Static Analysis of Digital Filters [1]

Jérôme Feret

Département d'Informatique de l'École Normale Supérieure,
75230 PARIS Cedex 5, FRANCE

Abstract

We present an Abstract Interpretation-based framework for automatically analyzing programs containing digital filters. Digital filtering consists in implementing numerical recursions: the value of a variable S (the output) is computed from a fixed finite number of the last consecutive values of the variable S and from a fixed finite number of the last consecutive values of another variable E (the input). Our framework allows us to refine existing analyses so that they can handle given classes of digital filters. We only have to design a class of symbolic properties that describe the invariants throughout filter iterations and to describe how these properties are transformed by filter iterations. Then, the analysis allows both inference and proofs of the properties about the program variables that are tied to any such filter. In case of linear filters, we propose a systematic method for designing the abstract domain by using interval and elliptic constraints.

Key words: Abstract Interpretation, symbolic domains, numerical domains, digital filtering, synchronous systems, critical systems, embedded systems.

1 Introduction

Digital filters are widely used in real-time embedded systems (as found in automotive, aeronautic, and aerospace applications) since they allow modeling behaviors previously ensured by analogical filters into software . A filter transforms an input stream of floating-point values into an output stream. Existing analyses are very imprecise in bounding the range of the output stream, because of the lack of precise linear properties that would entail that the output is bounded. The lack of precise domains when analyzing digital filters was indeed the cause of almost all the remaining warnings (potential floating-point

[1] This work was partially supported by the ASTRÉE RNTL project.

overflows) in the certification of a critical software family with the ASTRÉE analyzer [1, 2, 7].

In this paper, we propose an Abstract Interpretation-based framework for designing new abstract domains which handle filter classes. Human intervention is required for discovering the general shape of the properties that are required in proving the stability of such a filter. Roughly speaking, filter properties are mainly an abstraction of the input stream, from which we deduce bounds on the output stream. Our framework can be used to build such abstract domains and to propagate all the abstract properties throughout the abstract computations of programs. Our approach is not syntactic, so that loop unrolling, filter reset, boolean control, and trace (or state) partitioning are dealt with for free and any filter of the class (for any setting) is analyzed precisely.

Given a generic form of recursive sequence (such as $S_{n+2} = aS_{n+1} + bS_n + cE_{n+2} + dE_{n+1} + eE_n$) computed in the floating-point world, an abstract property relates some variables (here S_{n+2}, S_{n+1}, E_{n+2}, and E_{n+1}) to some abstract values. Such a property means that, up to rounding errors, the variables S_{n+2} and S_{n+1} are associated with two consecutive values of the recursive sequence while the variables E_{n+2} and E_{n+1} are associated with the last two input values (the input value E_n is not relevant, since it will not be used in the next iteration of the filter). The abstract domain captures an abstraction of the input stream (E_n), initial conditions S_0 and S_1, and the overall contribution of rounding errors. The most difficult part lies in refining interval constraints using filter constraints.

We give a systematic method to build abstract domains for analyzing linear filters. Formally, the output can be split into three summands: the contribution of the last N inputs if the digital filter were computed in the real field, the contribution of the initial outputs and of the other inputs if the digital filter were computed in the real field, and the difference between the result of the computation in the floating-point world and the result of the computation in the real field. The integer N is a parameter of the approximation. The first summand can be symbolically computed as a known function from the last input values into the real field. The second and the third summands both satisfy the simplified recursion $S'_{n+2} = aS'_{n+1} + bS'_n + E'_n$ and $S''_{n+2} = a.S''_{n+1} + bS''_n + E''_n$. Absolute rounding errors (E''_n) at each filter iteration can be bounded by using Antoine Miné's framework [16]. Thus, we are left to design an abstract domain to deal with simplified filters. In our case, the simplified recursion is of the form $S_{n+2} = aS_{n+1} + bS_n + F_n$. We can use elliptic invariants of the form $S^2_{n+2} - aS_{n+1}S_n - bS^2_n \leq k^2$ to discover a bound on the output stream S_n. Then, we can compute an approximated reduced product with the already existing abstract domains. Reduction steps are performed when necessary in order not to lose accuracy. Intervals and equality constraints can be collected on the fly to capture the initial conditions of each filter.

In the general case, we can bound the output of simplified linear filters (i.e. filters that implement recursions of the form $S_{n+p} = a_1 S_n + \ldots + a_p S_{n+p-1} + F_n$) by splitting the recursion as a sum of second order recursions and of first order recursions. We require a factorization of the linear recursion characteristic polynomial into irreducible (in the real field) polynomials (we only need a sound approximation of the coefficient of each polynomial). We also need that the characteristic polynomial has only single roots. Then, we can bound second order summands by using elliptic constraints [2], while first order summands can be bounded by using interval constraints [4]. These bounds can be discovered iteratively by using simple transfer functions. We have given in [9] necessary conditions (taking into account rounding errors) for proving the convergence of first and second order filters, and we have given explicit bounds that can be used to accelerate abstract iterations. Nevertheless only the soundness of the transfer function is required to prove the soundness of the analysis [9]: the analysis is sound even if the acceleration is not sound. In the case when these necessary conditions are not satisfied, we can use the arithmetic-geometric progression domain [10] in order to compute bounds that depend of the program life time.

The framework was fully implemented in OCaml [13] and plugged into the astrée [2,7] analyzer. We have obtained bounds that are very close to sample experimental results, which has allowed solving nearly all of our remaining warnings.

Previous works. To our knowledge, our analysis [9] is the first analysis that abstracts filter output invariants. Nevertheless, some work has been done in filter optimization. In [12], affine equality relationships [11] among variables at the beginning and at the end of loop iterations are used to factorize filters at compile time. In our case, because of floating-point rounding errors, there are no such affine equality relationships, so a more complex domain such as polyhedra [8] is required to perform the same task. Moreover, our programs involve complex boolean control flows. Thus, filter factorization cannot be performed without a highly expensive partitioning. Furthermore, our goal is just to prove the absence of error at runtime, and not to describe precisely the global behavior of filters.

Outline. In Sect. 2, we present two case studies. In Sect. 3, we present the syntax and semantics of our language. In Sect. 4, we describe a generic abstraction for this language. In Sect. 5, we define a generic extension for refining existing abstractions. In Sect. 6, we give numerical abstract domains for describing sets of real numbers. In Sect. 7, we describe abstract domains to deal with first order and second order simplified filters. In Sect. 8, we show how to refine an abstract domain for a given class of simplified filters of a given order so that it can deal precisely with more complex filters of the same order. In these complex filters, more than one output is involved at each iteration.

$V \in \mathbb{R}$; $E_1 := 0$;$S := 0$;

while $(V \geq 0)$ {

$V \in \mathbb{R}$; $T \in \mathbb{R}$;

$E_0 \in [-1;1]$;

if $(T \geq 0)$ $\{S := 0\}$

else $\{S := 0.999 \times S + E_0 - E_1\}$

$E_1 := E_0$;

}

Fig. 1. A high bandpass filter.

In Sect. 9, we show how to build abstract domains for any class of simplified filters. In Sect. 10, we describe the impact of these extensions on the analysis results. In Sect. 11, we conclude and discuss some of the basic ideas of our framework.

2 Case studies

In this section, we present two examples of digital filter implementations. With a view to simplifying, we consider that these programs are computed in real arithmetics, although the case of floating-points arithmetics will be considered later.

2.1 The high bandpass filter

A *high bandpass* filter can be encoded by the program [2] given in Fig. 1. Roughly speaking, 0.999 is a coefficient of the filter. Variables V and T allow control flow enforcement. At each loop iteration, the variable S denotes the value of the current filter output, the variable E_0 denotes the value of the current filter input, and the variable E_1 denotes the value of the previous filter input. Depending on the value of the variable T, the filter is either reset (i.e., the output is set to 0), or iterated (i.e., the value of the next output is calculated from the last output value and the last two input values).

We now describe the behavior of a usual analyzer based on the use of interval

[2] The notation $V \in I$ where V is a variable and I is an interval means that a random value picked in the interval I is assigned to the variable V.

constraints. First, the analyzer infers the following sound counterpart $\mathbb{F}^\sharp \triangleq X \mapsto \textit{convex-hull}(\{0.999s + e_0 + e_1 \mid s \in X \cup \{0\},\ e_0, e_1 \in [-1; 1]\})$ to the loop body. Then, the analyzer starts iterating the abstract transfer function $\mathbb{F}^\sharp$ until it discovers a post-fixpoint. This gives the following iteration:

- $\mathbb{X}_0^\sharp = \emptyset$;
- $\mathbb{X}_1^\sharp = \mathbb{F}^\sharp(\mathbb{X}_0^\sharp) = [-2; 2]$;
- $\mathbb{X}_2^\sharp = \mathbb{F}^\sharp(\mathbb{X}_1^\sharp) = [-3.998; 3.998]$.

The iteration cannot reach a post-fixpoint this way. In order to extrapolate a post-fixpoint, we use a widening operator [5]. We propose the use of a widening with thresholds. This widening consists in replacing each unstable interval bound with the next bound in a finite list of thresholds. In our example, we consider that the thresholds are the numbers of the form 10^n.

Thus the analyzer tries each threshold until it discovers a post-fixpoint:

- $\mathbb{X}_3^\triangledown = [-10; 10]$, but $\mathbb{F}^\sharp([-10; 10]) \not\subseteq [-10; 10]$;
- $\mathbb{X}_4^\triangledown = [-100; 100]$, but $\mathbb{F}^\sharp([-100; 100]) \not\subseteq [-100; 100]$;
- $\mathbb{X}_5^\triangledown = [-1000; 1000]$, but $\mathbb{F}^\sharp([-1000; 1000]) \not\subseteq [-1000; 1000]$;
- $\mathbb{X}_6^\triangledown = [-10000; 10000]$ and $\mathbb{F}^\sharp([-10000; 10000]) \subseteq [-10000; 10000]$.

Finally, we keep on iterating to refine the solution:

- $\mathbb{X}_0^\triangle = [-10000; 10000]$,
- $\mathbb{X}_1^\triangle = [-9992; 9992]$,
- $\mathbb{X}_2^\triangle = [-9984.008; 9984.008]$,

Of course, better results could have been obtained by driving the analysis, as stated by Thm. 1:

Theorem 1 (High bandpass filter *(history-insensitive version)*) *Let $D \geq 0$, $m \geq 0$, a, X and Z be real numbers such that: $|X| \leq D$ and $aX - m \leq Z \leq aX + m$. We have:*

(1) $|Z| \leq |a|D + m$;

(2) if $|a| < 1$ *and* $D \geq \frac{m}{1-|a|}$, *then* $|Z| \leq D$.

PROOF. Let $D \geq 0$, $m \geq 0$, a, X and Z be real numbers such that $|X| \leq D$ and $aX - m \leq Z \leq aX + m$.

(1) Since $m \geq 0$, we have $|Z| \leq |a||X| + m$. Since $|a| \geq 0$ and $|X| \leq D$, we conclude that: $|Z| \leq |a|D + m$.
(2) We assume that both $|a| < 1$ and $D \geq \frac{m}{1-|a|}$. We have $1 - |a| > 0$, so

$m \leq (1-|a|)D$. Then $|Z| \leq |a|D+m \leq |a|D+(1-|a|)D$. So we conclude that $|Z| \leq D$. □

In Thm. 1, the variable a is a parameter of the filter. The variable X denotes the current output of the filter, which ranges between the value $-D$ and the value D. The current input ranges between the value $-m$ and the value m. Then, the next output Z is picked within the interval $[aX - m; aX + m]$. Thm. 1 states that: if the filter is contracting (i.e. $|a| < 1$), then the value $\frac{m}{1-|a|}$ is an inductive bound on the absolute value of the output stream. In our example, we have $\frac{m}{1-|a|} = 2000$. so we can get a more precise analysis when the number 2000 is chosen as a threshold.

Nevertheless, the number 2000 is not the best accurate bound. We have lost information when abstracting away the cancellation effect in the sub-term $E_0 - E_1$. To take into account this cancellation effect, we propose to express the value of the current output as a linear combination of all previous inputs and of the value of the output at filter initialization. We obtain the following theorem:

Theorem 2 (High bandpass filter *(history-sensitive version)*) *Let $a \in [\frac{1}{2}; 1[$, i, and $m \geq 0$ be real numbers. Let E_n be a sequence of real numbers, such that $\forall k \in \mathbb{N},\ E_k \in [-m; m]$. Let S_n be the sequence of real numbers, defined by: $S_0 = i$ and $S_{n+1} = aS_n + E_{n+1} - E_n$. We have: $|S_n| \leq 2m + |i|$.*

PROOF. Let $a \in [\frac{1}{2}; 1[$, i, and $m \geq 0$ be real numbers. Let E_n be a sequence of real numbers, such that $\forall k \in \mathbb{N},\ E_k \in [-m; m]$. Let S_n be the sequence of real numbers, defined by: $S_0 = i$ and $S_{n+1} = aS_n + E_{n+1} - E_n$. We first prove, by induction over $n \in \mathbb{N}$, that, for any natural number $n \in \mathbb{N}$, we have $S_n = a^n i + E_n - a^n E_0 + \sum_{l=1}^{n-1}(a-1)a^{l-1}E_{n-l}$:

- We have $S_0 = i = a^0 i + E_0 - a^0 E_0$.
- We suppose that there exists $n_0 \in \mathbb{N}$ such that $S_{n_0} = a^{n_0} i + E_{n_0} - a^{n_0} E_0 + \sum_{l=1}^{n_0-1}(a-1)a^{l-1}E_{n_0-l}$. We have $S_{n_0+1} = aS_{n_0} + E_{n_0+1} - E_{n_0}$, so $S_{n_0+1} = E_{n_0+1} - E_{n_0} + a\left(a^{n_0} i + E_{n_0} - a^{n_0} E_0 + \sum_{l=1}^{n_0-1}(a-1)a^{l-1}E_{n_0-l}\right)$. So $S_{n_0+1} = a^{n_0+1} i + E_{n_0+1} - a^{n_0+1} E_0 + \left(\sum_{l'=1}^{n_0-1}(a-1)a^{(l'+1)-1}E_{n_0+1-(l'+1)}\right) + (a-1)E_{n_0}$. We replace $l'+1$ with l to conclude that $S_{n_0+1} = a^{n_0+1} i + E_{n_0+1} - a^{n_0+1} E_0 + \sum_{l=1}^{n_0+1-1}(a-1)a^{l-1}E_{n_0+1-l}$.

So, for any natural number $n \in \mathbb{N}$, we have $S_n = a^n i + E_n - a^n E_0 + \sum_{l=1}^{n-1}(a-1)a^{l-1}E_{n-l}$. Then, $|S_n| \leq |a|^n|i| + |E_n| + |a|^n|E_0| + \sum_{l=1}^{n-1}|(a-1)a^{l-1}||E_{n-l}|$. Since $0 < a < 1$ and $|E_i| < m$, we have $|S_n| \leq a^n|i| + m + a^n m + (1-a)m\sum_{l=1}^{n-1} a^{l-1}$. So, $|S_n| \leq a^n|i| + m(1 + a^n + 1 - a^n)$. We conclude that $|S_n| \leq a^n|i| + 2m$. Since $a < 1$, we conclude that $|S_n| \leq 2m + |i|$. □

$V \in \mathbb{R}$;

$E_1 := 0$; $E_2 := 0$; $S_0 := 0$; $S_1 := 0$; $S_2 := 0$;

while $(V \geq 0)$ {

$V \in \mathbb{R}$; $T \in \mathbb{R}$;

$E_0 \in [-1; 1]$;

if $(T \geq 0)$ $\{S_0 := 0; S_1 := 0; E_1 := 0\}$

else $\{S_0 := 1.5 \times S_1 - 0.7 \times S_2 + 0.5 \times E_0 - 0.7 \times E_1 + 0.4 \times E_2\}$;

$E_2 := E_1$; $E_1 := E_0$;

$S_2 := S_1$; $S_1 := S_0$

}

Fig. 2. A second order filter.

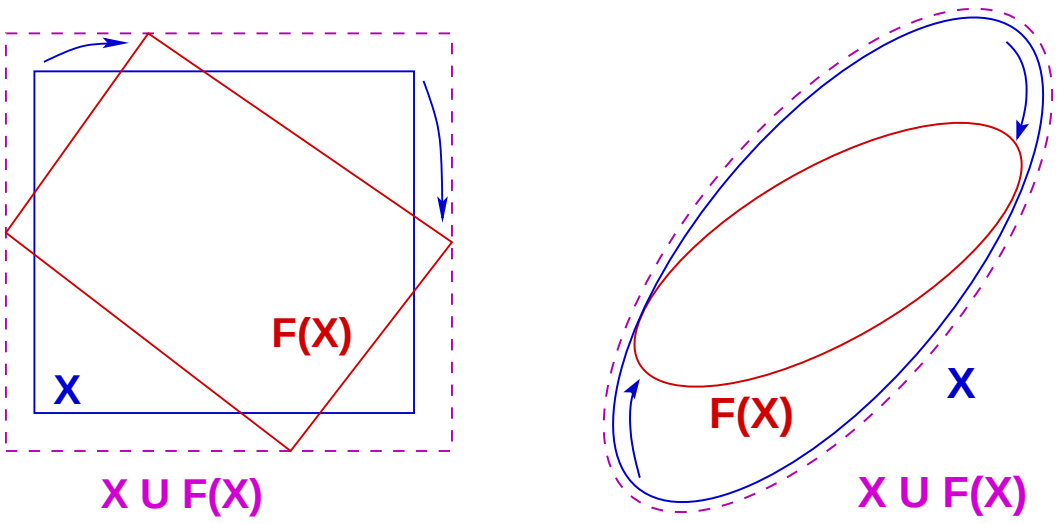
In Thm. 2, a is a parameter of the filter. The variable i denotes the value of the output at the initialization of the filter. The sequence (E_n) denotes the input stream and the sequence (S_n) denotes the output stream. First, we expand the current output as a linear combination of the initial output and of all previous inputs. Then we can simplify this expression to get a bound which only depends on the initial output and on the bound on the input stream. By using Thm. 2, we can build an accurate domain that discovers that 2 is a correct bound on the value of $|S|$.

So, we have seen that we can analyze a program which contains a high band-pass filter quite easily. Even an interval-based analysis will bound the output. The bound can be refined by driving the analysis. Nevertheless, we need to formally expand the output as a linear combination of all previous inputs in order to get a better accuracy.

2.2 The second order filter

The case of the second order filter is much more interesting, because the output of such a filter cannot be bound without relational information. Moreover, existing relational domains are not likely to provide inductive invariants.

The program given in Fig. 2 implements a second order filter. Roughly speaking, the numbers 1.5, -0.7, 0.5, -0.7, and 0.4 are the coefficients of the filter. Variables V and T allow control flow enforcement. At each loop iteration, the variable S_0 denotes the value of the current filter output, variables S_1 and S_2 denote the last two values of the filter output, the variable E_0 denotes the



(a) Linear constraints. (b) Elliptic constraints.

Fig. 3. Transformation induced by the body of the loop.

value of the current filter input, and variables E_1 and E_2 denote the last two values of the filter input. Depending on the value of the variable T, either the filter is reset (i.e., both the current and the previous outputs are set to the same value), or iterated (i.e., the value of the next output is calculated from the last two output values and the last three input values).

When analyzing such a program, we usually look for a post-fixpoint of the following sound counterpart $\mathbb{F}^\sharp \triangleq (X, Y) \mapsto (\mathit{convex\text{-}hull}(\{1.5x - 0.7y + 0.5e_0 + 0.7e_1 + 0.5e_2 \mid x \in X, y \in Y,\ e_0, e_1, e_2 \in [-1; 1]\}), X)$ to the loop body. Unfortunately, the only post-fixpoint is $\mathbb{R}^2$ itself. So we fail in discovering a bound on the filter output. More generally, whenever we apply the body of the loop to a set of points, first the set is enlarged according to one direction, then it is contracted according to another direction, and finally the solution rotates around the origin. So an abstract domain should take care of both the rotation and the enlargement in order to find an inductive invariant. Linear constraints are very unlikely to provide invariants because of the corners (Cf. Fig. 3(a)). Following Thm. 3, elliptic constraints are much convenient to discover inductive invariants[3] (Cf. Fig. 3(b)).

Theorem 3 (second order filter *(history insensitive version)*)
Let a, b, $k \geq 0$, $m \geq 0$, X, Y, Z be real numbers that satisfy: $a^2 + 4b < 0$, $X^2 - aXY - bY^2 \leq k^2$, and $aX + bY - m \leq Z \leq aX + bY + m$. We have:

(1) $Z^2 - aZX - bX^2 \leq \left(\sqrt{-b}k + m\right)^2$;

[3] Of course, the polygons that are close enough from a stable ellipse are also stables. But such polygons would be as difficult to discover as the ellipse itself.

(2) if $\sqrt{-b} < 1$ *and* $k \geq \frac{m}{1-\sqrt{-b}}$, *then* $Z^2 - aZX - bX^2 \leq k^2$.

PROOF. Let a, b, $k \geq 0$, $m \geq 0$, X, Y, Z be real numbers that satisfy: $a^2 + 4b < 0$, $X^2 - aXY - bY^2 \leq k^2$, and $aX + bY - m \leq Z \leq aX + bY + m$.

(1) Let $t = Z - aX - bY$. We have $Z^2 - aZX - bX^2 = (t + aX + bY)^2 - a(t + aX + bY)X - bX^2$. So $Z^2 - aZX - bX^2 = -b(X^2 + (a - 2a)XY - bY^2) + 2btY + (t+aX)^2 - a(t+aX)X$. Then $Z^2 - aZX - bX^2 = -b(X^2 - aXY - bY^2) + t(2bY + t + 2aX - aX)$. We obtain that $Z^2 - aZX - bX^2 = -b(X^2 - aXY - bY^2) + t(t + 2bY + aX)$. Moreover, we have: $(aX + 2bY)^2 = -4b(\frac{a^2}{-4b}X^2 - aXY - bY^2)$. Since $4b < 0$ and $a^2 + 4b < 0$, we obtain that $(aX + 2bY)^2 \leq -4b(X^2 - aXY - bY^2)$. Then, since $4b < 0$ and $X^2 - aXY - bY^2 \leq k^2$, we deduce that $(aX + 2bY)^2 \leq -4bk^2$. Since $-4b > 0$, we conclude that $|aX + 2bY| \leq 2k\sqrt{-b}$. Then, we have $t(t + 2bY + aX) \leq |t|(|t| + |aX + 2bY|)$. Since $|t| \leq m$ and $|aX + 2bY| \leq 2k\sqrt{-b}$, we conclude that $t(t + 2bY + aX) \leq m(m + 2k\sqrt{-b})$. Moreover, we know that $X^2 - aXY - bY^2 \leq k^2$ and $b < 0$. Since $Z^2 - aZX - bX^2 = -b(X^2 - aXY - bY^2) + t(t + 2bY + aX)$, we have $Z^2 - aZX - bX^2 \leq -bk^2 + m(m + 2k\sqrt{-b})$. We conclude that $Z^2 - aZX - bX^2 \leq (k\sqrt{-b} + m)^2$.

