have implemented the method of [4]: $\epsilon_i\epsilon_j$ is taken within $[0, 1]$ whenever $i = j$, and $[-1, 1]$ otherwise. The method is cost-effective but not always the most precise one. A more accurate but more costly technique is to use SemiDefinite Programming (SDP) :

$$\max_{|\epsilon_i|\le 1} \sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i^x \alpha_j^y \epsilon_i \epsilon_j = \max_{|\epsilon_i|\le 1} \varepsilon^t.\Phi.\varepsilon \le \inf_{\mu \in \mathbb{R}_+^n} \{\text{trace}(\mu I_n) | \Phi - \mu I_n \preceq 0\} \tag{2}$$

where $(\phi_{i,j})_{1\le i,j\le n} = \frac{1}{2}(\alpha_i^x \alpha_j^y + \alpha_j^x \alpha_i^y)$ and $M \preceq 0$ means that matrix M is negative semidefinite. The equality holds when matrix Φ is negative semidefinite. The right hand side of (2) is a typical SDP problem. We give first experimental results in section 4.

2.2 Order-Theoretic Operations

Perturbed Affine Forms defined in [5] extend standard affine forms by adding special noise symbols ϵ_U, called join symbol, that allow simple and precise order-theoretic operations We define then exemplify the (pseudo) join operation.

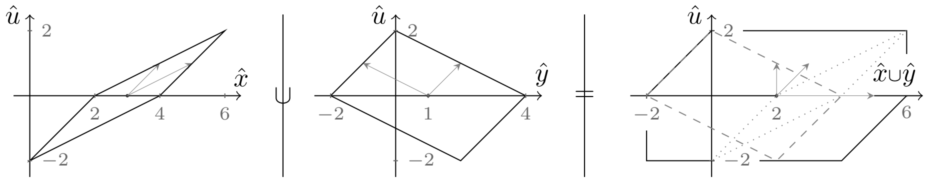
Definition 1. *The join operation $\hat{z} = \hat{x} \cup \hat{y}$ defines an upper bound of $\hat{x}$ and $\hat{y}$, which is minimal in "generic" situations, and whose interval concretisation is the union of interval concretisations of $\hat{x}$ and $\hat{y}$:*

$$\begin{array}{ll}
\alpha_0^z = mid(\gamma(\hat{x}) \cup \gamma(\hat{y})) & \textit{(central value of } \hat{z}\textit{)} \\
\alpha_i^z = \underset{min(\alpha_i^x, \alpha_i^y) \le \alpha \le max(\alpha_i^x, \alpha_i^y)}{\operatorname{argmin}} (|\alpha|), \forall i \ge 1 & \textit{(coeff. of } \epsilon_i\textit{)} \\
\beta^z = \sup(\gamma(\hat{x}) \cup \gamma(\hat{y})) - \alpha_0^z - \sum_{i \ge 1} |\alpha_i^z| & \textit{(coeff. of } \epsilon_U\textit{)}
\end{array}$$

where the γ function returns the interval concretisation of an affine form and $mid([a,b]) := \frac{1}{2}(a+b)$ and $\underset{a \le x \le b}{\operatorname{argmin}}(|x|) := \{x \in [a,b], |x|$ is minimal $\}$.

Example 1. By the formula of definition 1:

$$\begin{pmatrix} \hat{x} = 3 + \epsilon_1 + 2\epsilon_2 \\ \hat{u} = 0 + \epsilon_1 + \epsilon_2 \end{pmatrix} \cup \begin{pmatrix} \hat{y} = 1 - 2\epsilon_1 + \epsilon_2 \\ \hat{u} = 0 + \epsilon_1 + \epsilon_2 \end{pmatrix} = \begin{pmatrix} \hat{x} \cup \hat{y} = 2 + \epsilon_2 + 3\epsilon_U \\ \hat{u} \cup \hat{u} = 0 + \epsilon_1 + \epsilon_2 \end{pmatrix}$$



We also define the cyclic unfold, denoted by $(i, c, \mathcal{N})$, as the one obtained by initially unrolling i times the loop, and from then computing the fixpoint of the loop functional iterated c times until convergence, this with at most $\mathcal{N}$ iterations, after which a classical interval semantics is used [1]. As proved in [5], and shown in Section 4, the cyclic unfold schemes together with the join operator ensures termination with accurate fixpoint bounds for linear iterative schemes.

3 Implementation Aspects

The APRON Project [2] provides a uniform high level interface for numerical domains. For the time being, intervals, convex polyhedra, octagons, and congruences abstract domains are interfaced. We enrich here the library with a domain based on affine forms, called Taylor1+.

As we represent coefficients of affine forms by double precision floating-point numbers instead of real numbers, we have to adapt our transfer functions. For instance, instruction `z = x + y;` is abstracted by

$$\hat{z} = \hat{x} \oplus \hat{y} = float(\alpha_0^x + \alpha_0^y) + \sum_{i=1}^{n} float(\alpha_i^x + \alpha_i^y)\epsilon_i + \left(\sum_{i=0}^{n} dev(\alpha_i^x + \alpha_i^y) \right) \epsilon_{n+1}$$

where $float(x)$ is the nearest double-precision floating-point number to the real number x and $dev(x) := \triangleright(|x - float(x)|)$, ($\triangleright$ being rounding towards $+\infty$).

