ccsd-00084291, version 1 - 7 Jul 2006

Compositional analysis of floating-point linear numerical filters

David Monniaux

CNRS / Laboratoire d'informatique de l'École normale supérieure
David.Monniaux@ens.fr

Abstract. Digital linear filters are used in a variety of applications (sound treatment, control/command, etc.), implemented in software, in hardware, or a combination thereof. For safety-critical applications, it is necessary to bound all variables and outputs of all filters.
We give a compositional, effective abstraction for digital linear filters expressed as block diagrams, yielding sound, precise bounds for fixed-point or floating-point implementations of the filters.

1 Introduction

Discrete-time digital filters are used in fields as diverse as sound processing, avionic and automotive applications. In many of these applications, episodic arithmetic overflow, often handled through saturated arithmetics, is tolerable — but in safety-critical applications, it may lead to dramatic failures (e.g. disaster of the maiden flight of the Ariane 5 rocket). Our experience with the Astrée static analyzer [3] is that precise analysis of the numerical behavior of such filters is necessary for proving the safety of control-command systems using them.

We provide a method for efficiently computing bounds on all variables and outputs of any digital causal linear filter with finite buffer memory (the most common kind of digital filter). These bounds are sound with respect to fixed- or floating-point arithmetics, and may be used to statically check for arithmetic overflow, or to dimension fixed-point registers, inside the filter, or in computations using its results.

In many cases, filters are specified as diagrams in stream languages such as Simulink, SAO, Scade/Lustre, which are later compiled into lower-level languages. Our method targets such specifications, modularly and compositionally: the analysis results of sub-filters are used when analyzing a complex filter.

Our analysis results are valid for whatever range of the inputs. They can thus be used to simplify the analysis of a more complex, nonlinear filter comprising a linear sub-filter: the linear sub-filter can be replaced by its sound approximation.

In §3, we shall explain our mathematical model for the filters. In §4 we give a compositional semantics for ideal filters working on real numbers. In §5 we explain how to extract bound from this semantics. In §7, we recall some basic properties of floating-point computation. In §8 we enrich our semantics to deal with floating-point inaccuracies and other nonlinear behaviors. In §9, we consider numerical methods and implementation issues.

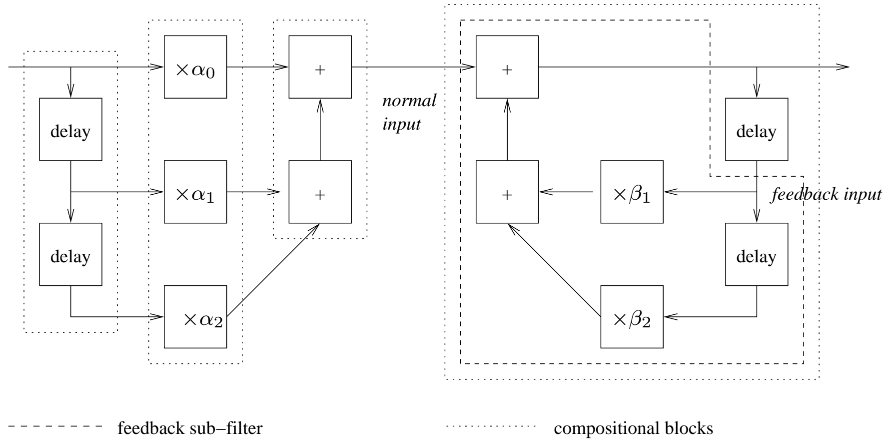

Fig. 1. Decomposition of the TF2 filter $S_n = \alpha_0 E_n + \alpha_1 E_{n-1} + \alpha_2 E_{n-2} + \beta_1 S_{n-1} + \beta_2 S_{n-2}$ into elementary blocks. The compositional blocks are chained by serial composition. Inside each compositional on the left, elementary gates are composed in parallel. On the right hand side, a feedback loop is used.