(2) We assume that both $\sqrt{-b} < 1$ and $k \geq \frac{m}{1-\sqrt{-b}}$. We have $1 - \sqrt{-b} > 0$, so $m \leq (1 - \sqrt{-b})k$. Then, since $0 \leq \sqrt{-b}k + m$, we have: $Z^2 - aZX - bX^2 \leq \left(\sqrt{-b}k + (1 - \sqrt{-b})k\right)^2$. So we can conclude that $Z^2 - aZX - bX^2 \leq k^2$. □

In Thm. 3, the variables a and b are two parameters of the filter. The variable X denotes the current output of the filter. The variable Y denotes the previous output. The current input ranges between the value -m and the value m. Then, the next output Z is picked within the interval $[aX + bY - m; aX + bY + m]$. Thm. 3 suggests to study the value of the expression $\sqrt{|V^2 - aVU - bU^2|}$ where U and V denote two successive outputs. At each iteration of the filter, an affine transformation is applied to this value. When $\sqrt{-b} < 1$, the filter output stays within a stable ellipse: in such a case an interval analysis of the value of the expression $\sqrt{|V^2 - aVU - bU^2|}$ will succeed in bounding the filter output. Otherwise, the filter diverges: in such a case we may use the arithmetic and geometric progression domain [10], to relate the ellipse ratio to the execution time of the program. In our example, the filter is convergent: a driven interval analysis discovers that the number 23 is a correct bound on the values of output stream.

Once again, the number 23 is not the most accurate bound. We have lost information when abstracting away the cancellation effect in the sub-term $0.5 \times E_0 - 0.7 \times E_1 + 0.4 \times E_2$. To take into account this cancellation effect, we

express the value of the current output as a linear combination of all previous inputs and of the value of the output at filter initialization. The expression that we obtain cannot be simplified as we did in the case of the high bandpass filter (Cf. Sect. 2.1). Nevertheless, we can choose a parameter N and express each filter output as a linear combination of the last N inputs and of a residue. We will see in Sect. 8 that the residue may be bound by using the history insensitive approximation. This approach is more generic than the one we use in the case of the high bandpass filter and can be applied with any linear filters. With this approach, our analysis detects and proves that the absolute value of any output of our filter is bounded by 1.42.

3 Language

In this section we introduce both the syntax and the semantics of the language that we want to analyze.

We analyze a subset of C without dynamic memory allocation nor side-effect. Moreover, the use of pointer operations is restricted to call-by reference. For the sake of simplicity, we introduce an intermediate language to describe programs that are interpreted between the concrete and an abstract level. Data structures have been translated by using a finite set of abstract cells (see [2, Sect. 6.1]). Non-deterministic branching over-approximates all the memory accesses (array accesses, pointer dereferencing) that are not fully statically resolved. Furthermore, floating-point expressions have been conservatively approximated by linear forms with real interval coefficients. These linear forms include both the rounding errors and some expression approximations (see [16]). We also suppose that the occurrence of runtime errors (such as floating-point overflows) can be described by interval constraints on the memory state.

3.1 Syntax

Let $\mathcal{V}$ be a finite set of variables. Let $\texttt{clock} \notin \mathcal{V}$ be an extra variable which is associated with a clock counter. The clock counter is explicitly incremented when a command **tick** is executed. The system stops when the clock counter overflows a maximum value which is defined by the end-user. We denote by $\mathcal{I}$ the set of all real number intervals (including $\mathbb{R}$ itself). We define inductively the syntax of programs in Fig. 4. We denote by $\mathcal{E}$ the set of expressions E.

$V \in \mathcal{V}$, $I \in \mathcal{I}$

$E := I \mid V \mid I \times V + E$

$P := V = E \mid \textbf{skip} \mid \textbf{tick} \mid \textbf{if } (V \geq 0)\ \{P\} \textbf{ else } \{P\} \mid \textbf{while } (V \geq 0)\ \{P\} \mid P; P$

Fig. 4. Syntax.

$$(\!|I|\!)(\rho) = I,\ (\!|V|\!)(\rho) = \{\rho(V)\}$$

$$(\!|I \times V + E|\!)(\rho) = \{b \times \rho(V) + a \mid a \in (\!|E|\!)(\rho), b \in I\}$$

$$[\![V = E]\!]_{\mathtt{mc}}(\rho) = \{\rho[V \mapsto x] \mid x \in (\!|E|\!)(\rho)\}$$

$$[\![\textbf{skip}]\!]_{\mathtt{mc}}(\rho) = \{\rho\}$$

$$[\![\textbf{tick}]\!]_{\mathtt{mc}}(\rho) = \begin{cases} \{\rho[\mathtt{clock} \mapsto \rho(\mathtt{clock}) + 1]\} & \text{if } \rho(\mathtt{clock}) < \mathtt{mc} \\ \emptyset & \text{otherwise} \end{cases}$$

$$[\![\textbf{if } (V \geq 0)\ \{P_1\} \textbf{ else } \{P_2\}]\!]_{\mathtt{mc}}(\rho) = \begin{cases} [\![P_1]\!]_{\mathtt{mc}}(\rho) & \text{if } \rho(V) \geq 0 \\ [\![P_2]\!]_{\mathtt{mc}}(\rho) & \text{otherwise} \end{cases}$$

$$[\![\textbf{while } (V \geq 0)\ \{P\}]\!]_{\mathtt{mc}}(\rho) = \{\rho' \in \mathit{Inv} \mid \rho'(V) < 0\}$$

$$\text{where } \mathit{Inv} = \mathrm{lfp}\,(X \mapsto \{\rho\} \cup (\bigcup\{[\![P]\!]_{\mathtt{mc}}(\rho') \mid \rho' \in X, \rho'(V) \geq 0\}))$$

$$[\![P_1; P_2]\!]_{\mathtt{mc}}(\rho) = \bigcup\{[\![P_2]\!]_{\mathtt{mc}}(\rho') \mid \rho' \in [\![P_1]\!]_{\mathtt{mc}}(\rho)\}$$

Fig. 5. Concrete semantics.

3.2 Semantics

We describe the semantics of these programs in a denotational way. An *environment* ($\rho \in \mathcal{V} \cup \{\mathtt{clock}\} \to \mathbb{R}$) denotes a memory state. It maps each variable, including the clock variable, to a real number. We denote by *Env* the set of all environments. The semantics of an expression E is a function $(\!|E|\!) \in \mathit{Env} \to I$ mapping each environment to an interval. Given a maximum value `mc` for the clock, the semantics of a program P is a function $[\![P]\!]_{\mathtt{mc}} \in \mathit{Env} \to \wp(\mathit{Env})$ mapping each environment ρ to the set of the environments that can be reached when applying the program P starting from the environment ρ. Returning a set of environments allows the description of both non-determinism and program halting (when the clock has reached its maximum value). The functions $(\!|\_|\!)$ and $[\![\_]\!]_{\mathtt{mc}}$ are defined by induction on the syntax of programs in Fig. 5. Loop semantics requires the computation of a *loop invariant*, which is the set of all environments that can be reached just before the guard of this loop is tested. This invariant is well-defined as the least fixpoint of a $\cup$-complete endomorphism[4] $f \in \wp(\mathit{Env}) \to \wp(\mathit{Env})$. Nevertheless, such a fixpoint is usually not computable, so we give a decidable approximate semantics in the next section.

[4] In fact, we only use the monotonicity of f.

We describe our two filter examples in our language. Unlike in Sect. 2, we also describe rounding errors.

Example 4 *A* high bandpass *filter can be encoded by the following program:*

$$
\begin{array}{l}
V = \mathbb{R};\ E_1 = [0;0];\ S = [0;0];\\
\textbf{while}\ (V \geq 0)\ \{\\
\quad V = \mathbb{R};\ T = \mathbb{R};\ E_0 = I;\\
\quad \textbf{if}\ (T \geq 0)\ \{S = [0;0]\}\\
\quad \textbf{else}\ \{S = A \times S + E_0 + (-E_1) + F\};\\
\quad E_1 = E_0;\\
\}
\end{array}
$$

Roughly speaking, the interval I denotes the range of filter entries. Floating-point rounding errors are captured by the range of both intervals A and F. The interval A describes the filter coefficient and satisfies $A \subseteq [\frac{1}{2}; 1[$. Variables V and T allow control flow enforcement. At each loop iteration, the variable S denotes the value of the current filter output, the variable E_0 denotes the value of the current filter input, and the variable E_1 denotes the value of the previous filter input. Depending on the value of T, the filter is either reset (i.e., the output is set to 0), or iterated (i.e., the value of the next output is calculated from the last output value and the two last input values). The analysis described in [2] only discovers inaccurate bounds for the variable S. It works as if the expression $A \times S + E_0 - E_1 + F$ were approximated by $A \times S + (2 \times I + F)$. The analysis discovers the first widening threshold l (see [1, Sect. 2.1.2]) such that l is greater than $\frac{2 \times i + f}{1-a}$, for any $(i, f, a) \in I \times F \times A$. It proves that l is stable, and then successive narrowing iterations refine the value l. Thus, the cancellation effects of $E_0 - E_1$ are ignored. □

Example 5 *A* second order *digital filter can be encoded as follows:*

$V = \mathbb{R}$; $E_1 = [0;0]$; $E_2 = [0;0]$; $S_0 = [0;0]$; $S_1 = [0;0]$; $S_2 = [0;0]$;

while $(V \geq 0)$ {

$V = \mathbb{R}$; $T = \mathbb{R}$; $E_0 = I$;

if $(T \geq 0)$ $\{S_0 = E_0;\ S_1 = E_0\}$

else $\{S_0 = A \times S_1 + B \times S_2 + C \times E_0 + D \times E_1 + E \times E_2 + F\}$;

$E_2 = E_1$; $E_1 = E_0$; $S_2 = S_1$; $S_1 = S_0$

}

Roughly speaking, the interval I denotes the range of filter entries. Intervals A, B, C, D and E denote filter coefficients and satisfy $A \subseteq [0;\infty[$, $B \subseteq]-1;0[$ and $\forall (a,b) \in A \times B,\ a^2 + 4 \times b < 0$. Floating-point rounding errors are captured by the range of intervals A, B, C, D, E and F. Variables V and T allow control flow enforcement. At each loop iteration, the variable S_0 denotes the value of current filter output, variables S_1 and S_2 denote the last two values of filter output, the variable E_0 denotes the value of the current filter input, and variables E_1 and E_2 denote the last two values of filter input. Depending on the value of T, either the filter is reset (i.e., both the current and the previous outputs are set to the same value), or iterated (i.e., the value of the next output is calculated from the last two output values and the last three input values). The analysis described in [2] fails to discover any bound for the variables S_0, S_1, S_2. □

4 Underlying domain

We use the Abstract Interpretation framework [3, 5, 6] to derive a generic approximate semantics. An abstract domain $Env^\sharp$ is a set of properties about memory states. Each abstract property is related to the set of the environments which satisfy it via a concretization map γ. An operator $\sqcup$ allows the gathering of information about different control flow paths. To effectively compute an approximation of concrete fixpoints, we introduce an iteration basis $\bot$, a widening operator $\triangledown$, and a narrowing operator $\triangle$. The primitives ASSIGN, GUARD, and TICK are sound counterparts to concrete assignments, guards, and clock ticks. Several abstract domains collaborate and use simple constraints to refine each other. We introduce two domains of simple constraints. The domain of intervals $\mathcal{V} \cup \{\texttt{clock}\} \to \mathcal{I}$ and the domain of equality relationship $\wp(\mathcal{V}^2)$. The interval constraints encoded by a map $\rho^\sharp \in \mathcal{V} \cup \{\texttt{clock}\} \to \mathcal{I}$ are satisfied by the environment set $\gamma_{\mathcal{I}}(\rho^\sharp) = \{\rho \in Env \mid \rho(X) \in \rho^\sharp(X),\ \forall X \in \mathcal{V} \cup \{\texttt{clock}\}\}$. The constraints encoded by a subset $\mathcal{R} \subseteq \mathcal{V}^2$ are satisfied by the environment set $\gamma_=(\mathcal{R}) = \{\rho \in Env \mid \forall (X,Y) \in \mathcal{R},\ \rho(X) = \rho(Y)\}$. The primitives RANGE

and EQU capture simple constraints about the values that are associated with variables by weakening the abstract elements of $Env^\sharp$. These constraints are useful in refining the filter domains. Conversely, a primitive RED uses the constraints that have been computed by the filter domains in order to refine the underlying domain.

Def. 6 formalizes the definition of a generic abstraction.

Definition 6 (Generic abstraction) *An abstraction is defined by a tuple* $(Env^\sharp, \gamma, \sqcup, \bot, \nabla, \triangle, \text{ASSIGN}, \text{GUARD}, \text{TICK}, \text{RANGE}, \text{RED}, \text{EQU})$ *such that:*

(1) structure:
- $Env^\sharp$ *is a set of properties;*
- $\gamma \in Env^\sharp \to \wp(Env)$ *is a concretization map;*
- $\forall a, b \in Env^\sharp, \gamma(a) \cup \gamma(b) \subseteq \gamma(a \sqcup b)$*;*

(2) extrapolation primitives:
- $\bot$ *is an abstract element in* $Env^\sharp$*;*
- ∇ *is a widening operator such that:* $\forall a, b \in Env^\sharp,\ \gamma(a) \cup \gamma(b) \subseteq \gamma(a \nabla b)$*; and* $\forall k \in \mathbb{N},\ \rho_1^\sharp, \ldots, \rho_k^\sharp \in (\mathcal{V} \cup \{\texttt{clock}\} \to \mathcal{I}),\ (a_i) \in (Env^\sharp)^\mathbb{N}$*, the sequence* (a_i^∇) *defined by* $a_0^\nabla = r(a_0)$ *and* $a_{n+1}^\nabla = r(a_n^\nabla \nabla a_{n+1})$ *with* $r = [X \mapsto \text{RED}(\rho_k^\sharp, X)] \circ \ldots \circ [X \mapsto \text{RED}(\rho_1^\sharp, X)]$*, is ultimately stationary;*
- $\triangle$ *is a narrowing operator such that:* $\forall a, b \in Env^\sharp,\ \gamma(a) \cap \gamma(b) \subseteq \gamma(a \triangle b)$*; and* $\forall k \in \mathbb{N},\ \rho_1^\sharp, \ldots, \rho_k^\sharp \in (\mathcal{V} \cup \{\texttt{clock}\} \to \mathcal{I}),\ (a_i) \in (Env^\sharp)^\mathbb{N}$*, the sequence* $(a_i^\triangle)$ *defined by* $a_0^\triangle = r(a_0)$ *and* $a_{n+1}^\triangle = r(a_n^\triangle \triangle a_{n+1})$*, with* $r = [X \mapsto \text{RED}(\rho_k^\sharp, X)] \circ \ldots \circ [X \mapsto \text{RED}(\rho_1^\sharp, X)]$*, is ultimately stationary;*

(3) transfer functions:
- $\forall a \in Env^\sharp, X \in \mathcal{V}, E \in \mathcal{E}, \rho \in \gamma(a)$,
  $[\![X = E]\!]_{\texttt{mc}}(\rho) \subseteq \gamma(\text{ASSIGN}(X = E, a))$*;*
- $\forall a \in Env^\sharp, X \in \mathcal{V} \cup \{\texttt{clock}\}, I \in \mathcal{I}$,
  $\{\rho \in \gamma(a) \mid \rho(X) \in I\} \subseteq \gamma(\text{GUARD}(X, I, a))$*;*
- $\forall a \in Env^\sharp, \{\rho[\texttt{clock} \mapsto \rho(\texttt{clock}) + 1] \mid \rho \in \gamma(a)\} \subseteq \gamma(\text{TICK}(a))$*;*

(4) conversion primitives:
- $\forall a \in Env^\sharp, \rho^\sharp \in (\mathcal{V} \cup \{\texttt{clock}\} \to \mathcal{I}),\ \gamma(a) \cap \gamma_\mathcal{I}(\rho^\sharp) \subseteq \gamma(\text{RED}(\rho^\sharp, a))$*;*
- $\forall a \in Env^\sharp,\ \gamma(a) \subseteq \gamma_\mathcal{I}(\text{RANGE}(a))$ *and* $\gamma(a) \subseteq \gamma_=(\text{EQU}(a))$*.*

Least fixpoint approximation is performed in two steps [5]: we first compute an approximation using the widening operator; then we refine it using the narrowing operator. More formally, let f be a monotonic map in $\wp(Env) \to \wp(Env)$ and $f^\sharp \in Env^\sharp \to Env^\sharp$ be an abstract counterpart of f satisfying $\forall a \in Env^\sharp,\ (f \circ \gamma)(a) \subseteq (\gamma \circ f^\sharp)(a)$. It is worth noting that the abstract counterpart $f^\sharp$ is usually not monotonic with respect to the partial order $\sqsubseteq^\sharp$ that is defined by $a \sqsubseteq^\sharp b \iff \gamma(a) \subseteq \gamma(b)$. The abstract upward iteration (C_n^∇) of $f^\sharp$ is defined by $C_0^\nabla = \bot$ and $C_{n+1}^\nabla = C_n^\nabla \nabla f^\sharp(C_n^\nabla)$. The sequence (C_n^∇) is ultimately stationary and we denote its limit by C_ω^∇. Then the abstract downward iteration $(D_n^\triangle)$ of $f^\sharp$ is defined by $D_0^\triangle = C_\omega^\nabla$ and $D_{n+1}^\triangle = D_n^\triangle \triangle f^\sharp(D_n^\triangle)$.

The sequence $(D_n^\triangle)$ is ultimately stationary and we denote its limit by $D_\omega^\triangle$. We define[5] $\mathrm{lfp}^\sharp(f^\sharp)$ by the limit $D_\omega^\triangle$ of the abstract downward iteration of $f^\sharp$. We introduce some lemmas in order to prove that $\mathrm{lfp}(f) \subseteq \gamma(D_\omega^\triangle)$:

Lemma 7 *We have* $f(\gamma(C_\omega^\triangledown)) \subseteq \gamma(C_\omega^\triangledown)$.

PROOF. $C_\omega^\triangledown$ is the limit of the upward-iteration, so $C_\omega^\triangledown = C_\omega^\triangledown \triangledown f^\sharp(C_\omega^\triangledown)$. By Def. 6.(2) of the widening, we obtain that $\gamma(f^\sharp(C_\omega^\triangledown)) \subseteq \gamma(C_\omega^\triangledown)$. By soundness of $f^\sharp$, we also have $f(\gamma(C_\omega^\triangledown)) \subseteq \gamma(f^\sharp(C_\omega^\triangledown))$. So $f(\gamma(C_\omega^\triangledown)) \subseteq \gamma(C_\omega^\triangledown)$. □

Lemma 8 *For any* $a \in \wp(Env)$ *and* $x \in Env^\sharp$, $a \subseteq \gamma(x) \implies a \cap f(a) \subseteq \gamma(x \triangle f^\sharp(x))$.

PROOF. Let $a \in \wp(Env)$ and $x \in Env^\sharp$ such that $a \subseteq \gamma(x)$. Since f is monotonic, we have $f(a) \subseteq f(\gamma(x))$. Then by soundness of $f^\sharp$, we have $f(\gamma(x)) \subseteq \gamma(f^\sharp(x))$. Thus $f(a) \subseteq \gamma(f^\sharp(x))$. So $a \cap f(a) \subseteq \gamma(x) \cap \gamma(f^\sharp(x))$. By Def. 6.(2), we have $\gamma(x) \cap \gamma(f^\sharp(x)) \subseteq \gamma(x \triangle f^\sharp(x))$. We conclude that $a \cap f(a) \subseteq \gamma(x \triangle f^\sharp(x))$. □

Lemma 9 *For any* $a \in \wp(Env)$, $f(a) \subseteq a \implies f(f(a) \cap a) \subseteq f(a) \cap a$.

PROOF. Let $a \in \wp(Env)$ such that $f(a) \subseteq a$. Since f is monotonic, we have $f(f(a)) \subseteq f(a)$. Moreover, we have $f(a) \cap a = f(a)$. We conclude that $f(f(a) \cap a) = f(f(a)) \subseteq f(a) = f(a) \cap a$. □

Lemma 10 (transfinite kleenean iteration)
For any $a \in \wp(Env)$, *we have:* $f(a) \subseteq a \implies lfp(f) \subseteq a$.

Theorem 11 *We have* $lfp(f) \subseteq \gamma(D_\omega^\triangle)$.

PROOF. We introduce the sequence (u_n) that is defined by $u_0 = \gamma(C_\omega^\triangledown)$ and $u_{n+1} = u_n \cap f(u_n)$ for any $n \in \mathbb{N}$. We can prove by induction that $\forall n \in \mathbb{N}$, we have both $u_n \subseteq \gamma(D_n^\triangle)$ and $f(u_n) \subseteq u_n$:

- When $n = 0$: by definition, we have $u_0 = \gamma(C_\omega^\triangledown) = \gamma(D_0^\triangle)$ and thanks to Lemma 7, we have $f(u_0) \subseteq u_0$.
- We now suppose there exists $n \in \mathbb{N}$ such that $u_n \subseteq D_n^\triangle$ and $f(u_n) \subseteq u_n$.