We are working on some techniques, namely those used in [8] and [10], to control the potential increase of the number of noise symbols during analysis.

However, in practise, the number of symbols reaches high levels very scarcely, since our join operator has the effect of reducing the number of noise symbols by collapsing some of them into a join symbol.

4 Experiments and Benchmarks

We analyse hereafter two simple iterative schemes. We used a laptop equipped with Intel(R) Core(TM)2 CPU (1.06GHz) and 2GB of RAM. All numerical values are rounded to two significant decimal digits for readability's sake.

4.1 Linear Iterative Schemes

Consider the following 2^{nd} order filter :

$$S_n = 0.7E_n - 1.3E_{n-1} + 1.1E_{n-2} + 1.4S_{n-1} - 0.7S_{n-2}$$

where E_n are independent inputs with unknown values in range $[0, 1]$, and S_n is the output of the filter at iteration n. Pôles are inside the unit circle (norm close to 0.84), so the output in real numbers is provably bounded, and can be tightly estimated by manual methods to $[-1.09, 2.75]$. We also study a 8^{th} order linear recursive digital filter used in an industrial test case (whose code is omitted for obvious reasons), whose output is provably bounded in $[-0.20, 1.20]$.

Unrolled schemes. We first fully unroll the 2^{nd} order filter scheme to compute the abstract value at each iteration. Figure 1 compares accuracy and performance of Taylor1+ with three domains, provided in APRON: Boxes (Interval Analysis), Octagons, Polyhedra (both PK [11] and PPL [12] implementations were tested). The current version of the Octagonal domain does not integrate any of the symbolic enhancement methods of [13], which leads to inaccurate results. The Polyhedra domain with exact arithmetic (using GMP) gives the exact bounds for the filter output. One can see that Taylor1+ wraps very closely the exact range given by polyhedra (left figure) with great performance (right figure).

Fixpoint computation using Kleene-like iteration. For both filters, we detail results for two different $(i, c, \mathcal{N})$-iteration schemes (see end of Section 2.2) for Taylor1+, with $i = 0$ and $\mathcal{N} = 10^3$. Table 1 summarizes the results, for the 2^{nd} order filter (left tables) and the 8^{th} order filter (right tables). For boxes, octagons, and polyhedra domains, their respectively classical widening operator were used if a fixpoint is not reached after 100 iterations. For T1+ domain, beyond this threshold, i.e. 100, and before $\mathcal{N}$, we accelerated convergence of the fixpoint computation by only keeping noise terms with equal coefficients and collapsing all the others.

Since the output diverges for Boxes and Octagons domains, the fixpoint computation diverges as well. Polyhedra gives the least fixpoint in a short time for the second order filter, however it takes an enormous amount of time for the filter of order 8, so we aborted computation. For the 2^{nd} (resp. 8^{th}) order filter, the

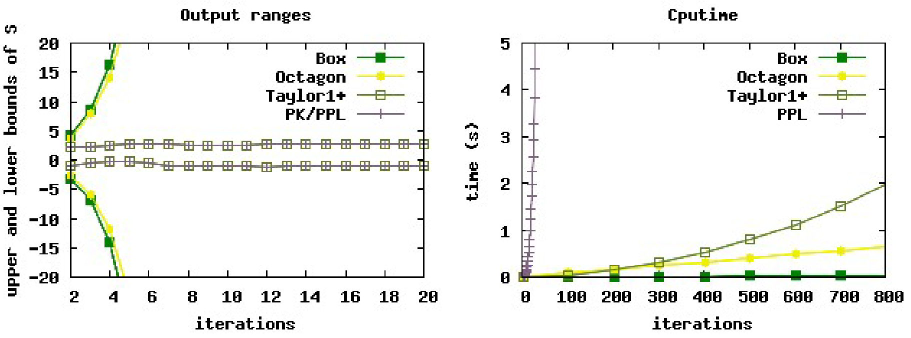


Fig. 1. Unrolled scheme for the 2^{nd} order filter

Table 1. Fixpoint computation (2^{nd}&8^{th}o filters) using Kleene-like iteration technique

filter o2	fixpoint	t(s)
Boxes	$\top$	6×10^{-3}
Octagons	$\top$	0.19
Polyhedra	$[-1.30\ ,\ 2.82]$	0.49
T.1+(5)	$[-8.90\ ,\ 10.57]$	0.1
T.1+(20)	$[-5.40\ ,\ 7.07]$	0.2

filter o8	fixpoint	t(s)
Boxes	$\top$	0.01
Octagons	$\top$	21
Polyhedra	abort	$>24h$
T.1+(5)	$[-19.77\ ,\ 20.77]$	0.74
T.1+(20)	$[-3.81\ ,\ 4.81]$	0.5

fixpoint reached in Taylor1+ for the scheme $c = 20$ (i.e. for the loop functional iterated c times) is $[-1.18, 2.84]$ (resp. $[-0.27, 1.27]$). From there, the computation of the fixpoint of the loop is slightly wider: $[-5.40, 7.07]$ (resp. $[-3.81, 4.81]$); we are working on improvements.