2 Introduction to linear filters and Z-transforms

Let us consider the following piece of C code, which we will use as a running example (called "TF2"):

```
Y = A0*I + A1*Ibuf[1] + A2*Ibuf[2];
O = Y + B1*Obuf[1] + B2*Obuf[2];
Ibuf[2]=Ibuf[1]; Ibuf[1]=I;
Obuf[2]=Obuf[1]; Obuf[1]=O;
```

All variables are assumed to be real numbers (we shall explain in later sections how to deal with fixed- and floating-point values with full generality and soundness). The program takes `I` as an input and outputs `O`; `A0` etc. are constant coefficients. This piece of code is wrapped inside a (reactive) loop; the *time* is the number of iterations of that loop. Equivalently, this filter can be represented by the block diagram in Fig. 1.

Let us note a_0 etc. the values of the constants and i_n (resp. y_n, o_n) the value of `I` (resp. `Y`, `O`) at time n. Then, assuming $o_k = 0$ for $k < 0$, we can develop the recurrence: $o_n = y_n + b_1.o_{n-1} + b_2.o_{n-2} = y_n + b_1.(y_{n-1} + b_1.o_{n-2} + b_2.o_{n-3}) + b_2.(y_{n-2} + b_1.o_{n-3} + b_2.o_{n-4}) = y_n + b_1.y_{n-1} + (b_2 + b_1^2 b_0).y_{n-2} + \dots$ where $\dots$ depends solely on y_k with $k < n-2$. More generally: there exist coefficients c_0, $c_1 \dots$ such that for all n, $o_n = \sum_{k=0} c_k y_{n-k}$. These coefficients solely depend on the b_k; we shall see later some general formulas for computing them.

But, itself, $y_n = a_0.i_n + a_1.i_{n-1} + a_2.i_{n-2}$. It follows that there exist coefficients c'_n (depending on the a_k and the b_k) such that $o_n = \sum_{k=0} c'_k i_{n-k}$. We again find a similar shape of formula, known as a *convolution product*. The c'_k sequence is called a *convolution kernel*, mapping i to o.

Let us now suppose that we know a bound M_I on the input: for all n, $|i_n| \leq M_I$; we wish to derive a bound M_O on the output. By the triangle inequality, $|O_n| \leq \sum_{k=0} |c'_k|.M_I$. The quantity $\sum_{k=0} |c'_k|$ is called the $l1$-norm of the convolution kernel c'.

What our method does is as follows: from the description of a complex linear filter, it compositionally computes compact, finite representations of convolution kernels mapping the inputs to the outputs of the sub-blocks of the filter, and accurately computes the norms of these kernels (or rather, a close upper bound thereof). As a result, one can obtain bounds on any variable in the system from a bound on the input.

3 Linear filters: formalism and behavior

In this section, we give a rough outline of the class of filters that we analyze and how their basic properties allow them to be analyzed.

3.1 Linear filters

We deal with numerical filters that take as inputs and output some (unbounded) discrete streams of floating-point numbers, with *causality*; that is, the output of the filter at time t depends on the past and present inputs (times 0 to t), but not on the future inputs. [1] In practice, they are implemented with state variables (in the TF2 example, the `Ibuf[]` and `Obuf[]` arrays), and the output at time t is a function of the input at time t and the internal state (resulting from time $t-1$), which is then updated. In software, this is typically one piece of a synchronous reactive loop:

```
while(true) { ...
  (state, output) = filter(state, input);
... }
```

We are particular interested in filters of the following form (or compounds thereof): if (s_k) and (e_k) are respectively the input and output streams of the filter, there exist real coefficients $\alpha_0, \alpha_1, \dots \alpha_n$ and $\beta_1, \dots \beta_m$ such that for all time t, s_t (the output at time t) is defined as: $s_t = \sum_{k=0}^{n} \alpha_k e_{t-k} + \sum_{k=1}^{m} \beta_k s_{t-k}$. In TF2, $n = m = 2$.

Consider the reaction (s_k) of the system to a unit impulse ($e_0 = 1$ and $\forall k > 0 \ e_k = 0$). If the β are all null, the filter has necessarily *finite impulse response* (FIR): $\exists N \ \forall k \geq N, s_k = 0$. Otherwise, it may have *infinite impulse response* (IIR): s_k decays exponentially if the filter is *stable*; a badly designed IIR filter may be *unstable*, and the response then amplifies with time.

It is possible to design filters that should be stable, assuming the use of real numbers in computation, but that exhibit gross numerical distortions due to the use of floating-point numbers in the implementation.