(1) We have $u_{n+1} = u_n \cap f(u_n)$ and $u_n \subseteq \gamma(D_n^\triangle)$. By Lemma 8, we have $u_{n+1} \subseteq \gamma(D_n^\triangle \triangle f^\sharp(D_n^\triangle))$. By definition of $D_{n+1}^\triangle$, we obtain that $u_{n+1} \subseteq \gamma(D_{n+1}^\triangle)$.

[5] $\mathrm{lfp}^\sharp(f^\sharp)$ is an approximation of the concrete least fixpoint; it may not be a least fixpoint of the abstract counterpart $f^\sharp$ which is not supposed to be monotonic.

$$[\![V = E]\!]^\sharp_{\mathtt{mc}}(a) = \textsc{assign}(V = E, a)$$

$$[\![\mathbf{skip}]\!]^\sharp_{\mathtt{mc}}(a) = a$$

$$[\![\mathbf{tick}]\!]^\sharp_{\mathtt{mc}}(a) = \textsc{guard}(\mathtt{clock}, [0; \mathtt{mc}], \textsc{tick}(a))$$

$$[\![\mathbf{if}\ (V \geq 0)\ \{P_1\}\ \mathbf{else}\ \{P_2\}]\!]^\sharp_{\mathtt{mc}}(a) = a_1 \sqcup a_2,$$
$$\text{where } \begin{cases} a_1 = [\![P_1]\!]^\sharp_{\mathtt{mc}}(\textsc{guard}(V, [0; +\infty[, a)) \\ a_2 = [\![P_2]\!]^\sharp_{\mathtt{mc}}(\textsc{guard}(V,]-\infty; 0[, a)) \end{cases}$$

$$[\![\mathbf{while}\ (V \geq 0)\ \{P\}]\!]^\sharp_{\mathtt{mc}}(a) = \textsc{guard}(V,]-\infty; 0[, Inv^\sharp),$$
$$\text{where } Inv^\sharp = \mathrm{lfp}^\sharp \left(X \mapsto a \sqcup [\![P]\!]^\sharp_{\mathtt{mc}}(\textsc{guard}(V, [0; +\infty[, X))\right)$$

$$[\![P_1; P_2]\!]^\sharp_{\mathtt{mc}}(a) = [\![P_2]\!]^\sharp_{\mathtt{mc}}([\![P_1]\!]^\sharp_{\mathtt{mc}}(a))$$

Fig. 6. Abstract semantics.

(2) We have $f(u_{n+1}) = f(u_n \cap f(u_n))$ and $f(u_n) \subseteq u_n$. By Lemma 9, we obtain that $f(u_{n+1}) \subseteq u_{n+1}$.

Then let $n \in \mathbb{N}$ be a natural such that $D^\triangle_\omega = D^\triangle_n$. We have $u_n \subseteq \gamma(D^\triangle_\omega)$ and $f(u_n) \subseteq u_n$. By lemma 10, we have $\mathrm{lfp}(f) \subseteq \gamma(D^\triangle_\omega)$. □

The abstract semantics of a program is given by a function ($[\![\_]\!]^\sharp_{\mathtt{mc}} \in Env^\sharp \to Env^\sharp$) in Fig. 6. Its soundness can be proved by induction on the syntax:

Theorem 12 *For any program P, environment ρ, abstract element a, and maximum clock value* mc*, we have: $\rho \in \gamma(a) \implies [\![P]\!]_{\mathtt{mc}}(\rho) \subseteq \gamma\left([\![P]\!]^\sharp_{\mathtt{mc}}(a)\right)$.*

PROOF. The proof is made by induction over the syntax of P:

- In the case when P matches $V = E$: For any environment $\rho \in \gamma(a)$, we have $[\![P]\!]_{\mathtt{mc}}(\rho) \subseteq \gamma(\textsc{assign}(V = E, a))$ (Def. 6.(3)).
- In the case when P matches **skip**: For any environment $\rho \in \gamma(a)$, we have $[\![P]\!]_{\mathtt{mc}}(\rho) = \{\rho\}$ and $[\![\mathbf{skip}]\!]^\sharp_{\mathtt{mc}}(a) = a$, so $[\![P]\!]_{\mathtt{mc}}(\rho) \subseteq \gamma(a)$.
- In the case when P matches **tick**: For any environment $\rho \in \gamma(a)$,

(1) when $\rho(\mathtt{clock}) < \mathtt{mc}$, we have $[\![P]\!]_{\mathtt{mc}}(\rho) = \{\rho[\mathtt{clock} \mapsto \rho(\mathtt{clock}) + 1]\}$, then, by Def. 6.(3), $\{\rho[\mathtt{clock} \mapsto \rho(\mathtt{clock}) + 1]\} \subseteq \gamma(\textsc{tick}(a))$; since $\rho(\mathtt{clock}) + 1 \in [0; \mathtt{mc}]$ and by Def. 6.(3), we conclude that $\{\rho[\mathtt{clock} \mapsto \rho(\mathtt{clock}) + 1]\} \subseteq \gamma(\textsc{guard}(\mathtt{clock}, [0; \mathtt{mc}], \textsc{tick}(a)))$;
(2) otherwise, we have $[\![P]\!]_{\mathtt{mc}}(\rho) = \emptyset$, then $[\![P]\!]_{\mathtt{mc}}(\rho) \subseteq \gamma([\![\mathbf{tick}]\!]^\sharp_{\mathtt{mc}}(a))$.

- In the case when P matches **if** $(V \geq 0)$ $\{P_1\}$ **else** $\{P_2\}$:
  For any environment $\rho \in \gamma(a)$:

(1) when $\rho(V) \geq 0$, by Def. 6.(3), we have $\rho \in \textsc{guard}(V, [0; +\infty[, a)$; then, by structural induction, $[\![P_1]\!]_{\mathtt{mc}}(\rho) \subseteq \gamma([\![P_1]\!]^\sharp_{\mathtt{mc}}(\textsc{guard}(V, [0; +\infty[, a)))$;
(2) when $\rho(V) < 0$, the same proof applies if we replace $[0; +\infty[$ with $]-\infty; 0]$ and P_1 with P_2.

- In the case when P matches **while** $(V \geq 0)$ $\{P'\}$: For any environment $\rho \in \gamma(a)$: we introduce $f = X \mapsto \{\rho\} \cup (\bigcup\{[\![P']\!]_{\mathtt{mc}}(\rho') \mid \rho' \in X, \rho'(V) \geq 0\})$ and $f^\sharp = X \mapsto a \sqcup [\![P']\!]^\sharp_{\mathtt{mc}}(\textsc{guard}(V, [0; +\infty[, X))$; the map f is a $\cup$-complete endomorphism, moreover, by structural induction and thanks to Def. 6.(3) and Def. 6.(1), we have $(f \circ \gamma)(a) \subseteq (\gamma \circ f^\sharp)(a)$ for any $a \in \mathit{Env}^\sharp$; then by Thm. 11, we have $\mathrm{lfp}(f) \subseteq \gamma(\mathrm{lfp}^\sharp(f^\sharp))$; let ρ' be an environment in $[\![\textbf{while } (V \leq 0) \{P'\}]\!]_{\mathtt{mc}}(\rho)$, we have $\rho' \in \gamma(\mathrm{lfp}^\sharp(f^\sharp))$ and $\rho'(V) < 0$, so by Def. 6.(3), we have $\rho' \in \gamma([\![\textbf{while } (V \geq 0) \{P'\}]\!]^\sharp_{\mathtt{mc}}(\rho))$.
- In the case when P matches $P_1; P_2$: For any environment $\rho \in \gamma(a)$, by structural induction, we have $[\![P_1]\!]_{\mathtt{mc}}(\rho) \subseteq \gamma([\![P_1]\!]^\sharp_{\mathtt{mc}}(a))$; then, for any environment $\rho' \in [\![P_1]\!]_{\mathtt{mc}}(\rho)$, by structural induction, $[\![P_2]\!]_{\mathtt{mc}}(\rho') \subseteq \gamma([\![P_2]\!]^\sharp_{\mathtt{mc}}([\![P_1]\!]^\sharp_{\mathtt{mc}}(a))$; thus, $[\![P_1; P_2]\!]_{\mathtt{mc}}(\rho) \subseteq \gamma([\![P_2]\!]^\sharp_{\mathtt{mc}}([\![P_1]\!]^\sharp_{\mathtt{mc}}(a))$. □

5 Generic extension

We now show how an analysis can be extended to take into account a given class of digital filters. First we give an intuitive description of what we want to do. Then we introduce all the primitives that we need to build such a domain. Next we define a generic domain of constraints, this domain is independent from the other abstract domains which are handled with by the analysis. At last, we build an approximate reduced product between our generic domain and an underlying domain.

5.1 Intuitions

A filter class is given by: the number p of outputs and the number q of inputs that are involved in the computation of the next output, a (generic/symbolic) description of the recursive function with parameters, and some conditions over these parameters.

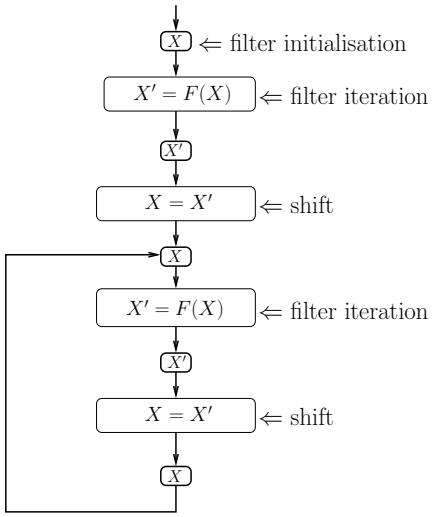
For instance, in the case of the second order filter, the next output depends on the last two outputs (i.e. $p = 2$) and on the last three inputs (i.e. $q = 3$), the next output is given by a function F where $F(S_{n-1}, S_{n-2}, E_n, E_{n-1}, E_{n-2}) = aS_{n-1} + bS_{n-2} + cE_n + dE_{n-1} + eE_{n-2}$ where a, b, c, d, and e are parameters; moreover, the parameters should satisfy $a^2 + 4b < 0$.

To analyze the behavior of such a filter, we have to consider some generic properties that tie the variables that are involved in two successive iterations of the filter (in our example, just before the filter iteration our property relates the values of variables S_{n-1}, S_{n-2}, E_{n-1}, and E_{n-2}, whereas, after the filter iteration, our property relates the values of variables S_n, S_{n-1}, E_n, and

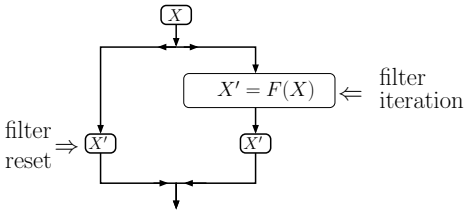
E_{n-1}). The abstract property also contains the generic description of the filter that is iterated (i.e. the definition of F) and an abstraction of the input stream. The abstraction of the input stream is parametric: the more concrete choice is a set of input streams; when computing a history-sensitive abstraction (to take into account the cancellation effect), we can abstract a set of input streams by a bound on their absolute value; last, when computing a history-insensitive abstraction we only keep invariants about the variables that are involved in the output stream (here a bound on the value of the expression $\sqrt{|S_{n-1}^2 - aS_{n-1}S_{n-2} - bS_{n-2}^2|}$ just before the iteration of the filter and a bound on the value of the expression $\sqrt{|S_n^2 - aS_nS_{n-1} - bS_{n-1}^2|}$ just after the iteration of the filter).

In Fig. 7, we describe both control flow patterns that have to be taken into account when dealing with a given filter. In this figure, the symbols X and X' denote two tuples of variables. More precisely, the symbol X denotes the tuple of the variables that are tied before an iteration of the filter and the symbol X' denotes the tuple of the variables that are tied after an iteration of the filter. The instruction $X' = F(X)$ computes an iteration of the filter. In the abstract, the abstract property that relates the variables in the tuple X before the instruction gives another abstract property that relates the variables in the tuple X' after the instruction. If, when we encounter an instance of the instruction $X = F(X')$, we have no constraint over the tuple of variables X, we have to build such a constraint by weakening the constraints that we have in the underlying domain. This way, we capture filters re-initializations. Between two iterations of the filter, the variables are shifted which is described by the instruction $X' = X$. In Fig. 7(a), at the first iteration of the filter, we have no constraints over X, so we build such a constraint by weakening the constraints in the underlying domain. Then we iterate the filter to obtain a constraint over the variables in the tuple X'. The filter iteration is followed by a shift instruction. We get back a constraint that relates the variables in the tuple X. Then, for each next iteration of the filter, a constraint over the variables in the tuple X is available. First it is translated into a constraint over the variables in X' when iterating the filter. Then it gives back a constraint over the variables in X when shifting the variables. Each filter may be reset. Filter resets are taken into account when merging control flow paths. In Fig. 7(b), the right branch corresponds to a regular filter iteration whereas the left one describes a filter reset. When we merge the two control flow paths, we have a constraint that ties the variables in the tuple X' in the right control flow path whereas we have no corresponding constraint in the left control flow path. So we build the missing constraint by weakening the constraints that we have in the underlying domain.

This way, to deal with filters, we only have to compute sound counterpart to filters iteration and variables shift. We also require a primitive to weaken



(a) filter iterations.



(b) filter reset.

Fig. 7. Filter implementation control flow.

the constraints of the underlying domain to compute the constraints that are missing before a filter iteration or when merging control flow paths.

In the next subsection, we describe a generic domain to describe both filter constraints and primitives to handle with these constraints.

5.2 Domain and primitives

A filter domain collects constraints that relate some variables, some parameters, and abstract ranges. The variables in such constraints are the variables that will be used again at the next filter iteration. The parameters are the filter parameters. The abstract ranges provide an approximation of both filter input and output. Depending on the abstraction, abstract ranges may encode the input stream itself, a bound on the input stream, or some abstract properties about the last outputs (which is in some sense also an approximation of the input stream). For instance, in the case of the second order filter, the variables may be the last two outputs, the parameters some directing coefficients, and the abstract range a radius inferred during the analysis. Let m be the number of variables to be used and n the number of the filter parameters. The abstract domain maps tuples in $\mathcal{T} = \mathcal{V}^m \times \mathbb{R}^n$ to elements in the set $\mathcal{B}$ of abstract ranges. In case of diverging filters, the abstract range may be related with the clock counter. So we introduce a concretization $\gamma_{\mathcal{B}}$ that relates the filter parameters and an abstract range to a subset of $\wp(\{x_i \mid 1 \leq i \leq m\} \uplus \texttt{clock} \to \mathbb{R})$. The variable x_i denotes the value of the i-th variable involved in the computation of the next output whereas the variable `clock` denotes the value of the clock counter. The set of environments

that satisfy a constraint $c = ((X_1, \ldots, X_m, \alpha_1, \ldots, \alpha_n), b) \in \mathcal{T} \times \mathcal{B}$ is given by the concretization $\Gamma_{\mathcal{B}}(c)$ that is defined as the set of all environments ρ for which there exists a function $f \in \gamma_{\mathcal{B}}((\alpha_1, \ldots, \alpha_n), b)$ such that $\rho(X_i) = f(x_i)$ for any i between 1 and m and such that $\rho(\texttt{clock}) = f(\texttt{clock})$. An operator $\sqcup_{\mathcal{B}} \in \mathcal{B} \times \mathcal{B} \to \mathcal{B}$ is used to merge constraints related to the same tuple, but coming from distinct flows. Another operator $\sqcap_{\mathcal{B}} \in \wp(\mathcal{B}) \setminus \{\emptyset\} \to \mathcal{B}$ is used to take the meet of constraints related to the same tuple [6].

In order to perform abstract iterations, we introduce an element $\bot_{\mathcal{B}} \in \mathcal{B}$ that provides the basis of iterations. We also use extrapolation operators $\triangledown_{\mathcal{B}} \in \mathcal{B} \times \mathcal{B} \to \mathcal{B}$ and $\triangle_{\mathcal{B}} \in \mathcal{B} \times \mathcal{B} \to \mathcal{B}$ to approximate concrete post-fixpoint.

When computing assignment $X = E$ in the abstract, we need to know which constraints are required in the precondition. For that purpose, we introduce a primitive $\textsc{rlvt} \in \mathcal{E} \mapsto \wp(\mathcal{T})$. The set $\textsc{rlvt}(E)$ denotes the set of tuples corresponding to the constraints that are modified by this assignment (usually the tuples containing some variables that occurs in the expression E). Then each constraint associated with a tuple in $\textsc{rlvt}(E)$ provides another constraint that tied another tuple of variables. To compute this constraint we need to extract some information from the expression E, such as the filter coefficients and a bound on the current input. To achieve this goal, we may have to use the constraints in the underlying domain. We introduce the set $\textsc{info}$ of the data that can be collected from an assignment. Then we introduce the primitive $\textsc{pt}$ that takes an assignment, a tuple t of variables and of filter parameters in $\mathcal{T}$, and interval constraints $\rho^\sharp \in \mathcal{V} \cup \{\texttt{clock}\} \to \mathcal{I}$ from the underlying domain. Then the primitive $\textsc{pt}$ returns a pair made of a tuple of variables and of filter parameters, and the information that is required to compute the iteration of the filter. Last, we introduce a primitive $\delta \in \textsc{info} \times \mathcal{B} \to \mathcal{B}$. The primitive δ collects the information given by the primitive $\textsc{pt}$ and the current abstract ranges of the filter. Then it updates the abstract ranges. When computing a filter iteration, some abstract properties may constrain the same tuple of variables: in such a case we use the abstract primitive $\sqcap_{\mathcal{B}}$ to take the meet of these constraints.

To deal with diverging filters, filter constraints may relate some values to the clock counter. In such a case, we introduce an abstract primitive $\texttt{clock} \in \mathcal{B} \to \mathcal{B}$ to simulate the increment of the clock counter.

Last primitives describe the interaction between a filter domain and the underlying domain. A primitive $\textsc{build} \in \mathbb{R}^n \times \{x_i \mid 1 \leq i \leq m\} \to \mathcal{I} \times \{x_i \mid 1 \leq i \leq m\}^2 \to \mathcal{B}$ receives the filter parameters, some ranges for both the filter initial inputs and the filter initial outputs, and equality constraints from the

[6] In practice, the result of a meet operator may depend on the order of the arguments, so a function from $\mathcal{B}^+$ (where $\mathcal{B}^+$ denotes the set of finite and non empty sequences of elements in $\mathcal{B}$) into $\mathcal{B}$ would have been more appropriate.

underlying domain about these initial values. Then it builds an abstract range for the filter. It is useful either at the filter initialization, or when the filter is reset. Conversely, the function $\text{TO} \in \mathbb{R}^n \times \mathcal{B} \times \mathcal{I} \to (\{x_i \mid 1 \leq i \leq m\} \to \mathcal{I})$ extracts information about the range of the output stream to be passed to the underlying domain. To deal with diverging filters, it is necessary to relate the range of the output stream to the clock counter. In $\text{TO}(p, a, i)$, p encodes the filter parameters, a denotes some abstract ranges for the filter output, and i is the range of the clock counter.

Def. 13 formalizes the definition of these primitives and gives their soundness requirements.

Definition 13 (Generic extension) *An abstract extension is given by a tuple* $(\mathcal{T}, \mathcal{B}, \gamma_{\mathcal{B}}, \sqcup_{\mathcal{B}}, \sqcap_{\mathcal{B}}, \bot_{\mathcal{B}}, \nabla_{\mathcal{B}}, \Delta_{\mathcal{B}}, \text{RLVT}, \text{INFO}, \text{PT}, \delta, \text{TICK}, \text{BUILD}, \text{TO})$ *so that:*

(1) structure:
- $\mathcal{T} = \mathcal{V}^m \times \mathbb{R}^n$*, where* $m, n \in \mathbb{N}$*;*
- $\mathcal{B}$ *is a set of abstract ranges;*
- $\gamma_{\mathcal{B}} \in \mathbb{R}^n \times \mathcal{B} \to \wp(\{x_i \mid 1 \leq i \leq m\} \uplus \{clock\} \to \mathbb{R})$ *is a concretization map;*
- $\forall p \in \mathbb{R}^n, \forall a, b \in \mathcal{B},\ \gamma_{\mathcal{B}}(p, a) \cup \gamma_{\mathcal{B}}(p, b) \subseteq \gamma_{\mathcal{B}}(p, a \sqcup_{\mathcal{B}} b)$*;*
- $\forall p \in \mathbb{R}^n, \forall A \in \wp(\mathcal{B}) \setminus \{\emptyset\},\ \bigcap\{\gamma_{\mathcal{B}}(p, a) \mid a \in A\} \subseteq \gamma_{\mathcal{B}}(p, \sqcap_{\mathcal{B}} A)$*;*

(2) extrapolation primitives:
- $\bot_{\mathcal{B}}$ *is an abstract element in* $\mathcal{B}$*;*
- $\nabla_{\mathcal{B}}$ *is a widening operator such that:* $\forall a, b \in \mathcal{B},\ \forall p \in \mathbb{R}^n,\ \gamma_{\mathcal{B}}(p, a) \cup \gamma_{\mathcal{B}}(p, b) \subseteq \gamma_{\mathcal{B}}(p, a \nabla_{\mathcal{B}} b)$*; and* $\forall (a_i) \in \mathcal{B}^{\mathbb{N}}$*, the sequence* (a_i^{∇}) *defined by* $a_0^{\nabla} = a_0$ *and* $a_{i+1}^{\nabla} = a_i^{\nabla} \nabla_{\mathcal{B}} a_{i+1}$ *is ultimately stationary;*
- $\Delta_{\mathcal{B}}$ *is a narrowing operator such that:* $\forall a, b \in \mathcal{B},\ \forall p \in \mathbb{R}^n, \gamma_{\mathcal{B}}(p, a) \cap \gamma_{\mathcal{B}}(p, b) \subseteq \gamma_{\mathcal{B}}(p, a \Delta_{\mathcal{B}} b)$*;* $\forall (a_i) \in \mathcal{B}^{\mathbb{N}}$*, the sequence* (a_i^{Δ}) *defined by* $a_0^{\Delta} = a_0$ *and* $a_{i+1}^{\Delta} = a_i^{\Delta} \Delta_{\mathcal{B}} a_{i+1}$*, is ultimately stationary;*

(3) transfer functions:
- ***assignments:*** RLVT *maps an expression* $E \in \mathcal{E}$ *to a subset* T *of* $\mathcal{T}$*;* INFO *is a set of filter parameters;* $\forall X \in \mathcal{V}, E \in \mathcal{E}, t \in T, \rho^\sharp \in \mathcal{V} \to \mathcal{I}$*, we have* $\text{PT}(X = E, t, \rho^\sharp) \in \mathcal{T} \times$ INFO *and* $\forall t \in \text{RLVT}(E), a \in \mathcal{B}$*, such that* $\rho \in \Gamma_{\mathcal{B}}(t, a) \cap \gamma_{\mathcal{I}}(\rho^\sharp)$*, we have*[7] $[\![X = E]\!]_{\texttt{mc}}(\rho) \subseteq \Gamma_{\mathcal{B}}(t', \delta(info, a))$*, with* $(t', info) = \text{PT}(X = E, t, \rho^\sharp)$*;*
- ***clock ticks:*** $\text{TICK}_{\mathcal{B}} \in \mathcal{B} \to \mathcal{B}$ *satisfies: for any tuple of parameters* $p \in \mathbb{R}^n$*, for any abstract range* $a \in \mathcal{B}$*, we have* $\{f[\texttt{clock} \mapsto f(\texttt{clock} + 1)] \mid f \in \gamma_{\mathcal{B}}(p, a)\} \subseteq \gamma_{\mathcal{B}}(\text{TICK}_{\mathcal{B}}(p, a))$*.*

(4) conversion primitives:
- ***filter constraint synthesis*** *from the underlying domain:*

[7] We recall that $\Gamma_{\mathcal{B}}(((X_i), p), a))$ is defined as the set of all environments ρ for which there exists a function $f \in \gamma_{\mathcal{B}}(p, b)$ such that $\rho(X_i) = f(x_i)$ for any i between 1 and m and such that $\rho(\texttt{clock}) = f(\texttt{clock})$.