4.2 Non-linear Iterative Scheme

The non-linear scheme we are considering is based on a Householder method of order 3 that converges towards the inverse of the square root of an input A. It originates from an industrial code, used as a test case in [14]; The current estimate of the inverse of the square root is updated as follows:

$$x_{n+1} = x_n + x_n\left(\frac{1}{2}h_n + \frac{3}{8}h_n^2\right)$$

where $h_n = 1 - Ax_n^2$, $A \in [16, 20]$ and $x_0 = 2^{-4}$.

We study the fully unrolled scheme for 5 iterations, then the fixpoint computation by the $(5, 1, 10^3)$-iteration scheme for Taylor1+, and compare different implementations of the multiplication; results are shown in Table 4.2. We compute here all possible values, whatever the stopping criterion (on $\mid x_{n+1} - x_n \mid$) of the loop may be. The fixpoint of $\sqrt{A}$ (right table), deduced from this estimate, thus encloses the first 5 iterations and is hence naturally wider than the result

Table 2. Comparison of domains on Householder (o3) example

Unrolling (5 It.)	$\sqrt{A} = Ax_n$	t(s)
Boxes	[0.51 , 8.44]	1×10^{-4}
Octagons	[0.51 , 7.91]	0.01
Polyhedra	[2.22 , 6.56]	310
T.1+ :	[3.97 , 4.51]	1×10^{-3}
• 10 subdivisions	[4.00 , 4.47]	0.02
• SDP	[3.97 , 4.51]	0.16

Kleene Iteration	$\sqrt{A} = Ax_n$	t(s)
Boxes	$\top$	1×10^{-4}
Octagons	$\top$	0.04
Polyhedra	abort	$>24h$
T.1+ :	[1.80 , 4.51]	0.01
• 10 subdivisions	[1.80 , 4.48]	0.2
• SDP	[1.80 , 4.51]	0.86

of the unrolled scheme. We can see that results are tight even for non linear computations. The SDP solver is costly in time and does not seem to buy much more precision. However, for a larger range for input A, SDP gives tighter results than the standard multiplication. Moreover, the real advantage of SDP over subdviding is that the process of subdividing inputs might become intractable when several inputs would need subdividing. We tested here a non-guaranteed SDP solver [15], but we plan in the future to use guaranteed SDP solver such as the one described in [16].

References

[1] Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: ACM POPL 1977, pp. 238–252 (1977)

[2] APRON Project. Numerical abstract domain library (2007), `http://apron.cri.ensmp.fr`

[3] Comba, J.L.D., Stolfi, J.: Affine arithmetic and its applications to computer graphics. In: SIBGRAPI 1993 (1993)

[4] Goubault, E., Putot, S.: Static analysis of numerical algorithms. In: Yi, K. (ed.) SAS 2006. LNCS, vol. 4134, pp. 18–34. Springer, Heidelberg (2006)

[5] Goubault, E., Putot, S.: Perturbed affine arithmetic for invariant computation in numerical program analysis (2008), `http://arxiv.org/abs/0807.2961`

[6] Miné, A.: The Octagon abstract domain. Higher-Order and Symbolic Computation, 31–100 (2006)

[7] Cousot, P., Halbwachs, N.: Automatic discovery of linear restraints among variables of a program. In: ACM POPL 1978, pp. 84–97 (1978)

[8] Girard, A.: Reachability of uncertain linear systems using zonotopes. In: Morari, M., Thiele, L. (eds.) HSCC 2005. LNCS, vol. 3414, pp. 291–305. Springer, Heidelberg (2005)

[9] Guibas, L.J., Nguyen, A., Zhang, L.: Zonotopes as bounding volumes. In: Symposium on Discrete Algorithms, pp. 803–812 (2003)

[10] Kühn, W.: Zonotope dynamics in numerical quality control. In: Mathematical Visualization, pp. 125–134. Springer, Heidelberg (1998)

[11] Jeannet., B., et al.: Newpolka library, `http://www.inrialpes.fr/pop-art/people/bjeannet/newpolka`

[12] PPL Project. The Parma Polyhedra Library, `http://www.cs.unipr.it/ppl/`

[13] Miné, A.: Symbolic methods to enhance the precision of numerical abstract domains. In: Emerson, E.A., Namjoshi, K.S. (eds.) VMCAI 2006. LNCS, vol. 3855, pp. 348–363. Springer, Heidelberg (2005)

[14] Goubault, E., Putot, S., Baufreton, P., Gassino, J.: Static analysis of the accuracy in control systems: Principles and experiments. In: FMICS (2007)
[15] Borchers, B.: A C library for Semidefinite Programming (1999), `https://projects.coin-or.org/Csdp`
[16] Jansson, C., Chaykin, D., Keil, C.: Rigorous error bounds for the optimal value in semidefinite programming. SIAM J. Numer. Anal. 46(1), 180–200 (2007)