[1] There exist non-causal numerical filtering schemes; such as Matlab's `filtfilt` function. However, they require buffering the data and thus cannot be used in real time.

3.2 Formal power series and rational functions

The output streams of a linear filter, as an element of $\mathbb{R}^{\mathbb{N}}$, are linear functions of the inputs and the initial values of the state variables.

Neglecting the floating-point errors and assuming that state variables are initialized to 0, the output O is the *convolution product*, denoted $C \star I$ of the input I by some *convolution kernel* C: there exists a sequence $(q_n)_{n\in\mathbb{N}}$ of reals such that for any n, $o_n = \sum_{k=0}^{n} c_k i_{n-k}$. The filter is FIR if this convolution kernel is null except for the first few values, and IIR otherwise.

Consider two sequences of real numbers $A : (a_k)_{k\in\mathbb{N}}$ and $B : (b_k)_{k\in\mathbb{N}}$. We can equivalently note them as some "infinite polynomials" $\sum_{k=0}^{i} nfty a_k z^k$ and $\sum_{k=0}^{i} nfty b_k z^k$; such "infinite polynomials", just another notation for sequences of reals, are known as *formal power series*. This "Z-transform" notation is justified as follows: the sum $C = A + B$ of two sequences is defined by $c_n = a_n + b_n$, which is the same as the coefficient n of the sum of two polynomials $\sum_k a_k z^k$ and $\sum b_k z^k$; the convolution product $C = A \star B$ of two sequences is defined by $c_n = \sum_k a_k b_{n-k}$, the same as the coefficient n of the product of two polynomials $\sum_k a_k z^k$ and $\sum b_k z^k$.

Consider now some "ordinary" (finite) polynomials in one variable $P(z)$ and $Q(z)$; we define, as usual, the *rational function* P/Q. We shall be particularly interested in the set $\mathbb{R}[z]_{(z)}$ of such rational functions where the 0-degree coefficient of Q is 1 (note that, up to equivalence by multiplication of the numerator and the denominator by the same quantity, this is the same as requesting that the 0-degree coefficient of Q is non-zero). Any sum or product of such fractions is also of the same form. For fraction in $\mathbb{R}[z]_{(z)}$, we can compute its Taylor expansion around 0 to any arbitrary order: $P(z)/Q(z) = c_0 + c_1 z + c_2 z^2 + \ldots + c_n z^n + o(z^n)$. By doing so to any arbitrary n, we define another formal power series $\sum_{k=0}^{\infty} c_k$. We shall identify such rational fraction with the power series that it defines.

We shall see more formally in §4 that the Z-transform of the convolution kernel(s) of any finite-memory, causal linear filter is a rational function

$$\frac{\alpha_0 + \alpha_1 z + \cdots + \alpha_n z^n}{1 - \beta_1 z - \cdots - \beta_m z^m} \tag{1}$$

The above fraction is the Z-transform for a filter implementing $s_n = \sum_{k=0}^{n} \alpha_k . e_{n-k} + \sum_{k=0}^{m} \beta_. s_{n-k}$, and thus *any* ideal causal finite-memory linear filter with 1 input and 1 output is equivalent to such a filter.[2]

3.3 Bounding the response

Let $I : (i_k)_{n\in\mathbb{N}}$ be a sequence of real or complex numbers. We call l_∞-*norm* of I, if finite, and denote by $\|I\|_\infty$ the quantity $\sup_{k\in\mathbb{N}} |i_k|$. Because of the

[2] Though all designs with the same Z-transform compute the same on real numbers, they may differ when implemented in fixed- or floating- point arithmetics. Precision and implementation constraints determine the choice of the design.

isomorphism between sequences and formal power series, we shall likewise note $\|\sum_k i_k z^k\|_\infty = \sup_k |i_k|$. For a sequence (or formal series) A, we denote by $\|A\|_1$ the quantity $\sum_{k=0}^{\infty} |a_k|$, called its l_1-*norm*, if finite.