$\forall p \in \mathbb{R}^n, \forall (f_I, \mathcal{R}) \in (\{x_i \mid 1 \leq i \leq m\} \to \mathcal{I}) \times \{x_i \mid 1 \leq i \leq m\}^2$, *for any mapping* $f \in \{x_i \mid 1 \leq i \leq m\} \uplus \{\texttt{clock}\} \to \mathbb{R}$, *if* $f(x_i) \in f_I(x_i)$ *for each* i *such that* $1 \leq i \leq m$ *and if* $f(x_i) = f(x_j)$ *for each* $(x_i, x_k) \in \mathcal{R}$, *then the mapping* f *belongs to the concretization* $\gamma_{\mathcal{B}}(p, \textsc{build}(p, f_I, \mathcal{R}))$*;*

- ***interval constraint synthesis:***
  $\textsc{to} \in \mathbb{R}^n \times \mathcal{B} \times \mathcal{I} \to (\{x_i \mid 1 \leq i \leq m\} \to \mathcal{I})$ *and for any tuple* $p \in \mathbb{R}^n$ *of parameters, any abstract range* $a \in \mathcal{B}$, *and any interval* I *for the clock counter, for any mapping* $f \in \{x_i \mid 1 \leq i \leq m\} \uplus \{\texttt{clock}\} \to \mathbb{R}$, *if* $f \in \gamma_{\mathcal{B}}(p, \mathcal{B})$ *and if* $f(\texttt{clock}) \in I$, *then we have* $f(x_i) \in \textsc{to}(p, a, I)(x_i)$ *for any integer* i *between* 1 *and* m.

The next subsection describes how to build an autonomous abstract domain from these primitives.

5.3 Abstract domain

We now build an abstraction from a generic extension. We first enrich the set $\mathcal{B}$ with an extra element $\top \notin \mathcal{B}$. That element denotes the fact that a tuple is tied to no constraint. We set $\gamma_{\mathcal{B}}(p, \top) = \{x_i \mid 1 \leq i \leq m\} \uplus \{clock\} \to \mathbb{R}$ and $\textsc{to}(t, \top, I) = [\_ \mapsto \mathbb{R}]$. We also lift other primitives of the domain so that they return $\top$ as soon as one of their arguments is $\top$. We also extend the definition of $\sqcap_{\mathcal{B}}$ by $\sqcap_{\mathcal{B}}(\emptyset) = \top$. The abstract domain $Env_F^\sharp = (\mathcal{T} \to \mathcal{B})$ is related to $\wp(Env)$ by the concretization function γ_F which maps $f \in Env_F^\sharp$ to the set of environments $\bigcap\{\Gamma_{\mathcal{B}}(t, f(t)) \mid t \in \mathcal{T}\}$ that satisfy all the constraints encoded by f. The operator $\sqcup_F$ applies component wise the operator $\sqcup_{\mathcal{B}}$. The $\bot_F$ element maps each tuple to the element $\top$. The operators $(\triangledown_F, \triangle_F)$ are defined component wise. The abstract assignment may require information about variable ranges in order to extract filter parameters (in particular the input values). The abstract assignment $\textsc{assign}_F^{\rho^\sharp}(X = E, f)$ of an abstract element f under the interval constraints $\rho^\sharp \in \mathcal{V} \to \mathcal{I}$, is given by the abstract element f' where:

(1) if E is a variable Y, each constraint containing Y gives a constraint for X: we take $f'(t) = f(\sigma t)$, where σ substitutes each occurrence of X by Y in the tuple t of variables and of parameters;
(2) otherwise, we remove each constraint involving X, and add each constraint corresponding to some filter iteration: we define $f'(t)$ as $f(t)$ whenever $t \in (\mathcal{V} \setminus \{X\})^m \times \mathbb{R}^n$, otherwise, we define[8] $f(t)$ as:

$$\sqcap_{\mathcal{B}}\{\delta(info, f(t_{-1})) \mid (t, info) = \textsc{pt}(X = E, t_{-1}, \rho^\sharp)\}.$$

[8] We recall that, by definition, $\sqcap_{\mathcal{B}}(\emptyset) = \top$

Since filter invariants do not rely on guards, we set $\textsc{guard}_F(X, I, f) = f$. Each tick of clock is applied with each filter constraint, so we define the primitive $\textsc{tick}_F$ component-wise.

We have no information about the clock counter, so the function $\textsc{range}_F(f)$ maps each variable X to the interval:

$$\bigcap \left\{ \textsc{to}_F((a_i), f(t), \mathbb{R})(x_i) \;\middle|\; \begin{array}{l} \exists i \in \mathbb{N}, \exists t \in \mathcal{T}, 1 \leq i \leq m, \\ t \text{ matches } (Y_1, \ldots, Y_m, a_1, \ldots, a_n), \\ Y_i = X \end{array} \right\};$$

since filters do not generate syntactic equality constraints, we take $\textsc{equ}_F = \emptyset$. We also set $\textsc{assign}_F(X = E, f) = \textsc{assign}_F^{\textsc{range}_F(f)}(X = E, f)$. We also ignore the constraints that are passed by the other domains, this way we take $\textsc{red}_F(\rho^\sharp, a) = a$. We define the tuple $\mathcal{A}_F$ as: ($Env_F^\sharp$,γ_F,$\sqcup_F$,$\bot_F$,$\triangledown_F$,$\triangle_F$,$\textsc{assign}_F$, $\textsc{guard}_F$,$\textsc{tick}_F$,$\textsc{range}_F$,$\textsc{red}_F$,$\textsc{equ}_F$).

Theorem 14 *The tuple $\mathcal{A}_F$ is an abstraction.*

5.4 Product and approximate reduced product

Unfortunately the abstraction $\mathcal{A}_F$ cannot compute any constraint, mainly because of inaccurate assignments and guards, and because abstract iterations always return the top element. Hence we use a reduced product between this abstraction and an existing underlying abstraction to refine and improve the analysis.

Let ($Env_0^\sharp$,γ_0,$\sqcup_0$,$\bot_0$,$\triangledown_0$,$\triangle_0$,$\textsc{assign}_0$,$\textsc{guard}_0$,$\textsc{tick}_0$,$\textsc{range}_0$,$\textsc{red}_0$,$\textsc{equ}_0$)
be an abstraction that we denote by $\mathcal{A}_0$. We first introduce the product of the two abstractions. The abstract domain $Env^\sharp$ is the Cartesian product $Env_0^\sharp \times Env_F^\sharp$. The operator $\sqcup$, the extrapolation operators ($\bot, \triangledown, \triangle$), the transfer functions ($\textsc{assign}, \textsc{guard}, \textsc{tick}$), the function $\textsc{red}$ and are all defined pairwise. The concretization and the remaining refinement primitives are defined as follows:

$$\begin{cases} \gamma(a, f) = \gamma_0(a) \cap \gamma_F(f), \\ \textsc{equ}(a, f)(X, Y) \iff \textsc{equ}_0(a)(X, Y) \text{ or } \textsc{equ}_F(f)(X, Y), \\ \textsc{range}(a, f)(X) = \textsc{range}_0(a)(X) \cap A(X), \end{cases}$$

where $A(X)$ is defined as follows:

$$\bigcap \left\{ \mathrm{TO}_F((p_j), f(t), \mathrm{RANGE}_0(a)(\texttt{clock}))(x_i) \;\middle|\; \begin{array}{l} \exists i \in \mathbb{N},\ 1 \leq i \leq m, \\ \exists t \in \mathcal{T},\ t \text{ matches} \\ ((Y_j)_{1\leq j\leq m}, (p_j)_{1\leq j\leq n}) \\ \text{and } Y_i = X \end{array} \right\},$$

which corresponds to taking the meet of collected information. We notice that we have take into account the value of the clock counter when abstracting the range of the output stream.

We refine abstract assignments and binary operators:

- we want to compute the assignment $X = E$, starting from the precondition (a, f):
  - before the assignment, we synthesize the filter constraints that are relevant for the assignment and that are missing: so we define $f'(t)$ where t matches $((X_k)_{1\leq k\leq m}, (p_k)_{1\leq k\leq n})$ as follows:

$$\begin{cases} \mathrm{BUILD}((p_i), \phi, \mathcal{R}) & \text{if } t \in \mathrm{RLVT}(E) \text{ and } f(t) = \top, \\ f(t) & \text{otherwise,} \end{cases}$$

    where $\phi = [x_k \mapsto \mathrm{RANGE}_0(a)(X_k)]$ and $(x_k, x'_k) \in \mathcal{R}$ if and only if $(X_k, X'_k) \in \mathrm{EQU}_0(a)$;
  - during the assignment, we use the underlying constraints to interpret the expression: so we define (a'', f'') as follows:

$$(\mathrm{ASSIGN}_0(X = E, a), \mathrm{ASSIGN}_F^{\mathrm{RANGE}_0(a)}(X = E, f'));$$

  - and after the assignment, we use the filter constraints to refine the properties in the underlying domain: so we define $\mathrm{ASSIGN}'(X = E, (a, f))$ by $(\mathrm{RED}_0(\mathrm{RANGE}(a'', f''), a''), f'')$.
- we want to apply a binary operator $\circledast'$ (where $\circledast \in \{\sqcup, \triangle, \triangledown\}$) with two abstract elements (a_1, f_1) and (a_2, f_2):
  - before applying the binary operator, we refine the filter constraints so that both arguments constrain the same set of tuples: so for $i \in \{1; 2\}$, we define $f'_i(t)$ where t matches $((X_k)_{1\leq k\leq m}, (p_k)_{1\leq k\leq n})$ as follows:

$$\begin{cases} \mathrm{BUILD}((p_i), \phi_i, \mathcal{R}_i) & \text{if } f_i(t) = \top \text{ and } f_{2-i}(t) \neq \top, \\ f_i(t) & \text{otherwise,} \end{cases}$$

    where $\phi_i = [x_k \mapsto \mathrm{RANGE}_0(a_i)(X_k)]$ and $(x_k, x'_k) \in \mathcal{R}_i$ if and only if $(X_k, X'_k) \in \mathrm{EQU}_0(a_i)$;

- we then apply the classical operator: we define (a'', f'') as $(a_1 \circledast_0 a_2, f_1' \circledast_F f_2')$;
- after applying the binary operator, we use the filter constraints to refine the underlying domain properties: we define $(a_1, f_1) \circledast' (a_2, f_2)$ by $(\text{RED}_0(\text{RANGE}(a'', f''), a''), f'')$.

We set $\mathcal{A} = (Env^\sharp, \gamma, \sqcup', \bot, \triangledown', \vartriangle', \text{ASSIGN}', \text{GUARD}, \text{TICK}, \text{RANGE}, \text{EQU}, \text{RED})$.

Theorem 15 *The tuple $\mathcal{A}$ is an abstraction.*

PROOF. We sketch the proof of Th. 15: all soundness requirements come from the soundness of both the underlying domain and the filter domain. During an extrapolation iteration (ascending or descending), the set of filter constraints (i.e., the set of the tuples of variables and of filter parameters, that are not mapped into $\top$) is ultimately stationary (since the number of constraints is increasing, whereas $\mathcal{V}$ is finite); then each sequence of constraints associated to a given tuples of variables and of parameters is ultimately stationary; once the information that refines the underlying domain is fixed, the underlying domain termination criteria in Def. 6.(2) apply. □

6 Numerical abstract domains

6.1 Abstracting real numbers

Until now, we have only used real numbers. In order to implement numerical abstract domains, we use a finite subset $\mathbb{F}$ of real numbers (such as the floating-point numbers) that contains the set of numbers $\{0, 1\}$ and that is closed under negation. The set $\overline{\mathbb{F}}$ is obtained by enriching the set $\mathbb{F}$ with two extra elements $+\infty$ and $-\infty$ that respectively describe the reals that are greater (resp. smaller) than the greatest (resp. smallest) element of $\mathbb{F}$. We denote the set $\{x \in \mathbb{F} \mid x \geq 0\}$ by $\mathbb{F}^+$ and the set $\mathbb{F}^+ \cup \{+\infty\}$ by $\overline{\mathbb{F}}^+$. The result of a computation on elements of $\overline{\mathbb{F}}$ may be not in $\overline{\mathbb{F}}$. So we suppose that we are given a rounding function $\lceil \_ \rceil \in \mathbb{R} \to \overline{\mathbb{F}}$ such that $\lceil x \rceil \geq x$, for any $x \in \mathbb{R}$. We also suppose that $\lceil x \rceil \leq 0$, for any $x \leq 0$. Then we define the rounding function $\lfloor \_ \rfloor \in \mathbb{R} \to \overline{\mathbb{F}}$ as $\lfloor x \rfloor = -\lceil -x \rceil$. Thanks to the properties of the function $\lceil \_ \rceil$, we know that for any element $x \in \mathbb{R}$, we have $\lfloor x \rfloor \leq x$ and that for any $x \geq 0$, we have $\lfloor x \rfloor \geq 0$. In the following, we will often divide some expressions by other expressions of the form $1 - \alpha$ when $0 < \alpha < 1$. In order to achieve these kind of computation steps, we assume that for any $x > 1$, $\lfloor x - 1 \rfloor > 0$ and that for any $x < 1$, $\lfloor 1 - x \rfloor > 0$. Roughly speaking it means that there are at least as many representable numbers around 0 as around 1 which is usually

true in floating-point arithmetics because of denormalized numbers. The set $\overline{\mathbb{F}}$ is also endowed with a finite widening ramp L [2, Sect. 7.1.2]. For any two elements a and b in $\overline{\mathbb{F}}$ we set $a\nabla_{\overline{\mathbb{F}}}b = min\{l \in L \cup \{a; +\infty\} \mid max(a,b) \leq l\}$.

We now introduce two integers q and r in order to tune our extrapolation strategy. We introduce the domain $\mathcal{F}_{q,r}$ as the set $\overline{\mathbb{F}} \times \mathbb{Z}$. In a pair $(e,k) \in \mathcal{F}_{q,r}$, the representable number e denotes a set of real numbers whereas the integer k encodes an abstraction of the extrapolation history of the constraint. More precisely, when in upward iteration, the number k encodes the number of time the constraint has been unstable without having been widened. When in downward iteration, it counts the number of time the constraint has been narrowed. The domain $\mathcal{F}_{q,r}$ is related to the concrete domain $\wp(\mathbb{R})$ via the concretization $\gamma_{\mathcal{F}_{q,r}}$ that maps any pair (e,k) into the set of the reals r such that $|r| \leq e$. The bottom element $\bot_{\mathcal{F}_{q,r}}$ is $(-\infty, 0)$. The binary operators $\sqcup_{\mathcal{F}_{q,r}}$, $\nabla_{\mathcal{F}_{q,r}}$, and $\triangle_{\mathcal{F}_{q,r}}$ are defined as follows:

- $(a_1,k_1) \sqcup_{\mathcal{F}_{q,r}} (a_2,k_2) = (max(a_1,a_2), 0)$;
- $(a_1,k_1)\nabla_{\mathcal{F}_{q,r}}(a_2,k_2) = \begin{cases} (a_1,k_1) & \text{if } a_1 \geq a_2 \\ (a_2,k_1+1) & \text{if } a_2 > a_1 \text{ and } k_1 < q \\ (a_1 \nabla_{\overline{\mathbb{F}}} a_2, 0) & \text{otherwise;} \end{cases}$
- $(a_1,k_1)\triangle_{\mathcal{F}_{q,r}}(a_2,k_2) = \begin{cases} (a_1,k_1) & \text{if } a_1 \leq a_2 \text{ or } k_1 \leq (-r) \\ (a_2, min(k_1,0)-1) & \text{if } a_2 < a_1 \text{ and } k_1 > (-r); \end{cases}$

A constraint is only widened if it has been unstable q times since its last widening. On the other side, narrowing stops after r iterations.

6.2 Approximating the iterates of numerical functions

When analyzing filters, we iterate functions $f \in \mathbb{R}^2 \to \mathbb{R}$ of the form $f(i,x) = \alpha x + i$. In $f(i,x)$, the variable i encodes some information about the input stream, whereas the variable x encodes some information about the output stream. This way, an iteration of f is a sequence of the form $u_0 = i_0$, $u_{n+1} = f(i_{n+1}, u_n)$, for a given input sequence (i_n).

When $|\alpha| < 1$, for any $i \in \mathbb{R}$ the function $[x \mapsto (i,x)]$ is convex and we know explicitly a real $f_l(i)$ such that $f(i, f_l(i)) \leq f_l(i)$. In such a case, we can use the function f_l to accelerate the abstraction of the iterates of f. We propose a solution which is sound (even if the function f_l is not implemented correctly) and accurate (we do not jump to the limit during the first unrolled iterates and we do not jump above the limit when the input stream is constant) and terminates.

When the function is not convex, the iterates may diverge. Nevertheless, we

can use the arithmetic-geometric progression domain [10] to relate the value of the iterates to the value of the clock counter.

6.2.1 Generic domain

We now define a parametric domain for abstracting the iterates of a function f of the form $f(x,i) = \alpha x + i$. The abstract domain $\mathcal{F}^*$ is a generic abstract domain to establish relations between the clock counter and a real value. Thus, it is related to $\wp(\mathbb{R}^2)$ via a concretization map $\gamma_{\mathcal{F}^*}$. The domain $\mathcal{F}^*$ is fitted with the following binary operators: an abstract union $\sqcup_{\mathcal{F}^*}$, a widening operator $\triangledown_{\mathcal{F}^*}$, and a narrowing operator $\triangle_{\mathcal{F}^*}$. It also contains a bottom element $\bot_{\mathcal{F}^*}$ that is the basis of abstract iterations. An operator $\sqcap_{\mathcal{F}^*}$ takes the meet of abstract elements. We also require two primitives $\text{AFF}_{\mathcal{F}^*}$ and $\text{TICK}_{\mathcal{F}^*}$ to handle respectively with the iteration of affine functions and with the clock counter increment. Last, the primitive $\text{C}_{\overline{\mathbb{F}}\to\mathcal{F}^*}$ convert a representable number into a property in the domain $\mathcal{F}^*$. Conversely, the primitive $\text{C}_{\mathcal{F}^*\to\overline{\mathbb{F}}}$ weakens an abstract property in $\mathcal{F}^*$ to get a sound bound in $\mathbb{F}$ over the abstracted value (this bound depends on the range for the clock counter).

Definition 16 (Iteration domain) *An iteration domain is given by a tuple* $(\mathcal{F}^*, \gamma_{\mathcal{F}^*}, \sqcup_{\mathcal{F}^*}, \sqcap_{\mathcal{F}^*}, \bot_{\mathcal{F}^*}, \triangledown_{\mathcal{F}^*}, \triangle_{\mathcal{F}^*}, \text{AFF}_{\mathcal{F}^*}, \text{TICK}_{\mathcal{F}^*}, \text{C}_{\overline{\mathbb{F}}\to\mathcal{F}^*}, \text{C}_{\mathcal{F}^*\to\overline{\mathbb{F}}})$ *which satisfies:*

(1) structure:
- $\mathcal{F}^*$ *is an abstract domain of properties;*
- $\gamma_{\mathcal{F}^*} \in \mathcal{F}^* \to \wp(\mathbb{R}^2)$ *is a concretization map;*
- $\forall a, b \in \mathcal{F}^*,\ \gamma_{\mathcal{F}^*}(a) \cup \gamma_{\mathcal{F}^*}(b) \subseteq \gamma_{\mathcal{F}^*}(a \sqcup_{\mathcal{F}^*} b)$*;*
- $\forall A \in \wp(\mathcal{F}^*) \setminus \emptyset,\ \bigcap\{\gamma_{\mathcal{F}^*}(a) \mid a \in A\} \subseteq \gamma_{\mathcal{F}^*}(\sqcap_{\mathcal{F}^*} A)$*;*

(2) extrapolation primitives:
- $\bot_{\mathcal{F}^*}$ *is an element in* $\mathcal{F}^*$*;*
- $\triangledown_{\mathcal{F}^*}$ *is a widening operator such that:* $\forall a, b \in \mathcal{F}^*,\ \gamma_{\mathcal{F}^*}(a) \cup \gamma_{\mathcal{F}^*}(b) \subseteq \gamma_{\mathcal{F}^*}(a \triangledown_{\mathcal{F}^*} b)$*; and* $\forall (a_i) \in \mathcal{F}^{*\mathbb{N}}$*, the sequence* $(a_i^\triangledown)$ *defined by* $a_0^\triangledown = a_0$ *and* $a_{n+1}^\triangledown = a_n^\triangledown \triangledown_{\mathcal{F}^*} a_{n+1}$ *is ultimately stationary;*
- $\triangle_{\mathcal{F}^*}$ *is a narrowing operator such that:* $\forall a, b \in \mathcal{F}^*,\ \gamma_{\mathcal{F}^*}(a) \cap \gamma_{\mathcal{F}^*}(b) \subseteq \gamma_{\mathcal{F}^*}(a \triangle_{\mathcal{F}^*} b)$*;* $\forall (a_i) \in \mathcal{F}^{*\mathbb{N}}$*, the sequence* $(a_i^\triangle)$ *defined by* $a_0^\triangle = a_0$ *and* $a_{n+1}^\triangle = a_n^\triangle \triangle_{\mathcal{F}^*} a_{n+1}$*, is ultimately stationary;*

(3) transfer functions:
- ***function iteration:*** *for any real numbers* α *and* β *such that* $\alpha > 0$ *and* $\beta \geq 0$*,* $\text{AFF}_{\mathcal{F}^*}[\alpha, \beta]$ *is a function in* $\mathcal{F}^* \to \mathcal{F}^*$ *such that: for any* $a \in \mathcal{F}^*$*,*$(\mathtt{vc}, x) \in \gamma_{\mathcal{F}^*}(a)$*, we have:* $(\mathtt{vc}, \alpha x + \beta) \in \gamma_{\mathcal{F}^*}(\text{AFF}_{\mathcal{F}^*}[\alpha, \beta](a))$*;*
- ***clock ticks:*** $\text{TICK}_{\mathcal{F}^*}$ *is a function in* $\mathcal{F}^* \to \mathcal{F}^*$ *such that:* $\forall a \in \mathcal{F}^*$*,* $\{(n+1, x) \mid (n, x) \in \gamma_{\mathcal{F}^*}(a)\} \subseteq \gamma_{\mathcal{F}^*}(\text{TICK}_{\mathcal{F}^*}(a))$*.*

(4) conversion primitives:
- $\text{C}_{\overline{\mathbb{F}}\to\mathcal{F}^*}$ *is a function in* $\overline{\mathbb{F}} \to \mathcal{F}^*$ *such that for any representable number*

$f \in \overline{\mathbb{F}}$, *we have* $(\mathbb{R}^+ \times \{r \mid |r| \leq f\}) \subseteq \gamma_{\mathcal{F}^*}(\mathrm{C}_{\overline{\mathbb{F}} \to \mathcal{F}^*}(f))$;

- $\mathrm{C}_{\mathcal{F}^* \to \overline{\mathbb{F}}}$ *is a function on* $\mathcal{F}^* \times \mathcal{I} \to \overline{\mathbb{F}}$ *such that for any abstract element* $a \in \mathcal{F}^*$ *and any interval* $I \in \mathcal{I}$ *for the clock counter, any real number* r *such that there exists a clock value* $\mathtt{vc} \in I$ *such that* $(\mathtt{vc}, r) \in \gamma_{\mathcal{F}^*}(a)$, *we have* $|r| \leq \mathrm{C}_{\mathcal{F}^* \to \overline{\mathbb{F}}}(a, I)$.

6.2.2 A domain for accelerating the iterates of convex functions

We introduce an abstract domain $\mathcal{F}^*_{q,r}$ to accelerate the iterates of convex functions. The domain $\mathcal{F}^*_{q,r}$ is the Cartesian product between $\mathcal{F}_{0,r}$ and $\mathcal{F}_{q,0}$. It is related to $\wp(\mathbb{R}^2)$ via a concretization map $\gamma_{\mathcal{F}^*_{q,r}}$ that maps elements (a, b) to $\mathcal{R} \times (\gamma_{\mathcal{F}_{0,r}}(a) \cap \gamma_{\mathcal{F}_{q,0}}(b))$. This way, it ignores the clock counter and each abstract elements describe two constraints. The element $\bot_{\mathcal{F}^*_{q,r}}$ is the pair $(\bot_{\mathcal{F}_{0,r}}, \bot_{\mathcal{F}_{q,0}})$. Abstract union, abstract intersection, and narrowing are defined pairwise. The widening operator is defined as follows:

$$((a_1, k_1), (b_1, l_1)) \triangledown_{q,r} ((a_2, k_2), (b_2, l_2)) = ((min(a_3, b_3), 0), (b_3, l_3)),$$

where $(a_3, k_3) = (a_1, k_1) \triangledown_{\mathcal{F}_{0,r}} (a_2, k_2)$ and $(b_3, l_3) = (b_1, l_1) \triangledown_{\mathcal{F}_{q,0}} (b_2, l_2)$.

The elements a and b in the pair (a, b) are intended to describe the same set of reals. Nevertheless, they will be iterated using distinct extrapolation strategies and distinct transfer functions. The element a will be computed via a transfer function $f \in \overline{\mathbb{F}} \to \overline{\mathbb{F}}$, whereas b will be computed via a relaxed transfer function that try to guess the fixpoint of f. The element b is used to refine the element a after widening, in case it becomes more precise. This allows computing arbitrary thresholds during iterations. To ensure termination, it is necessary also to widen b, but this is not done at each iteration.

We now define transfer functions and conversion primitives.

(1) *affine transformations:* We first introduce two functions $f \in \mathbb{R}^2 \to \overline{\mathbb{F}} \to \overline{\mathbb{F}}$ and $g \in \mathbb{R}^2 \to \overline{\mathbb{F}}$. The function f computes an affine transformation in floating-point arithmetics, that is to say that we define f by $f(\alpha, \beta)(x) = \lceil \lceil \lceil a \rceil \, x \rceil + \lceil b \rceil \rceil$. The function g gives an inductive limit to an affine transformation. It should satisfy $f(\alpha, \beta)(g(\alpha, \beta)) \leq g(\alpha, \beta)$. If there would be no rounding errors in the computation of f, the function g could be defined as $g(\alpha, \beta) = \left\lceil \frac{\beta}{\lfloor 1 - \alpha \rfloor} \right\rceil$ whenever $|\alpha| \leq 1$, and as $g(\alpha, \beta) = +\infty$ otherwise. Nevertheless, because of the rounding errors in the computation of f, we would have $f(\alpha, \beta)(g(\alpha, \beta)) \not\leq g(\alpha, \beta)$. So g would provide sound limits, but the analyzer would be unable to prove its soundness. Such a choice would lead to an error-prone implementation of the domain. So we propose to relax the definition of g, so that it provides a bound that the analyzer can check. For that purpose, we suppose that the difference between any computation step in floating-point arithmetics and the same

computation step in the real fields in always bounded by a maximal error which is an affine function of the result of the computation step in the real field (this property is satisfied in floating-point arithmetics whenever computation steps do not overflow [16]). As a consequence, we can provide two coefficients $\varepsilon_a > 0$ and $\varepsilon_M > 0$, such that for any representable number $x \in \mathbb{F}$, we have $\lceil \alpha x + \beta \rceil - (\alpha x + \beta) \leq \varepsilon_a(\alpha x + \beta) + \varepsilon_M$ and $(\alpha x + \beta) - \lfloor \alpha x + \beta \rfloor \leq \varepsilon_a(\alpha x + \beta) + \varepsilon_M$ (these coefficients are only defined by the size of the floating point number representation). Then, we can define $g(\alpha, \beta)$ as $\frac{\beta + \varepsilon_M}{1 - |\alpha| - \varepsilon_a}$ whenever $|\alpha| + \varepsilon_a < 1$, and as $g(\alpha, \beta) = +\infty$ otherwise.

We define the primitive $\text{AFF}_{\mathcal{F}^*_{q,r}}$ as follows:

$$\text{AFF}_{\mathcal{F}^*}[\alpha, \beta]((a, k), (b, l)) = (f(\alpha, \beta)(a), max\{f(\alpha, \beta)(b), g(\alpha, \beta)\}).$$

Thus if g provides sound bounds, the second domain jumps directly at an inductive bound. The transfer function is sound even if the function g is not. In the case when g is not sound, the second domain will no jump directly at the limits, so some extrapolation will be necessary to find an inductive invariants which will cause a loss of accuracy. Before the begin of the extrapolation process, the first domain is more precise which gives crucial information during the first iterates.

(2) *clock ticks:* Abstract properties do not depend on the clock, so we set $\text{TICK}_{\mathcal{F}^*}(a) = a$.

We now describe conversion primitives. We define the abstract element synthesis as $\text{C}_{\overline{\mathbb{F}} \to \mathcal{F}^*_{q,r}}(m) = ((m, 0), (m, 0))$. Then, given an abstract element, each component encodes a sound bound, so we define $\text{C}_{\mathcal{F}^*_{q,r} \to \overline{\mathbb{F}}}(((a, k), (b, l)), I) = min\{a, b\}$

We define the tuple $\mathcal{A}_{q,r}$ as:

$$(\mathcal{F}^*_{q,r}, \gamma_{\mathcal{F}^*_{q,r}}, \sqcup_{\mathcal{F}^*_{q,r}}, \sqcap_{\mathcal{F}^*_{q,r}}, \bot_{\mathcal{F}^*_{q,r}}, \nabla_{\mathcal{F}^*_{q,r}}, \triangle_{\mathcal{F}^*_{q,r}}, \text{AFF}_{\mathcal{F}^*_{q,r}}, \text{TICK}_{\mathcal{F}^*_{q,r}}, \text{C}_{\overline{\mathbb{F}} \to \mathcal{F}^*_{q,r}}, \text{C}_{\mathcal{F}^*_{q,r} \to \overline{\mathbb{F}}}).$$

Theorem 17 *The domain $\mathcal{A}_{q,r}$ is an iteration domain.*

6.3 The arithmetic-geometric progression domain

We introduce the domain $\mathcal{F}^*_d$ as the set of the 5-tuples $(M, a, b, a', b') \in (\overline{\mathbb{F}}^+)^5$. The domain $\mathcal{F}^*_d$ is ordered by the product order $\sqsubseteq_{\mathcal{F}^*_d}$. Intuitively, an element (M, a, b, a', b') of this domain encodes an arithmetic-geometric progression. The real M is a bound on the initial value of the progression. The affine transformation $[X \mapsto a'X + b']$ over-approximates the composition of all the affine transformations that can be applied to a value between two consecutive clock ticks. Finally, the affine transformation $[X \mapsto aX + b]$ over-approximates

the composition of all the affine transformations that have been applied to a value since the last clock tick. Thus, given a clock value $\mathtt{vc} \in \mathbb{N}$, we can define the concretization $\gamma^{\mathtt{vc}}_{\mathcal{F}^*_d}(d)$ of such a tuple $d = (M, a, b, a', b') \in \mathcal{F}^*_d$ by the set of all the elements $X \in \mathbb{R}$ such that $|X| \leq [X \mapsto aX + b]((\![X \mapsto a'X + b']\!)^{(\mathtt{vc})}(M))$. The element $\bot_{\mathcal{F}^*_d}$ is defined as $(0, 0, 0, 0, 0)$. The join operator $\sqcup_{\mathcal{F}^*_d}$ (resp. the widening operator $\triangledown_{\mathcal{F}^*_d}$, resp. the narrowing operator $\triangle_{\mathcal{F}^*_d}$ applies the maximum function (resp. the widening operator $\triangledown_{\overline{\mathbb{F}}}$, resp. the first projection) component wise.

We now define transfer functions and conversion primitives:

(1) The primitive $\textsc{aff}_{\mathcal{F}^*_d}$ applies an affine transformation with an element in $\mathcal{F}^*_d$. Let $d = (M, a, b, a', b')$ be an element in $\mathcal{F}^*_d$, let α and $\beta 0$ be two real coefficients. We define the element $\textsc{aff}_{\mathcal{F}^*_d}[\alpha, \beta](d) \in \mathcal{F}^*_d$ by $(M, \lceil a \times |\alpha| \rceil, \lceil b + |\beta| \rceil, a', b')$, when $\alpha \neq 0$, and by $(\lceil |beta| \rceil, 1, 0, 1, 0)$, otherwise.

(2) The primitive $\textsc{tick}_d \in \mathcal{F}^*_d \to \mathcal{F}^*_d$ simulates clock ticks. It maps any element $d = (M, a, b, a', b')$ into the element $(M, 1, 0, max(a, a'), max(b, b')) \in \mathcal{F}^*_d$. Thus, just after the clock tick, the arithmetic-geometric progression that has been applied since the last clock tick is the identity. The progression between two clock ticks is chosen by applying the worst case among the progression between the last two clock ticks, and the progression between any other two consecutive clock ticks.

(3) The arithmetic and geometric constraints synthesis is defined as follows:

$$\textsc{c}_{\overline{\mathbb{F}} \to \mathcal{F}^*_d}(m) = (M, 1, 0, 1, 0);$$

(4) The interval $\textsc{c}_{\mathcal{F}^*_d \to \overline{\mathbb{F}}}((M, a, b, a', b'), [m_{\mathtt{vc}}; M_{\mathtt{vc}}])$ is given by $[-l; l]$ where:

- $l = max(u_{m_{\mathtt{vc}}}, u_{M_{\mathtt{vc}}})$;
- $u_{\mathtt{vc}} = \lceil \lceil a \times v_{\mathtt{vc}} \rceil + b \rceil$;
- $v_{\mathtt{vc}} = \begin{cases} \lceil M + \lceil \mathtt{vc} \times b' \rceil \rceil & \text{if } a' = 1, \\ \left\lceil c_1^+ + c_2^+ \right\rceil & \text{otherwise;} \end{cases}$
- $\begin{cases} \exp_0^- = 1, \ \exp_{2\times n}^- = \lfloor \exp_n^- \times \exp_n^- \rfloor, \\ \exp_{2\times n+1}^- = \lfloor \lfloor \exp_n^- \times (\exp_n^-) \rfloor \times a' \rfloor; \end{cases}$
- $\begin{cases} \exp_0^+ = 1, \ \exp_{2\times n}^+ = \lceil \exp_n^+ \times \exp_n^+ \rceil, \\ \exp_{2\times n+1}^+ = \lceil \lceil \exp_n^+ \times \exp_n^+ \rceil \times a' \rceil; \end{cases}$
- $c_1^+ = \begin{cases} \left\lceil \exp_{\mathtt{vc}}^+ \times \left\lceil M - c_2^- \right\rceil \right\rceil & \text{if } M \geq c_2^- \\ \left\lceil \exp_{\mathtt{vc}}^- \times \left\lceil M - c_2^- \right\rceil \right\rceil & \text{otherwise;} \end{cases}$
- $c_2^- = \left\lfloor \frac{b'}{\lceil 1 - a' \rceil} \right\rfloor$ and $c_2^+ = \left\lceil \frac{b'}{\lfloor 1 - a' \rfloor} \right\rceil$.

Remark 18 *In the implementation, we use memoization to avoid computing the same exponential twice.*

We define the tuple $\mathcal{A}_{q,r}$ as:

$$(\mathcal{F}^*_{q,r}, \gamma_{\mathcal{F}^*_{q,r}}, \sqcup_{\mathcal{F}^*_{q,r}}, \sqcap_{\mathcal{F}^*_{q,r}}, \bot_{\mathcal{F}^*_{q,r}}, \triangledown_{\mathcal{F}^*_{q,r}}, \triangle_{\mathcal{F}^*_{q,r}}, \mathrm{AFF}_{\mathcal{F}^*_{q,r}}, \mathrm{TICK}_{\mathcal{F}^*_{q,r}}, \mathrm{C}_{\overline{\mathbb{F}}\to\mathcal{F}^*_{q,r}}, \mathrm{C}_{\mathcal{F}^*_{q,r}\to\overline{\mathbb{F}}}).$$

Theorem 19 *The domain $\mathcal{A}_{q,r}$ is an iteration domain.*

PROOF. See [10]. □

In the rest of the paper, we do not mention the extrapolation strategy parameters p and q anymore. So we denote by $\mathcal{F}$ our domain for abstracting bounds on real numbers (i.e. we use this domain to collect bounds on input streams) and by $\mathcal{F}^*$ our domain for abstracting the iterates of affine functions (either as a pair of real bounds, or as an arithmetic and geometric progression).

7 Simplified basic filters

Now, we propose a framework to build filter extension in the case of linear filters. First, we show how to build a history insensitive abstraction of the output stream of the first order filters and of the second order filters. Then, we show how we can refine a history insensitive abstraction in order to obtain a history sensitive one. Last, we show how the abstraction of first order filters and of second order filters can be used as basic blocks to build the abstraction of higher order filters.

To get the rough abstraction, we forget any historical information about the inputs during the filter iterations, so that each output is computed as an affine combination of the previous outputs. The constant term is an approximation of the contributions of both the previous inputs and the floating-point rounding errors.

7.1 Simplified high passband filter

A simplified high passband filter relates an input stream E_n to an output stream defined by $S_{n+1} = aS_n + E_n$.

Theorem 20 (Simplified first order filter (with rounding errors)) *Let $\varepsilon_a \geq 0$, $D \geq 0$, $m \geq 0$, a, X and Z be real numbers such that $|X| \leq D$ and $aX - (m + \varepsilon_a|X|) \leq Z \leq aX + (m + \varepsilon_a|X|)$. We have: $|Z| \leq (|a| + \varepsilon_a)D + m$.*

PROOF. We apply Thm. 1 by replacing m with $m + \varepsilon_a|X|$: we obtain that $|Z| \leq |a|D + (m + \varepsilon_a|X|)$. Then, we conclude that $|Z| \leq (|a| + \varepsilon_a)D + m$. □

We derive the following extension:

(1) structure and extrapolation primitives:
- a constraint relates the current output, the coefficient a of the filter, a relative error coefficient, and an abstract range: thus we set $\mathcal{T}_{r_1} = (\mathcal{V} \times \mathbb{F}^2)$;
- abstract ranges are abstracted in any domain for iterating linear functions: we set $(\mathcal{B}_{r_1}, \bot_{r_1}, \sqcup_{\mathcal{B}_{r_1}}, \triangledown_{\mathcal{B}_{r_1}}, \triangle_{\mathcal{B}_{r_1}}) = (\mathcal{F}^*, \bot_{\mathcal{F}^*}, \sqcup_{\mathcal{F}^*}, \triangledown_{\mathcal{F}^*}, \triangle_{\mathcal{F}^*})$;
- the concretization maps the abstract range to a set of values for the variable x_1 according to the value of the clock counter: a function $f \in \{x_1, \texttt{clock}\} \to \mathbb{R}$ is in the concretization $\gamma_{\mathcal{B}_{r_1}}((a, \varepsilon_a), e)$ if and only if there exists $(\texttt{vc}, m) \in \gamma_{\mathcal{F}^*}(e)$ such that $f(\texttt{clock}) = \texttt{vc}$ and $|f(x_1)| \leq m$;

(2) assignments:
- to compute transfer function, we need the filter coefficient, the relative error coefficient, and a bound on the current input: thus, we set $\mathit{info} = \mathbb{F}^2 \times \overline{\mathbb{F}}$
- $\textsc{rlvt}_{r_1}$ maps each expression E to the set of the tuples (X, a, ε_a), such that E matches $I \times X + E'$ with $I \subseteq [\frac{1}{2}; 1[$ and $I = [a - \varepsilon_a; a + \varepsilon_a]$;
- then we shift variables and collect a bound on the current input: we set $\textsc{pt}_{r_1}(Z = I \times X + E', (X, a, \varepsilon_a), \rho^\sharp) = ((Z, a, \varepsilon_a), (a, \varepsilon_a, m))$ where m is a bound on the absolute value of the expression E' in any environment in the concretization of $\rho^\sharp$;
- then we update the abstract range thanks to the domain $\mathcal{F}^*$: we define $\delta_{r_1}((a, \varepsilon_a, m), r)$ as $\textsc{aff}_{\mathcal{F}^*}\left[\lceil |a| + |\varepsilon_a| \rceil, m\right](r)$;

(3) clock ticks:
the abstraction of a clock tick is obtained by applying the clock tick on the abstract range: we set $\textsc{tick}_{\mathcal{B}_{r_1}} = \textsc{tick}_{\mathcal{F}^*}$;

(4) conversion primitives:
- the primitive $\textsc{build}_{r_1}$ just collect the abstraction of initial output value: we set $\textsc{build}_{r_1}((a, \varepsilon_a), f, \mathcal{R}) = \textsc{c}_{\mathbb{R} \to \mathcal{F}^*}(max\{\lceil -i \rceil, s\})$, where $f(x_1) = [-i, s]$.
- conversely, the primitive $\textsc{to}_{r_1}$ collects the range for the clock counter and extracts a bound for the output: we define $\textsc{to}_{r_1}((a, \varepsilon_a), e, I)$ as the function $[x_1 \mapsto [-\textsc{c}_{\mathcal{F}^* \to \mathbb{R}}(e, I); \textsc{c}_{\mathcal{F}^* \to \mathbb{R}}(e, I)]]$.

We define the tuple $\mathcal{G}_{r_1}$ as $(\mathcal{T}_{r_1}, \mathcal{B}_{r_1}, \gamma_{\mathcal{B}_{r_1}}, \sqcup_{\mathcal{B}_{r_1}}, \sqcap_{\mathcal{B}_{r_1}}, \bot_{\mathcal{B}_{r_1}}, \triangledown_{\mathcal{B}_{r_1}}, \triangle_{\mathcal{B}_{r_1}}, \textsc{rlvt}_{r_1}, \textsc{info}_{r_1}, \textsc{pt}_{r_1}, \delta_{r_1}, \textsc{tick}_{r_1}, \textsc{build}_{r_1}, \textsc{to}_{r_1})$.

Theorem 21 *The tuple $\mathcal{G}_{r_1}$ is a generic extension.*

A simplified second order filter relates an input stream E_n to an output stream defined by:

$$S_{n+2} = aS_{n+1} + bS_n + E_{n+2}.$$

Thus we experimentally observe, in Fig. 8, that starting with $S_0 = S_1 = 0$ and provided that the input stream is bounded, the pair (S_{n+2}, S_{n+1}) lies in an ellipse. Moreover, this ellipse is attractive, which means that an orbit starting out of this ellipse, will get closer of it. This behavior is explained by Thm. 22.

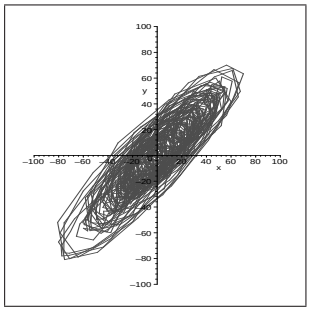


Fig. 8: Orbit.

Theorem 22 *Let a, b, $\varepsilon_a \geq 0$, $\varepsilon_b \geq 0$, $k \geq 0$, $m \geq 0$, X, Y be real numbers, such that $a^2+4b < 0$, and $X^2 - aXY - bY^2 \leq k^2$. Let Z be a real number such that: $aX + bY - (m + \varepsilon_a|X| + \varepsilon_b|Y|) \leq Z \leq aX + bY + (m + \varepsilon_a|X| + \varepsilon_b|Y|)$.*

Let δ be $2\frac{\varepsilon_b+\varepsilon_a\sqrt{-b}}{\sqrt{-(a^2+4b)}}$, we have:

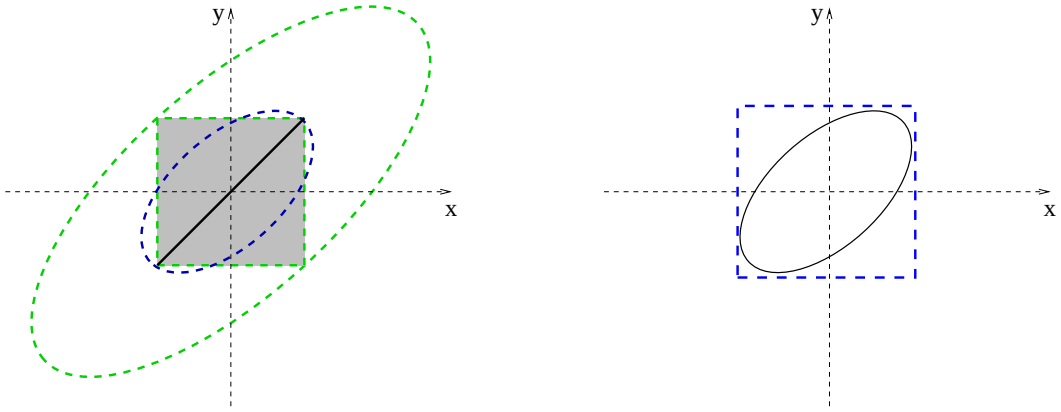
(1) $|X| \leq 2k\sqrt{\frac{b}{a^2+4b}}$ *and* $|Y| \leq 2k\sqrt{\frac{-1}{a^2+4b}}$;

(2) $Z^2 - aZX - bX^2 \leq \left((\sqrt{-b}+\delta)k + m\right)^2$;

PROOF.

(1) We have $4b < a^2$, so $b < 0$.
- We have $X^2 - aXY - bY^2 = -b\left(Y - \frac{a}{2b}X\right)^2 + \left(1 - \frac{a^2}{b}\right)X^2$. Since $b < 0$, $4b + a^2 < 0$, $k \geq 0$, and $X^2 - aXY - bY^2 \leq k^2$, we obtain that $|X| \leq 2k\sqrt{\frac{b}{a^2+4b}}$;
- We have $X^2 - aXY - bY^2 = \left(X - \frac{a}{2}Y\right)^2 - \left(\frac{4b+a^2}{4}\right)Y^2$. Since $4b+a^2 < 0$, $k \geq 0$, and $X^2 - aXY - bY^2 \leq k^2$, we obtain that $|X - \frac{a}{2}Y| \leq k$ and $|Y| \leq 2k\sqrt{\frac{-1}{a^2-4b}}$.

(2) We apply Thm. 3 by replacing m with $m + \varepsilon_a|X| + \varepsilon_b|Y|$. We obtain that $Z^2 - aZX - bX^2 \leq \left(\sqrt{-b}k + (m + \varepsilon_a|X| + \varepsilon_n|Y|)\right)^2$. Moreover, since $\sqrt{-b}k + (m + \varepsilon_a|X| + \varepsilon_n|Y|) \geq 0$, $\varepsilon_a \geq 0$, and $\varepsilon_b \geq 0$, we conclude that $Z^2 - aZX - bX^2 \leq \left(\sqrt{-b}k + \left(m + \varepsilon_a 2k\sqrt{\frac{b}{a^2+4b}} + \varepsilon_b 2k\sqrt{\frac{-1}{a^2-4b}}\right)\right)^2$. Then,



(a) Abstracting the underlying constraints into an ellipse.

(b) Abstracting an ellipse into interval ranges.

Fig. 9. Reduction between abstract domains.

$$Z^2 - aZX - bX^2 \leq \left(\left(\sqrt{-b} + 2\frac{\varepsilon_b + \varepsilon_a\sqrt{-b}}{\sqrt{-(a^2+4b)}}\right)k + m\right)^2. \qquad \square$$

Remark 23 *Because of rounding errors, some filters that would be stable when computed in real arithmetics may diverge when computing in floating points arithmetics. Nevertheless, we can prove that a second order filter converge when* $\sqrt{-b} + \delta < 1$ *(where* $\delta = 2\frac{\varepsilon_b + \varepsilon_a\sqrt{-b}}{\sqrt{-(a^2+4b)}}$*).*

We derive the following abstract extension:

(1) structure and extrapolation operator:
- a constraint relates the last two outputs, the coefficients a and b of the filter, relative error coefficients, and an abstract range: thus we set $\mathcal{T}_{r_2} = (\mathcal{V}^2 \times \mathbb{F}^4)$;
- abstract ranges are abstracted in any domain for iterating linear functions: we set $(\mathcal{B}_{r_2}, \bot_{r_2}, \sqcup_{\mathcal{B}_{r_2}}, \triangledown_{\mathcal{B}_{r_2}}, \triangle_{\mathcal{B}_{r_2}}) = (\mathcal{F}^*, \bot_{\mathcal{F}^*}, \sqcup_{\mathcal{F}^*}, \triangledown_{\mathcal{F}^*}, \triangle_{\mathcal{F}^*})$;
- the abstract range encodes the ratio of an ellipse: this way, the concretization $\gamma_{\mathcal{B}_{r_2}}((a, \varepsilon_a, b, \varepsilon_b), e)$ is given by the set of maps f for which there exists a pair $(\texttt{vc}, m) \in \gamma_{\mathcal{F}^*}(e)$ such that: $f(\texttt{clock}) = \texttt{vc}$ and $(f(x_1))^2 - af(x_1)f(x_2) - b(f(x_2))^2 \leq m$.

(2) assignments:
- to compute transfer functions, we need the filter coefficients, the relative error coefficients, and a bound on the current input: thus, we set $\mathit{info} = \mathbb{F}^4 \times \overline{\mathbb{F}}$;
- $\textsc{rlvt}_{r_2}$ maps each expression E to the set of the tuples $(X, Y, a, \varepsilon_a, b, \varepsilon_b)$, such that E matches $I_x \times X + I_y \times Y + E'$ with $I_x = [a - \varepsilon_a; a + \varepsilon_a]$, $I_y = [b - \varepsilon_b; b + \varepsilon_b]$, and $a^2 + 4b < 0$;
- then we shift variables and collect a bound on the current input: we define the pair $\textsc{pt}_{r_2}(Z = I_x \times X + I_y \times Y + E', (X, Y, a, \varepsilon_a, b, \varepsilon_b), \rho^\sharp)$ as

$((Z, Y, a, \varepsilon_a, b, \varepsilon_b), (a, \varepsilon_a, b, \varepsilon_b, m))$ where m is a bound on the absolute value of the expression E' in the abstract environment $\rho^\sharp$ (i.e. in any environment in the concretization of $\rho^\sharp$);

- then we update the abstract range thanks to the domain $\mathcal{F}^*$: we define $\delta_{r_2}((a, \varepsilon_a, b, \varepsilon_b, m), r)$ as $\text{AFF}_{\mathcal{F}^*}\left[\left[\left\lceil\sqrt{-b}\right\rceil + 2\left\lceil\left\lceil\frac{\left\lceil\varepsilon_b + \left\lceil\varepsilon_a\left\lceil\sqrt{-b}\right\rceil\right\rceil\right\rceil}{\left\lceil\sqrt{-\lceil\lceil a^2\rceil + \lceil 4b\rceil\rceil}\right\rceil}\right\rceil\right\rceil\right\rceil, m\right](r)$;

(3) clock ticks:
the abstraction of a clock tick is obtained by applying the clock tick on the abstract range: we set $\text{TICK}_{\mathcal{B}_{r_2}} = \text{TICK}_{\mathcal{F}^*}$;

(4) conversion primitives:

- to build an abstraction of the initialization state, we distinguish two cases in accordance with the fact that initial outputs are equal, or not; this is illustrated in Fig. 9(a) where we obtain two ellipses depending the fact that we know whether x and y are equal, or not; thus we define the abstract range $\text{BUILD}_{r_2}((a, \varepsilon_a, b, \varepsilon_b), f, \mathcal{R})$ as $\text{C}_{\mathbb{R}\to\mathcal{F}^*}(\lceil\sqrt{m}\rceil)$,
where $m = \begin{cases} \left\lceil\sqrt{\lceil\lceil 1 - \lfloor a + b\rfloor\rceil\,\lceil x^2\rceil\rceil}\right\rceil & \begin{cases}\text{if } (x_1, x_2) \in \mathcal{R}\\ \text{and } 1 - a - b \geq 0;\end{cases}\\ \left\lceil\sqrt{\lceil\lceil\lceil a + b\rceil - 1\rceil\,\lceil x^2\rceil\rceil}\right\rceil & \begin{cases}\text{if } (x_1, x_2) \in \mathcal{R}\\ \text{and } a + b - 1 > 0;\end{cases}\\ \lceil\lceil\lceil x^2\rceil + \lceil\lceil |a|x\rceil\, y\rceil\rceil + \lceil |b|\,\lceil y^2\rceil\rceil\rceil & \text{otherwise}\end{cases}$
and $x = max\{\lceil -i_x\rceil, \lceil s_x\rceil\}$, $x = f(x_1)$, $y = max\{\lceil -i_y\rceil, \lceil s_y\rceil\}$, $y = f(x_2)$;
- an ellipse can then be enclosed inside a rectangle (e.g. see Fig. 9(b)); thus, if $\lceil\lceil a^2\rceil + \lceil 4b\rceil\rceil < 0$, we define $\text{TO}_{r_2}((a, \varepsilon_a, b, \varepsilon_b), e, I)$ as $[x_1 \mapsto [-m_1; m_1], x_2 \mapsto [-m_2, m_2]]$,
where $\begin{cases} m_1 = \text{C}_{\mathbb{R}\to\mathcal{F}^*}\left(\left\lceil\lceil 2\text{C}_{\mathcal{F}^*\to\mathbb{R}}(e, I)\rceil\left\lceil\sqrt{\frac{-b}{-\lceil\lceil a^2\rceil + \lceil 4b\rceil\rceil}}\right\rceil\right\rceil\right),\\ m_2 = \text{C}_{\mathbb{R}\to\mathcal{F}^*}\left(\left\lceil\lceil 2\text{C}_{\mathcal{F}^*\to\mathbb{R}}(e, I)\rceil\left\lceil\sqrt{\frac{1}{\lfloor -\lceil\lceil a^2\rceil + \lceil 4b\rceil\rceil\rfloor}}\right\rceil\right\rceil\right);\end{cases}$
otherwise, we set $\text{TO}_{r_2}((a, \varepsilon_a, b, \varepsilon_b), e, I) = [x_1, x_2 \mapsto \mathbb{R}]$.

We define the tuple $\mathcal{G}_{r_2}$ as $(\mathcal{T}_{r_2}, \mathcal{B}_{r_2}, \gamma_{\mathcal{B}_{r_2}}, \sqcup_{\mathcal{B}_{r_2}}, \sqcap_{\mathcal{B}_{r_2}}, \bot_{\mathcal{B}_{r_2}}, \triangledown_{\mathcal{B}_{r_2}}, \vartriangle_{\mathcal{B}_{r_2}}, \text{RLVT}_{r_2}, \text{INFO}_{r_2}, \text{PT}_{r_2}, \delta_{r_2}, \text{TICK}_{r_2}, \text{BUILD}_{r_2}, \text{TO}_{r_2})$.

Theorem 24 *The tuple $\mathcal{G}_{r_2}$ is a generic extension.*

8 Formal expansion

In this section, we propose a framework to refine the history insensitive abstractions of filters. Thus we propose abstractions that take care of the can-

cellation effects between the inputs involved in the computation of several consecutive outputs. To achieve this goal, we split each output value into three summands: the contribution of the last N inputs if the digital filter were computed in the real field, the contribution of the initial outputs and of the other inputs if the digital filter were computed in the real field, and the difference between the result of the computation in the floating-point world and the result of the computation in the real field. The integer N is a parameter of the approximation. Then, the first summand can be symbolically computed as a known function from the last input values into the real field. The second and the third summands both satisfy simplified recursions, so they can be analyzed by using history insensitive abstractions (we recall that the abstract domains for first and second order simplified filters are given in Sect. 7, abstract domains for simplified higher order filters are given in Sect. 9).

More formally, our goal is to bound the iterates of a sequence of real numbers defined as: $S_n = i_n$, for any n such that $0 \leq n < p$, and $S_{n+p} = F(S_n, \ldots, S_{n+p-1}) + G(E_{n+p+1-q}, \ldots, E_{n+p}) + \epsilon_{n+p}$. Thus, in this section, we gather all rounding errors in the computation of the value of an expression into one single value. In this definition, p is the number of consecutive inputs which are involved in the computation of the next output (i.e. the order of the filter), q is the number of successive inputs which are involved in the computation of the next input, the sequence $(E_n)_{p-q<n}$ is the input stream, the sequence $(S_n)_{n\geq 0}$ is the output stream, and the sequence $(\varepsilon_n)_{n\geq p}$ denotes the sequences of the rounding errors that are performed at each iteration of the filter. The tuple $(i_n)_{0\leq n<p}$ of real numbers denotes the initial values of the output. The function F and G are linear (i.e. that is to say that $H(\lambda\overline{x}) = \lambda H(\overline{x})$ and that $H(\overline{x}\overline{+}\overline{y}) = H(\overline{x}) + H(\overline{y})$, for any $H \in \{F; G\}$, any tuple $\overline{x}$,$\overline{y}$ of real numbers (of correct length), and any real number λ).

8.1 Bounding the contribution of rounding errors

The first step consists in isolating the contribution of rounding errors. We define the sequence (R_n) as $R_n = i_n$, for any n such that $0 \leq n < p$, and $R_{n+p} = F(R_n, \ldots, R_{n+p-1}) + G(E_{n+p+1-q}, \ldots, E_{n+p})$. The sequence (R_n) denotes the ideal behavior of the filter (i.e. when it were computed in the real field). Then we introduce the sequence (ε_n) which is defined as $\varepsilon_n = S_n - R_n$, for any $n \in \mathbb{N}$. The sequence (ε_n) denotes the overall contribution of rounding errors in the n-th output.

Proposition 25 *The sequence (ε_n) satisfies: $\varepsilon_n = 0$ for any integer $n \in \mathbb{N}$ such that $0 \leq n < p$, and $\varepsilon_{n+p} = F(\varepsilon_n, \ldots, \varepsilon_{n+p-1}) + \epsilon_{n+p}$ for any integer $n \in \mathbb{N}$.*

PROOF. We introduce the sequence ε'_n of real numbers that is defined as: $\varepsilon'_n = 0$ for any integer $n \in \mathbb{N}$ such that $0 \leq n < p$, and $\varepsilon'_{n+p} = F(\varepsilon'_n, \ldots, \varepsilon'_{n+p-1}) + \epsilon_{n+p}$ for any integer $n \in \mathbb{N}$. We prove by induction over n that $\varepsilon_n = \varepsilon'_n$ for any $n \in \mathbb{N}$.

(1) for any $n < p$, we have $\varepsilon_n = S_n - R_n = i_n - i_n = 0 = \varepsilon'_n$;
(2) we suppose that there exists $n_0 \in \mathbb{N}$ such that $n_0 \geq p$ and such that for any integer $n \in \mathbb{N}$ that satisfies $n < n_0$ we have $\varepsilon_n = \varepsilon'_n$. By definition, we have $\varepsilon_{n_0} = S_{n_0} - R_{n_0}$. So $\varepsilon_{n_0} = F(S_{n_0-p}, \ldots, S_{n_0-1}) + G(E_{n_0+1-q}, \ldots, E_{n_0}) + \epsilon_{n_0} - F(R_{n_0-p}, \ldots, R_{n_0-1}) - G(E_{n_0+1-q}, \ldots, E_{n_0})$. Then, $\varepsilon_{n_0} = F(S_{n_0-p}, \ldots, S_{n_0-1}) - F(R_{n_0-p}, \ldots, R_{n_0-1}) + \epsilon_{n_0}$. Since F is linear, we obtain that $\varepsilon_{n_0} = F(S_{n_0-p} - R_{n_0-p}, \ldots, S_{n_0-1} - R_{n_0-1}) + \epsilon_{n_0}$. By definition of (ε_n), we have $\varepsilon_{n_0} = F(\varepsilon_{n_0-p}, \ldots, \varepsilon_{n_0-1}) + \epsilon_{n_0}$. By induction hypotheses, we conclude that $\varepsilon_{n_0} = F(\varepsilon'_{n_0-p}, \ldots, \varepsilon'_{n_0-1}) + \epsilon_{n_0}$. So, by definition of (ε'_n), we conclude that $\varepsilon_{n_0} = \varepsilon'_{n_0}$.

So for any integer $n \in \mathbb{N}$, we have $\varepsilon_n = \varepsilon'_n$. □

Thanks to Prop. 25, we can bound the contribution of the rounding errors by using the corresponding simplified filter domain.

8.2 Expanding the output of the ideal filter

To refine the output, we need to bound the sequence R_n. For that purpose, we express each output R_n as a linear combination of the previous inputs and of the initial outputs:

Definition 26 (Expansion coefficients) *We define the family $(c_k^n)_{(n,k)}$ of real numbers indexed by the pairs $(n,k) \in \mathbb{N}^2$ of integers such that $n \in \mathbb{N}$ and $k \geq p+1-q$ as follows*[9] *:*

$$\begin{cases} c_k^n = 0 & \text{when } n < p \text{ or } k > n, \\ c_k^n = F(c_k^{n-p}, \ldots, c_k^{n-1}) + G(\delta_k^{n-q+1}, \ldots, \delta_k^n) & \text{otherwise;} \end{cases}$$

and the family $(d_k^n)_{(n,k)}$ of real numbers indexed by the pairs $(n,k) \in \mathbb{N}^2$ of integers that satisfies $k < p$ as follows:

$$\begin{cases} d_k^n = \delta_k^n & \text{when } n < p, \\ d_k^n = F(d_k^{n-p}, \ldots, d_k^{n-1}) & \text{otherwise.} \end{cases}$$

[9] The symbol δ denotes the Kronecker function, i.e. δ_i^j is equal to 1 if $i = j$ and δ_i^j is equal to 0 otherwise.

Proposition 27 (Formal expansion) *For any integer $n \in \mathbb{N}$, the n-th output of the ideal filter satisfies: $R_n = \sum_{k=0}^{p-1} d_k^n i_k + \sum_{k=p+1-q}^{n} c_k^n E_k$.*

PROOF. By induction over the integer $n \in \mathbb{N}$. □

Roughly speaking, the coefficient c_k^n denotes the overall contribution of the k-th input in the value of the n-th output whereas the coefficient d_k^n denotes the overall contribution of the k-th initial output in the value of the n-th output.

Proposition 28 *For any pair (k, l) of integers such that $p + 1 - q \leq k \leq p$ and $l \geq 0$, we have $c_k^p = c_{k+l}^{p+l}$; for any pair (k, l) of integers such that $k \geq 0$ and $l \geq 0$, we have $c_{p+1-q}^{p+k} = c_{p+1-q+l}^{p+k+l}$.*

PROOF. We prove that $c_k^p = c_{k+l}^{p+l}$ by induction first over the integer $p - k \in \mathbb{N}$, then over the integer $l \in \mathbb{N}$. We prove that $c_{p+1-q}^{p+k} = c_{p+1-q+l}^{p+k+l}$ by induction first over the integer $p + k \in \mathbb{N}$, then over the integer $l \in \mathbb{N}$. □

8.3 Bounding the contribution of the last inputs

To compute an accurate bound on the value of the sequence (R_n), we propose to isolate the contribution of the last inputs. First, we fix an integer parameter $N \in \mathbb{N}$ such that both $N \geq p$ and $N \geq q$. Then, we define the contribution $\mathit{last}(N, n)$ of last N inputs in the n-th output as $\sum_{k=n+1-N}^{n} c_k^n E_k$ and the contribution $\mathit{tail}(N, n)$ of both the other inputs and the initial inputs as $R_n - \mathit{last}(N, n)$.

Proposition 29 *If $n > N + p$, we have $\mathit{last}(N, n) = \sum_{k=p+1-N}^{p} c_k^p E_{k+n-p}$.*

PROOF. We suppose that $n > N + p$. For any integer k such that $k \geq n + 1 - N$, thanks to Prop. 28, since $n \geq p$, we have $c_k^n = c_{k-n+p}^p$. So $\mathit{last}(N, n) = \sum_{k=n+1-N}^{n} c_{k-n+p}^p E_k$. Then we set $k' = k - n + p$ and we conclude that $\mathit{last}(N, n) = \sum_{k'=p+1-N}^{p} c_{k'}^p E_{k'+n-p}$. □

In practice, our abstraction collects a bound m on the input stream (i.e. such that $|E_n| \leq m$ for any integer $n \geq p + 1 - q$). Then, it can deduce that $|\mathit{last}(N, n)| \leq m \sum_{k=0}^{N-1} |c_{p-k}^p|$. This way, we can compute a bound on the contribution $\mathit{last}(N, n)$ of the last inputs that does not depend on the integer n.

8.4 Bounding the tail

We are left to bound the value of the expression $|R_n - last(N,n)|$. For any integer $n \in \mathbb{N}$, we denote $tail(N,n) = R_n - last(N,n)$.

Proposition 30 *Whenever $n \geq 2p$ (we recall that we have supposed that $N \geq q$), we have $tail(N,n) = A + B$, where:*

- $A = F(tail(N,n-p), \ldots, tail(N,n-1))$,
- $B = F(\sum_{k=0}^{p-1} c_{2p-N-k}^{p} E_{n-N-k}, \ldots, \sum_{k=0}^{p-p} c_{p+1-N-k}^{p} E_{n-N-k})$.

PROOF. Let $n \in \mathbb{N}$ such that $N \geq q$ and $n \geq 2p$. By definition, we have $tail(N,n) = \sum_{k=0}^{p-1} d_k^n i_k + \sum_{k=p+1-q}^{n-N} c_k^n E_k$. For any k such that $p+1-q \leq k \leq n-N$, we have $k < n-q+1$. Moreover, we have $n \geq p$. So by Def. 26, we have $tail(N,n) = \sum_{k=0}^{p-1} F(d_k^{n-p}, \ldots, d_k^{n-1}) i_k + \sum_{k=p+1-q}^{n-N} F(c_k^{n-p}, \ldots, c_k^{n-1}) E_k$. By linearity of F, we get that $tail(N,n) = F(\sum_{k=0}^{p-1} d_k^{n-p} i_k + \sum_{k=p+1-q}^{n-N} c_k^{n-p} E_k, \ldots,$ $\sum_{k=0}^{p-1} d_k^{n-1} i_k + \sum_{k=p+1-q}^{n-N} c_k^{n-1} E_k)$. Then we deduce that $tail(N,n)$ is equal to $F(tail(N,n-p) + \sum_{k=n-p-N+1}^{n-N} c_k^{n-p} E_k, \ldots, tail(N,n-1) + \sum_{k=n-N}^{n-N} c_k^{n-1} E_k)$. By linearity, we get that $tail(N,n)$ is equal to $F(tail(N,n-p), \ldots, tail(N,n-1)) + F(\sum_{k=n-p-N+1}^{n-N} c_k^{n-p} E_k, \ldots, \sum_{k=n-1-N+1}^{n-N} c_k^{n-1} E_k)$. Then, by setting $k' = n-N-k$, we obtain that $tail(N,n) = F(tail(N,n-p), \ldots, tail(N,n-1)) + F(\sum_{k'=0}^{p-1} c_{n-N-k'}^{n-p} E_{n-N-k'}, \ldots, \sum_{k'=0}^{p-p} c_{n-N-k'}^{n-1} E_{n-N-k'})$. By Prop. 28 and since $n-p \geq p$, we get that $tail(N,n) = F(tail(N,n-p), \ldots, tail(N,n-1)) + F(\sum_{k=0}^{p-1} c_{2p-N-k}^{p} E_{n-N-k}, \ldots, \sum_{k=0}^{p-p} c_{p+1-N-k}^{p} E_{n-N-k})$. □

This way, to compute a bound on $tail(N,n)$, we only require an abstraction of the input stream and a filter domain for analyzing the corresponding simplified filter.

8.5 Summary

The result that are required for bounding the output of filters are summarized in Thm. 31.

Theorem 31 *We suppose that N is bigger than both p and q. For any integer $n \in \mathbb{N}$ such that $n > N+p$, we have $S_n = last(N,n) + tail(N,n) + \varepsilon_n$. Moreover, we have:*

(1) for any integer $n \in \mathbb{N}$ such that $0 \leq n < p$, we have $S_n = i_k$;
(2) for any integer $n \in \mathbb{N}$ such that $p \leq n \leq N+p$, we have $S_n = \sum_{k=0}^{p-1} d_k^n i_k + \sum_{k=p+1-q}^{n} c_k^n E_k$;

(3) *the sequence* (ε_n) *satisfies:* $\varepsilon_n = 0$ *for any* $k \in \mathbb{N}$ *such that* $0 \leq k < p$, *and* $\varepsilon_{n+p} = F(\varepsilon_n, \ldots, \varepsilon_{n+p-1}) + \epsilon_{n+p}$;
(4) *for any integer* $n \in \mathbb{N}$ *such that* $n > N + p$, *we have* $\mathit{last}(N, n) = \sum_{k=p+1-N}^{p} c_k^p E_{k+n-p}$;
(5) *for any integer* $n \in \mathbb{N}$, *if* $n > N + p$ *and* $N \geq q$, *then* $\mathit{tail}(N, n) = A + B$, *where* $\begin{cases} A = F(\mathit{tail}(N, n-p), \ldots, \mathit{tail}(N, n-1)), \\ B = F(\sum_{k=0}^{p-1} c_{2p-N-k}^p E_{n-N-k}, \ldots, \sum_{k=0}^{p-p} c_{p+1-N-k}^p E_{n-N-k}). \end{cases}$
Moreover, for any integer $n \in \mathbb{N}$ *such that* $p \leq n < 2p$, *we know that* $\mathit{tail}(N, n) = \sum_{k=0}^{p-1} d_k^n i_k + \sum_{k=p+1-q}^{n-N} c_k^n E_k$.

8.6 Computing the coefficients of the formal expansion

Then to build an abstract domain in order to analyze our filters, we have to compute the coefficients that are involved in Thm. 31. Unfortunately, they are all real parameters and we cannot afford computation in the real field. To solve this problem, we propose to compute them by using floating point interval arithmetics. After each atomic step, we use the functions $\lceil \_ \rceil$ and $\lfloor \_ \rfloor$ to round results toward the correct direction.

If the coefficients were computed in the real field, the abstract gain (i.e. the ratio between the range of the output stream found by the analyzer and the range of the input stream) would converge exponentially to the concrete gain (i.e. the behavior of the filter in concrete computations) when N increases toward $+\infty$. But, because of the rounding errors in the computation of the parameters, first the accuracy increases until we cannot decide the sign of the coefficients anymore. Then it decreases exponentially. In Fig. 10, we give the abstract gain that we found when analyzing the example given in Exa. 5 with respect to the parameter N. The first graph describes the whole evolution of the abstract gain, whereas the second one focuses arround the best choice for the parameter N.

In the ASTRÉE analyzer, filters are not all expanded with the same parameter N. For each filter, we choose the parameter N as the last integer such that the sign of the coefficients is known. Moreover, since the computation of the coefficients is quite expensive, we use memoisation to avoid computing these coefficients twice.

We now suppose that we have a sound bound on each coefficient. This way, we suppose that we have some coefficients $\mathit{tail\_input}^\sharp(F, G)$, $\mathit{tail\_init}_\emptyset^\sharp(F, G)$, $\mathit{tail\_init}_=^\sharp(F, G)$, $\mathit{last}^\sharp(F, G)$, $\mathit{first\_outputs}_\emptyset^\sharp(F, G)$, and $\mathit{first\_outputs}_=^\sharp(F, G)$ such that:

(1) the coefficient $\mathit{tail\_input}^\sharp(F, G)$ is a bound on the value of expression

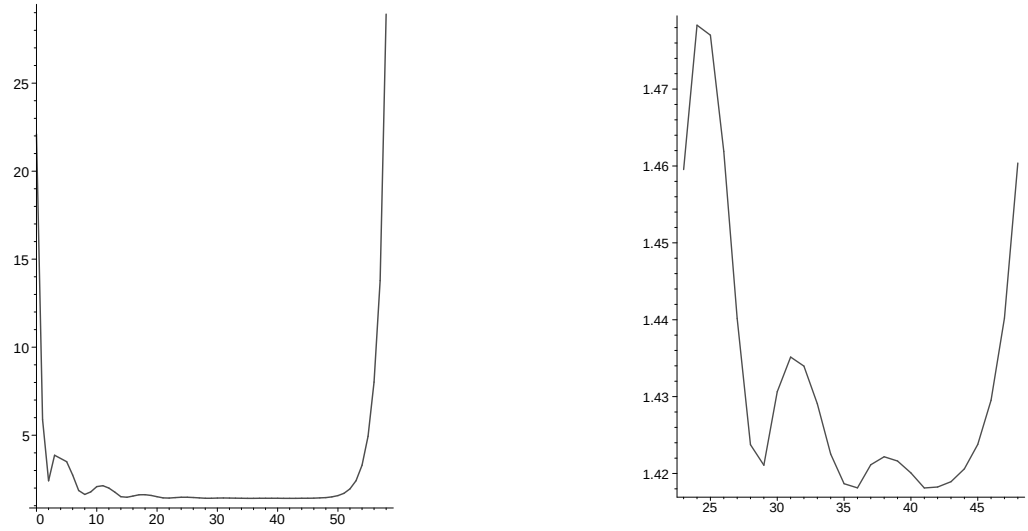


Fig. 10. Abstract gain with respect to the parameter N.

$|F(\sum_{k=0}^{p-1} c_{2p-N-k}^{p} x_k, \ldots, \sum_{k=0}^{p-p} c_{p+1-N-k}^{p} x_k)|$ for any tuple $(x_i) \in \{-1; 1\}^p$; it encodes a correcting multiplicative coefficient to be applied to the bound on the input stream when we compute the contribution of the inputs that are not exactly described in the formal expansion;

(2) the coefficient $\mathit{tail\_init}_0^\sharp(F, G)$ is a bound on the value of $\sum_{k=0}^{n-N} |c_k^n| + \sum_{k=0}^{p-1} |d_k^n|$ for any integer n between N and $N + p$; this coefficient is used to collect the initial abstraction of the tail of the filter (i.e. the contribution of both the first $n - N + 1$ inputs and of the initial outputs in the case when we have no equality relationship among the initial inputs and among the initial outputs);

(3) the coefficient $\mathit{tail\_init}_1^\sharp(F, G)$ is a bound on the value of $|\sum_{k=0}^{q-2} c_k^n + \sum_{k=0}^{p-1} d_k^n| + \sum_{k=q-1}^{n-N} |c_k^n|$ for any integer n between N and $N + p$; this coefficient is used to collect the initial abstraction of the tail of the filter (i.e. the contribution of the first $n - N + 1$ inputs and of the initial outputs in the case when the first $q - 1$ inputs and the first p outputs are all equal);

(4) the coefficient $\mathit{last}^\sharp(F, G)$ is a bound on the value of $\sum_{k=p+1-N}^{p} |c_k^p|$; it is a multiplicative coefficient to be applied on the bound on the input stream to get an over-approximation of the contribution of the last N inputs;

(5) the coefficient $\mathit{first\_outputs}_0^\sharp(F, G)$ is a bound on the value of expression $max\{1; \sum_{k=0}^{p-1} |d_k^n| + \sum_{k=0}^{n} |c_k^n|\}$ for any integer n between 0 and $N + p$; this is a multiplicative coefficient to get a bound on the first $N + p$ outputs with which the formal expansion cannot be applied yet; this coefficient is used when we have no equality relationship about the initial outputs and the initial inputs;

(6) the coefficient $\mathit{first\_outputs}_1^\sharp(F, G)$ is a bound on the value of expression $max\{1, |\sum_{k=0}^{p-1} d_k^n + \sum_{k=0}^{q-2} c_k^n| + \sum_{k=q-1}^{n} |c_k^n|\}$ for any integer n between 0 and $N + p$; this is a multiplicative coefficient to get on bound on the first $N + p$ outputs in the case when we know that the first p outputs and the

first $q-1$ inputs are equal.

8.7 Abstract extension

We derive the following abstract extension:

(1) structure and extrapolation operators:
- a constraint relates the last p outputs, the last $q-1$ inputs, and the $p+q$ parameters; thus we set $\mathcal{T}_{p,q} = (\mathcal{V}^{p+q-1} \times \mathbb{F}^{p+q})$;
- for each tuple of variables and of parameters, we collect the fact that the filter is well-initialized (i.e. the value of the initial output and of the initial outputs are equal), a bound on the input stream, an abstract range for the contribution of rounding errors, and an abstract range for the tail of the output stream; thus we define $\mathcal{B}_{p,q}$ as $(\{0,1\} \times \mathcal{F} \times \mathcal{F}^* \times \mathcal{F}^*)$;
- the cup operator and the extrapolation operators are defined componentwise (for booleans, the cup operator is *max* whereas the extrapolation operators are given by the tuple $(1, max, min)$);
- $\gamma_{\mathcal{B}_{p,q}}((a_1, \ldots, a_p, b_1, \ldots, b_q), (\text{B}, m, k_\varepsilon, k_{tail}))$ is the set of functions f such that there exists:
  (a) an integer $n \in \mathbb{N}$, two finite sequences $(R_k)_{0 \le k \le n}$ and $(E_k)_{p-q+1 \le k \le n}$,
  (b) a tuple $(\varepsilon_i)_{1 \le i \le p}$,
  such that:
  - for any integer k such that $0 \le k < p$, we have $|R_k| \le fst(m)$;
  - for any integer k such that $p-q \le k \le n$, we have $|E_k| \le fst(m)$;
  - for any integer k such that $p \le k$, we have $R_k = F(R_{k-p}, \ldots, R_{k-1}) + G(E_{k-q+1}, \ldots, E_k)$;
  - if B, then $X = Y$ for any $X, Y \in \{R_k \mid 0 \le k < p\} \cup \{E_k \mid p-q-1 \le k \le p\}$;
  - $|f(x_i) - R_{n+1-i}| \le \varepsilon_i$ for any integer i such that $1 \le i \le p$;
  - $f(x_{i+p}) = E_{n-i+1}$ for any integer i such that $1 \le i < q$;
  - $[x_i \mapsto \varepsilon_i, 1 \le i \le p] \in \gamma_{\mathcal{B}_{r_p}}(k_\varepsilon)$;
  - $[x_i \mapsto R_{n-i+1} - \sum_{k=0}^{N-1} c_{n-k-i+1}^{n-i+1} E_{n-k-i+1}, 1 \le i \le p] \in \gamma_{\mathcal{B}_{r_p}}(k_{tail})$;

  roughly speaking, (E_n) encodes the input stream, (R_n) the ideal output stream, and (ε_n) the stream of the overall contribution of rounding errors; first we assume that m is bound on both the initial outputs and the initial inputs; then, we state that (R_n) is the ideal output stream obtained by iterating the filter on the input stream (E_n); the boolean B abstract the initialization condition; then, we state that up to rounding errors, x_i, for $1 \le i \le p$, is the $n-i+1$-th output, and, for any $1 \le i \le q-1$, $x_i + p$ is the $n-i+1$-th input; last, we states that the last p contribution of rounding errors are related by the abstract range k_ε and that the last p value of the tail of output stream are related by

the abstract range k_{tail}; this way, the concretization contains enough information to make the induction;

(2) assignments:
- $info = \mathbb{F}^{p+q} \times \overline{\mathbb{F}} \times \overline{\mathbb{F}}$ (the first coefficient in $\overline{\mathbb{F}}$ denotes a bound on the current input and the second coefficient encodes a bound on the sum of any atomic rounding errors in the expression that is computed to iterate the filter);
- the primitive $\text{RLVT}_{p,q}$ is defined by pattern matching the expression in order to detect whether it corresponds to a filter iteration, or not; when a filter is iterated, the primitive $\text{PT}_{p,q}$ shifts the variables and collects a bound M on the current inputs and a bound ϵ on the sum of each atomic rounding error in the computation of the expression;
- $\delta_{p,q}(((a_i),(b_j),M,\epsilon),(b,m,k_\varepsilon,k_{tail})) = (b',m',k'_\varepsilon,k'_{tail})$, where:
  - $F(X_1,\ldots,X_p) = \sum a_k X_k$ and $G(X_1,\ldots,X_q) = \sum b_k X_k$;
  - $b' = b$;
  - $m' = m \sqcup_{\mathcal{F}} (M,0)$;
  - $k'_\varepsilon = \delta_{r_p}((a_1,0,\ldots,a_p,0,\epsilon),k_\varepsilon)$,
  - $k'_{tail} = \delta_{r_p}((a_1,0,\ldots,a_p,0,fst(m') \times (tail\_input^\sharp(F,G))),k_{tail})$;

  this way, we keep the abstraction about the initialization of the filter, then we compute a new bound on the input stream (we take into account the new encountered input values), we update the abstract range for the contribution of rounding errors, and we we update the abstract range for the tail (to do this we apply the corrective coefficient).

(3) clock tick: to simulate ticks of clock, we just update the abstract ranges of the contribution of rounding errors and of the output stream tail; thus, we define $\text{TICK}_{\mathcal{B}_{p,q}}(b,m,k_\varepsilon,k_{tail})$ as $(b,m,\text{TICK}_{\mathcal{B}_{r_p}}(k_\varepsilon),\text{TICK}_{\mathcal{B}_{r_p}}(k_{tail}))$;

(4) conversion primitives:
- $\text{BUILD}_{p,q}((a_i)_{1\le i\le p+q},f_I,\mathcal{R})$ is given by $(b,m,k_\varepsilon,k_{tail})$ where:
  - $F(X_1,\ldots,X_p) = \sum a_k X_k$ and $G(X_1,\ldots,X_q) = \sum a_{k+p} X_k$;
  - $b = 1$ if and only if $(x_i,x_j) \in \mathcal{R}$ for any pair of integers such that $1 \le i < j < p+q-1$; this way, b encodes the fact that the filter is initialized with equal initial values;
  - m is a bound in $\overline{\mathbb{F}}$ such that for any integer i such that $1 \le i < p+q$ we have $f_I(x_i) \subseteq [-m;m]$; it provides a bound on initial values;
  - $k_\varepsilon = \text{BUILD}_{r_p}((a_i)_{1\le i\le p},[x_i \mapsto 0],\{x_i \mid 1 \le i \le p\}^2)$; it provides the abstract range for the overall contribution of rounding errors;
  - $k_{tail} = \text{BUILD}_{r_p}((a_i)_{1\le i\le p},[x_i \mapsto [-m_{tail};m_{tail}]],\emptyset)$, where $m_{tail} = tail\_init_b^\sharp(F,G)$; it provides the abstract range for the tail of the output stream;
- $\text{TO}_{p,q}((a_i)_{1\le i\le p+q},(b,m,k_\varepsilon,k_{tail}),I)$ is given by $[x_i \mapsto [-S_\infty;S_\infty]]$ where $S_\infty = max(S_{\le N+p} + S_\varepsilon, S_{>N+p} + S_\varepsilon)$ where:
  - $S_{\le N+p} = first\_outputs_b^\sharp(F,G) \times m$: this is a bound on the first outputs;
  - $S_{>N+p} = \text{TO}_{r_p}((a_i)_{1\le i\le p},k_{tail},I) + m \times last^\sharp(F,G)$: this is the bound

on the next outputs;
- $S_\varepsilon = \text{TO}_{r_p}((a_i)_{1\leq i\leq p}, k_\varepsilon, I)$: this is a bound on the contribution of rounding errors;
- $F(X_1, \ldots, X_p) = \sum a_k X_k$;
- $G(X_1, \ldots, X_q) = \sum a_{k+p} X_k$.

We define the tuple $\mathcal{G}_{p,q}$ as $(\mathcal{T}_{p,q}, \mathcal{B}_{p,q}, \gamma_{\mathcal{B}_{p,q}}, \sqcup_{\mathcal{B}_{p,q}}, \sqcap_{\mathcal{B}_{p,q}}, \bot_{\mathcal{B}_{p,q}}, \triangledown_{\mathcal{B}_{p,q}}, \triangle_{\mathcal{B}_{p,q}}, \text{RLVT}_{p,q}, \text{INFO}_{p,q}, \text{PT}_{p,q}, \delta_{p,q}, \text{TICK}_{p,q}, \text{BUILD}_{p,q}, \text{TO}_{p,q})$.

Theorem 32 *The tuple $\mathcal{G}_{p,q}$ is a generic extension.*

9 Higher order filter

We now explain how to deal with higher order filters. We focus on history insensitive abstractions (i.e. we only consider one input per iteration), but the framework that is described in the previous section may be used to refine the abstraction to get a history sensitive one. The main idea to analyze a simplified higher order filter is to decompose this filter into a sum of some first order filters and of some second order filters.

9.1 Background

A simplified filter of class (k, l) computes sequences (S_n) defined by the relation $S_{n+p} = a_1 S_n + \ldots + a_p S_{n+p-1} + E_{n+p}$, where the polynomial P (that is defined as $X^p - a_p X^{p-1} - \ldots - a_1 X^0$) has no multiple roots (in $\mathbb{C}$) and can be factored into the product of k second order irreducible polynomials $X^2 - \alpha_i X - \beta_i$ and l first order polynomials $X - \delta_j$.

Then, there exists sequences $(x_n^i)_{n\in\mathbb{N}}$ for any integer i such that $1 \leq i \leq k$, and $(y_n^j)_{n\in\mathbb{N}}$ for any integer j such that $1 \leq j \leq l$, such that:

$$\begin{cases} S_n = (\sum\limits_{1\leq i\leq k} x_n^i) + (\sum\limits_{1\leq j\leq l} y_n^j) \\ x_{n+2}^i = \alpha_i . x_{n+1}^i + \beta_i . x_n^i + F^i(E_{n+2}, E_{n+1}) \\ y_{n+1}^j = \delta_j . y_n^j + G^j(E_{n+1}). \end{cases}$$

where $(F^i)_{1\leq i\leq k}$ and $(G^j)_{1\leq j\leq l}$ are families of linear functions.

The families of initial outputs $(x_0^i)_{1\leq i\leq k}$, $(x_1^i)_{1\leq i\leq k}$, and $(y_0^j)_{1\leq j\leq l}$, and sub-filter inputs (given by the families of functions $(F^i)_{1\leq i\leq k}$ and $(G^j)_{1\leq j\leq l}$) can be computed by solving the system of symbolic linear relations that is obtained

by applying the equation $S_n = (\sum_{i=1}^{k} x_n^i) + (\sum_{j=1}^{l} y_n^j)$ for any integer n such that $1 \leq n \leq 2 \times p$. The solution only depends on the roots of P.

9.2 Computing the decomposition

So each time we will encounter a higher order filter, we will decompose it into a sum of first order filters and of second order filters. As we did for the formal expansion, we do not consider relative error coefficients. This way, we consider expressions of the form $\tau = a_1 X_1 + \ldots + a_n X_n + I_\varepsilon$ where I_ε encodes the contribution of rounding errors in the evaluation of the expression (i.e. it depends on the environment ρ).

To decompose the filter, we require to factorize the characteristic polynomial. Since it is a complex procedure, we do not want to rely on the soundness of the factorization. So we suppose that we have an algorithm that maps the characteristic polynomial $P = a_1 X^0 + \ldots + a_n X^{n-1}$ to a factorized polynomial P'. Then we expand P' and compute a bound on the difference between P and P' in any environment satisfying the interval constraints that held in our precondition. This bound is included inside the input of the filter. Thanks to this artifact, we avoid relying on the soundness of the factorization algorithm by adding some imprecision over the input stream. To find the initial outputs ($(x_0^i)_{1\leq i\leq k}$, $(x_1^i)_{1\leq i\leq k}$, and $(y_0^j)_{1\leq j\leq l}$) of each sub-filter and the corrective mappings ($(F^i)_{1\leq i\leq k}$ and $(G^j)_{1\leq j\leq l}$) to compute their input stream, we solve the symbolic system of $2p$ equations (that relates the first $2p$ output of the higher order filter, to the $2p$ outputs of the sub-filters. Then we compute initial outputs and the corrective mappings in floating point interval arithmetics. In the case when we cannot solve the system or when the polynomial have multiple roots, we can perturb the polynomial P' to avoid these particular cases. As a result, we add some imprecision to be added into the input stream.

Let us fix some coefficients $F = (a_i)_{1\leq i\leq p}$. Let $k(F)$ and $l(F)$ be some integers. Let $(\alpha_i(F))_{1\leq i\leq k(F)}$, $(\beta_i(F))_{1\leq i\leq k(F)}$, and $(\delta_j(F))_{1\leq j\leq l(F)}$ be some families of coefficients such that the pairs $(\alpha_i(F), \beta_i(F))$ are distinct pair wise and such that the coefficients $(\delta_j(F))$ are distinct pair wise. We denote $(a'_i(F))_{1\leq i\leq p}$ the family of coefficients such that $X^p - (a'_p(F)X^{p-1} + \ldots + a'_1(F)X^0) = \Pi(X^2 - \alpha_i(F)X - \beta_i(F)) \times \Pi(X - \delta_j(F))$. We introduce the families $(cx_i^1(F))_{1\leq i\leq k(F)}$, $(cx_i^2(F))_{1\leq i\leq k(F)}$, and $(cy_j(F))_{1\leq j\leq l(F)}$ of coefficients and the families of linear functions in $\mathbb{R}^p \to \mathbb{R}$ $(f_i^0(F))_{1\leq i\leq k(F)}$, $(f_i^1(F))_{1\leq i\leq k(F)}$, and $(g_j(F))_{1\leq j\leq l(F)}$ such that: for any input stream (E_n) and any output stream (S_n) that satisfy $S_{n+p} = a'_1(F)S_n + \ldots + a'_p(F)S_{n+p-1} + E_{n+p}$ for any $n \geq 0$, the sequences (x_n^i) and (y_n^i) that are defined by:

- for any i such that $1 \leq i \leq k(F)$,

$$\begin{cases} x_0^i = f_i^0(F)(s_0, \ldots, s_{p-1}), \\ x_1^i = f_i^1(F)(s_0, \ldots, s_{p-1}), \\ x_{n+2}^i = \alpha_i(F)x_{n+1}^i + \beta_i(F)x_n^i + cx_i^1(F)E_{n+2} + cx_i^2(F)E_{n+1}; \end{cases}$$

- for any j such that $1 \leq j \leq l(F)$,
$$\begin{cases} y_0^j = g_j(F)(s_0, \ldots, s_{p-1}) \\ y_{n+1}^j = \delta_i(F)y_n^i + cy_j(F)E_{n+1}; \end{cases}$$

satisfy: $S_n = \sum x_n^i + \sum y_n^j$, for any $n \in \mathbb{N}$.

The existence of these coefficients and of these linear functions will be useful to discover an inductive invariant for the filter output. More precisely, they are used in the definition of the concretization. It is worth noting, that, having fixed the roots of the polynomial (i.e. the coefficients $\alpha_i(F)$, $\beta_i(F)$, and $\delta_i(F)$), there is at most one solution for the other coefficients. This solution can be found by solving a linear system of equations (i.e. by looking at the first $2p$ outputs of the filters).

To compute transfer functions and conversion primitives, we require to have some interval approximation of the solution of this system. So for any coefficient $c(F)$, we introduce an interval $I_c(F) = [c'(F) - c^\varepsilon(F); c'(F) + c^\varepsilon(F)]$ such that $c(F) \in I_c(F)$. Moreover, for any linear function $f \in \mathbb{R}^p \mapsto \mathbb{R}$, we introduce a function $I_f \in \mathbb{R}^p \mapsto \mathcal{I}$, such that for any tuple $t \in \mathbb{R}^p$, we have $f(t) \in I_f(t)$. We have some algorithms to compute these intervals and these interval functions effectively. This way, we can use them in the definition of the transfer functions and of the conversion primitives.

9.3 Abstract extension

We derive the following abstract extension:

(1) structure and extrapolation operators:
- a constraint relates the last p outputs and the p parameters; but, in order to keep the notations compatible with the ones in Sect. 7, we insert p fictitious 0 parameters, so we consider $2p$ parameters; thus we set $\mathcal{T}_{r_p^{k(F),l(F)}} = (\mathcal{V}^p \times \mathbb{F}^{2p})$;
- for each filter, we collect a bound on the input stream and an abstract range for each sub-filter; thus we define $\mathcal{B}_{r_p^{k(F),l(F)}}$ as $(\mathcal{F} \times \mathcal{F}^{*k(F)+l(F)})$;
- the cup operator and the extrapolation operators are defined componentwise;
- $\gamma_{\mathcal{B}_{r_p^{k(F),l(F)}}}((a_1, 0, \ldots, a_p, 0), (m, (e_i)_{1 \leq i \leq k(F)+l(F)}))$ is the set of functions f such that there exists:
  (a) an integer $n \in \mathbb{N}$,

(b) several finite sequences $(S_o)_{0\leq o\leq n}$, $(E_o)_{0\leq o\leq n}$, $((x_o^i)_{0\leq o\leq n})_{1\leq i\leq k}$, and $((y_o^j)_{0\leq o\leq n})_{1\leq j\leq l}$

such that:

- $|E_o| \leq fst(m)$, for any o such that $0 \leq o \leq n$;
- $S_{o+p} = a'_1(F)S_o + \ldots + a'_p(F)S_{o+p-1} + E_{o+p}$, for any o such that $p \leq o \leq n$;
- $S_o = \sum x_o^i + \sum y_o^j$, for any o such that $0 \leq o \leq n$;
- for any i such that $1 \leq i \leq k(F)$,
$$\begin{cases} x_0^i = f_i^0(F)(S_0, \ldots, S_{p-1}), \\ x_1^i = f_i^1(F)(S_0, \ldots, S_{p-1}), \\ x_{n+2}^i = \alpha_i(F)x_{n+1}^i + \beta_i(F)x_n^i + cx_i^1(F)E_{n+2} + cx_i^2(F)E_{n+1}; \end{cases}$$
- for any j such that $1 \leq j \leq l(F)$,
$$\begin{cases} y_0^j = g_j(F)(S_0, \ldots, S_{p-1}) \\ y_{n+1}^j = \delta_i(F)y_n^i + cy_j(F)E_{n+1}; \end{cases}$$
- $[x_1 \mapsto x_n^i, x_2 \mapsto x_{n-1}^i] \in \gamma_{\mathcal{B}_{r_2}}((\alpha'_i(F), \alpha_i^\epsilon(F), \beta'_i(F), \beta_i^\epsilon(F)), e_i)$ for any integer i such that $1 \leq i \leq k(F)$;
- $[x_1 \mapsto y_n^i] \in \gamma_{\mathcal{B}_{r_1}}((\delta'_i(F), \delta_i^\epsilon(F)), e_{i+k})$ for any integer i such that $1 \leq i \leq l(F)$;

where $F = (a_i)$; roughly speaking, (E_n) encodes the input stream of the composite filter, (S_n) encodes the output stream, and each stream (x_n^i) and (y_n^j) encodes the output stream of a sub-filter; the abstract element m encodes a bound on the input stream; each abstract range (e_i) encodes a bound on the output stream of a sub-filter; we state that the output stream is obtained by iterating the filter with the coefficient $(a'_i(F))$; we state that the whole filter is the sum of all sub-filters; then, we state that abstract ranges abstract the last outputs of each sub-filter; this way, the concretization contains enough information to make the induction;

(2) assignments:

- $info = \mathbb{F}^{2p} \times \overline{\mathbb{F}}$
- the primitive $\textsc{rlvt}_{r_p^{k(F),l(F)}}$ is defined by pattern matching the expression in order to detect whether it corresponds to a filter iteration; in such a case, the primitive $\textsc{pt}_{r_p^{k(F),l(F)}}$ shifts the variables and collects the current input (including rounding errors), that is to say, that we collect a bound M on the expression $E - (a'_1(F)X_1 + \ldots + a'_p(F)X_n)$ when we want to iterate a filter with parameters $F = (a_1, \ldots, a_p)$;
- $\delta_{r_p^{k(F),l(F)}}((a_1, 0, \ldots, a_p, 0, M), (m, (e_i))) = (m', (e'_i))$ where
  - $m' = m \sqcup_{\mathcal{F}} (M, 0)$,
  - $e'_i = \delta_{r_2}((\alpha'_i(F), \alpha_i^\epsilon(F), \beta'_i(F), \beta_i^\epsilon(F), E_{\text{inf}}), e_i)$ for any integer i such that $1 \leq i \leq k$, where E_{inf} is a representable number such that the interval $fst(m') \times (I_{cx_i^1(F)} + I_{cx_i^2(F)})$ is included in the interval $[-E_{\text{inf}}; E_{\text{inf}}]$
  - $e'_{j+k} = \delta_{r_1}((\delta'_j(F), \delta_j^\epsilon(F), E_{\text{inf}}), e_{i+p})$ for any integer i such that

$1 \leq i \leq l$, where E_{inf} is a representable number such that the interval $fst(m') \times I_{cy_j}(F)$ is included in the interval $[-E_{\text{inf}}; E_{\text{inf}}]$;

(3) clock ticks: we define $\text{TICK}_{\mathcal{B}_{r_p^{k(F),l(F)}}}(m,(e_i))$ as follows:

$$(m, (\text{TICK}_{\mathcal{B}_{r_2}}(e_i)_{1\leq i\leq k(F)}), (\text{TICK}_{\mathcal{B}_{r_1}}(e_{j+k})_{1\leq j\leq l(F)}));$$

(4) conversion primitives:

- $\text{BUILD}_{r_p^{k(F),l(F)}}((a_i)_{1\leq i\leq p+q}, f_I, \mathcal{R}) = ((m,0),(e_i))$, where:
  - m is a bound on $|f_I(x_i)|$, for any $i \in \mathbb{N}$ such that $1 \leq i \leq p$;
  - for any i such that $1 \leq i \leq k(F)$:

$$e_i = \text{BUILD}_{r_2}((\alpha_i'(F), \alpha_i^\epsilon(F), \beta_i'(F), \beta_i^\epsilon(F)), f_i, \emptyset),$$

where $f_i = \begin{cases} x_1 \mapsto I_{f_i^0}(F)((f_I(x_i))_{1\leq i\leq p}), \\ x_2 \mapsto I_{f_i^1}(F)((f_I(x_i))_{1\leq i\leq p}); \end{cases}$

  - for any j such that $1 \leq j \leq l(F)$:

$$e_{j+k(F)} = \text{BUILD}_{r_2}((\delta_j'(F), \delta_j^\epsilon(F)), [x_1 \mapsto I_{g_i}(F)((f_I(s_i))_{1\leq i\leq p})], \emptyset));$$

- $\text{TO}_{r_p^{k(F),l(F)}}((a_i)_{1\leq i\leq p+q}, (m,(e_i)), I)$ is defined as follows:

$$\sum \text{TO}_{r_2}((\alpha_i'(F), \alpha_i^\epsilon(F), \beta_i'(F), \beta_i^\epsilon(F)), e_i, I) + \sum \text{TO}_{r_1}((\delta_j'(F), \delta_j^\epsilon(F)), e_{j+p}, I).$$

We denote by $\mathcal{G}_{r_p^{k(F),l(F)}}$ the tuple that we obtain.

Theorem 33 *The tuple $\mathcal{G}_{r_p^{k(F),l(F)}}$ is a generic extension.*

10 Benchmarks

Our framework is integrated within the ASTRÉE analyzer [1,2,7]. We now give experimental results. We tested our framework with several programs, all given by the same end-user. These programs belong to two families, that correspond to two different generations of critical embedded software written in C. The first family contains a program that has been widely used since ten years. The second family contains different versions of a program in development.

For each program we tested the ASTRE [2] analyzer with the classical domains (intervals [5], octagons [14], decision trees, and arithmetic-geometric progression domain [10]). We perform three analyses with several levels of accuracy for the abstraction of filters. First, we use no filter abstraction; then we only use history insensitive abstractions; finally we use history sensitive abstractions. For each of these analyses, we report in Fig. 11 the analysis time, the number of iterations for the main loop, and the number of warnings (in polyvariant function calls). These results have been obtained on a 2.8 GHz, 8 Gb RAM PC.

lines of C	70,000			216,000			379,000			570,000		
iterations	106	109	50	105	77	61	124	144	133	126	189	119
analysis time	1h45	2h20	1h12	6h	7h15	5h30	12h	22h20	17h	20h	42h	26h
warnings	292	3	0	203	0	0	788	0	0	1417	25	22

Fig. 11. Some statistics.

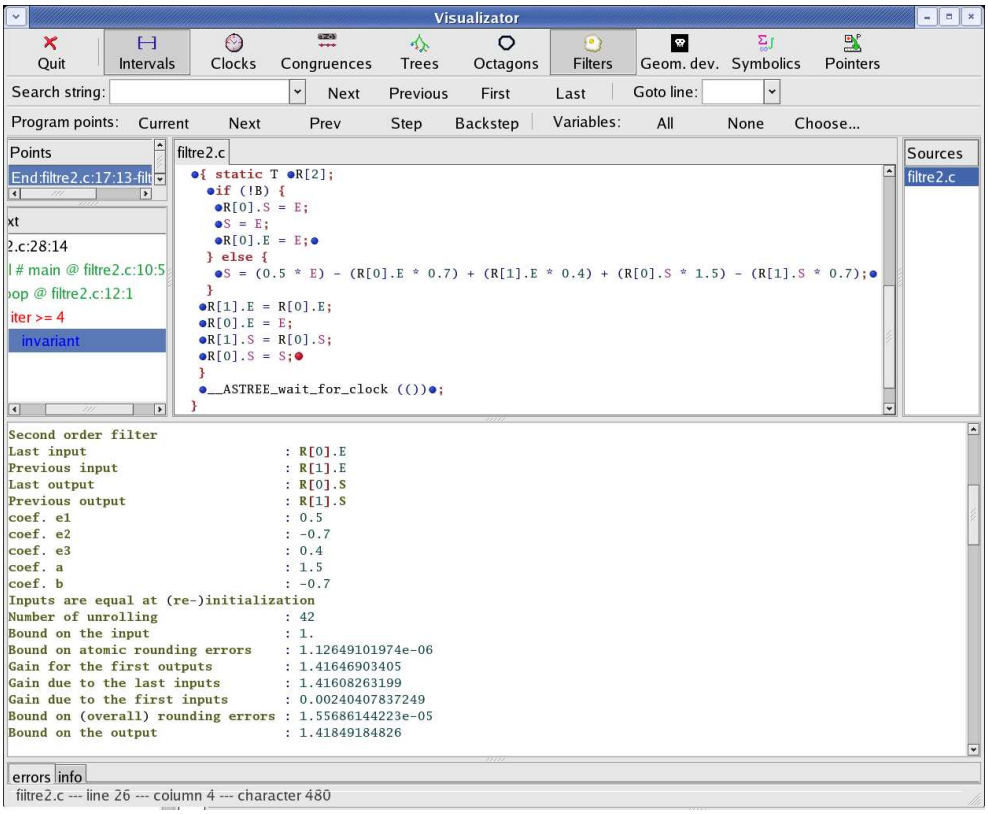


Fig. 12. The ASTRÉE interface.

With the history sensitive abstraction, the ASTRÉE analyzer reports a lot of useful information about the behavior of each filter. For each filter constraint, it gives which variables encode the current outputs, which variables encode the current inputs, which values the filter parameters take. It gives the parameter N of the formal expansion. It also gives a bound on the overall contribution of rounding errors, a bound on the input stream, a bound on the first outputs, a bound on the contribution of the last inputs, and a bound on the tail. In Fig. 12, we show a capture of the ASTRÉE interface. The ASTRÉE interface has been made by Antoine Miné.

11 Conclusion

We have proposed a highly generic framework to analyze programs with digital filtering. We have also given a general approach to instantiate this frame-

work in the case of linear filtering. We have enriched an existing analyzer, and obtained results that were far beyond our expectations. As a result, we solved nearly all remaining warnings when analyzing an industrial program of 570,000 lines of C: we obtain a sufficiently small number of warnings to allow manual inspection, and we discovered they could be eliminated without altering the functionality of the application by changing only three lines of code.

Linear filtering is widely used in the context of critical embedded software, so we believe that this framework is crucial to achieve full certification of the absence of runtime error in such programs.

11.1 Short discussions

11.1.1 About filter detections

In the presentation of our abstract domains, we use pattern matching to detect the use of filters. It works because in the programs that we analyze each filter iteration is encoded into one expression. A more general detection can be done by performing some symbolic handling of expressions (e.g. [15]). This way, we can deal with filters where the output is computed by scanning some arrays for instance. Then, an even more general detection can be obtained by considering any abstract properties that held after each assignment. This approach would be fully semantics, i.e. such a framework would rely on the properties of the environments instead of the instructions in the program. The drawback of this approach is the cost, since the inference of simple constraints such as linear equalities is cubic with respect to the number of variables. It prevents us to use this approach in an analyzer such as ASTRÉE which aims at scaling up.

11.1.2 About floating point numbers

In the design of abstract domains, floating point numbers occur twice. First, they occur in the concrete semantics. Then they occur in the implementation of abstract domains. The handling of these two kinds of floating point numbers are somehow orthogonal. To get rid of floating point numbers that occurs in the concrete semantics, we use the framework of Antoine Miné [16]. Applying the IEEE norm, we can enclose rounding errors inside intervals, then we are left with computation in the real field. This is essential to design abstract domains, since abstract domains rely on the nice structure of the real field. Then we are left with the floating point numbers that are encountered when implementing domains. We cannot afford real number computations, so we compute an over approximation of the real numbers that we need, by using

floating point number intervals. To compute the bounds of these intervals, we round each bound toward the correct direction.

11.1.3 About complex mathematical objects

The domains of expanded filters and of higher-order filters handle with complex mathematical objects. The existence of these objects are required to prove that some properties are inductive. So in the concretization, we handle with exact objects. We know that these objects exist, but we cannot, or we do not want to, compute these objects. For instance, in the case of higher order filters, we know that the output stream lies in an affine space of dimension k. We now how to characterize a decomposition of this affine space into lower dimension affine spaces. The existence of the decomposition is used in the concretization function to prove that we handle inductive invariants. Nevertheless, we never compute the exact decomposition. We only approximate some coefficients, when we want to weaken some filter constraints to compute a bound on some variables. Another point is that we have noticed that when we need to approximate a complex mathematical function (such as the polynomial factorization), it is usually possible not to rely on the soundness of the implementation of the corresponding algorithm, at the cost of a loss of accuracy in the analysis. In our example, our factorization may be not correct, but we compute an error term to be combined with the input stream.

11.2 Future works

Digital filters play an important role in critical embedded software. They usually correspond to continuous differential non-linear equations in a physical model. Usually, these equations have been discretized and linearized before the code synthesis. But, we would like to extend our framework to deal with non-linear filters, so that we could analyze software which would be closer to the physical model. The analysis of non-linear filters should also be useful when considering the dynamic systems such as biological systems.

Acknowledgments. We deeply thank the anonymous referees of the previous version [9] of this paper. We also thank Charles Hymans, Francesco Logozzo and each member of the magic team: Bruno Blanchet, Patrick Cousot, Radhia Cousot, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival.

References

[1] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. Design and implementation of a special-purpose static program analyzer for safety-critical real-time embedded software, invited chapter. In *The Essence of Computation: Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones*, LNCS 2566. Springer-Verlag, 2002.

[2] 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 *Proc. PLDI'03*. ACM Press, 2003.

[3] P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique des programmes*. PhD thesis, Université Scientifique et Mdicale de Grenoble, 1978.

[4] P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In *Proceedings of the Second International Symposium on Programming*, pages 106–130. Dunod, Paris, France, 1976.

[5] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. POPL'77*. ACM Press, 1977.

[6] P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of logic and computation*, 2(4), August 1992.

[7] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D.Monniaux, and X. Rival. The astrée analyzer. In M. Sagiv, editor, *European Symposium on Programming (ESOP'05)*, volume 3444 of *LNCS*, pages 21–30. Springer-Verlag, 2005.

[8] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proc. POPL'78*. ACM Press, 1978.

[9] J. Feret. Static analysis of digital filters. In *European Symposium on Programming (ESOP'04)*, number 2986 in LNCS. Springer-Verlag, 2004. Springer-Verlag.

[10] J. Feret. The arithmetic-geometric progression abstract domain. In *Verification, Model Checking and Abstract Interpretation (VMCAI'05)*, number 3385 in LNCS, pages 42–58. Springer-Verlag, 2005. Springer-Verlag.

[11] M. Karr. Affine relationships among variables of a program. *Acta Inf.*, 1976.

[12] A. A. Lamb, W. Thies, and S.P.Amarasinghe. Linear analysis and optimisation of stream programs. In *Proc. PLDI'03*. ACM Press, 2003.

[13] X. Leroy, D. Doligez, J. Garrigue, D. Rémy, and J. Vouillon. The Objective Caml system, documentation and user's manual. Technical report, INRIA, 2002.

[14] A. Miné. The octagon abstract domain. In *Proc. WCRE'01(AST'01)*, IEEE, 2001.

[15] A. Miné. Symbolic methods to enhance the precision of numerical abstract domains. In *VMCAI'06*, number 3855 in LNCS, pages 348–363. Springer, 2002.

[16] A. Miné. Relational abstract domains for the detection of floating-point run-time errors. In *Proc. ESOP'04*, LNCS. Springer, 2004.