We then have the following crucial and well-known results: [7, §11.3]:

Lemma 1. *For any I, $\|I \star C\|_\infty \leq \|I\|_\infty.\|C\|_1$. Furthermore, $\|C\|_1 = \infty$, for any $M > 0$ there exists a sequence I_M such that $\|I_M \star C\|_\infty > M$.*

In terms of filters:

- If $\|C\|_1$ is finite, we can easily bound the output of the filter. Our system will thus compute (or, rather, over-approximate very closely) C for any filter.
- If $\|C\|_1 = \infty$, then the filter is *unstable*: it is possible to obtain outputs of arbitrary size by feeding appropriate sequences into the filter.

If C is the power series expansion of a rational fraction P/Q (which will be the case for all the filters we consider, see below), then we have the following stability condition:

Lemma 2. *$\|P/Q\|_1$ is finite if and only if for all $z \in \mathbb{C}$ such that $Q(z) = 0$, then $|z| > 1$.*

Unsurprisingly, our algorithms will involve some approximation of the complex roots of polynomials.

4 Compositional semantics: real field

In this section, we give a compositional abstract semantics of compound filters on the real numbers, exact with respect to input/output behavior.

4.1 Formalism

A filter or filter element has

- n_i inputs $I_1, \ldots, I_{n_i}$ (collectively, vector I), each of which is a *stream* of real numbers;
- n_r reset state values $r_1, \ldots, r_{n_r}$ (collectively, vector R), which are the initial values of the state of the internal state variables of the filter;
- n_o output streams $O_1, \ldots, O_{n_o}$ (collectively, vector O).

In TF2, $n_i = n_o = 1$, and $n_r = 4$.

If M is a matrix (resp. vector) of rational functions, or series, let $N_x(M)$ denote the coordinate-wise application of the norm $\|\cdot\|_x$ to each rational function, or series, thereby providing a vector (resp. matrix) of nonnegative reals. We note $m_{i,j}$ the element in M at line i and column j.

When computed upon the real field, a filter F is characterized by:

- a matrix $T^F \in \mathcal{M}_{n_o,n_i}(\mathbb{R}[z]_{(z)})$ such that $t_{i,j}$ characterizes the linear response of output stream i with respect to input stream j;

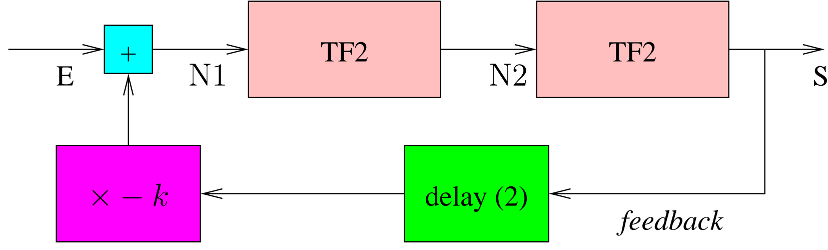

Fig. 2. A compound filter consisting of two second order filters and a feedback loop.

– a matrix $D^F \in \mathcal{M}_{n_o,n_r}(\mathbb{R}[z]_{(z)})$ such that $d_{i,j}$ characterizes the (decaying) linear response of output stream i with respect to reset value j.

We note $F(I,R)$ the vector of output streams of filter F over the reals, on the vector of input streams I and the vector of reset values R. $F(I,R) = T^F.I + D^F.R$, and thus $N_\infty(F(I,R)) \leq N_1(T^F).N_\infty(I) + N_1(D^F.R)$, which bounds the output according to the input.

When the number of inputs and outputs is one, and initial values are assumed to be zero, the characterization of the filter is much simpler — all matrices and vectors are scalars (reals, formal power series or rational functions), and D^F is null.

For most gates (addition, reorganization of wires, multiplication by a scalar, generation of a constant...), the interpretation in terms of linear application over power series, or a matrix of rational functions, is straightforward. The only difficulty is the feedback construct: given a circuit C with n_i inputs and $n_o < n_i$ outputs, feed back the outputs into some of the inputs through a unit delay; it can be shown that such systems have a unique solution, obtained by linear algebra over rational functions.

4.2 Examples

The TF2 filter of Fig. 1 is expressed by $S = \alpha_0.E + \alpha_1.\text{delay}_2(E) + \alpha_2.\text{delay}_2(E) + \beta_1.\text{delay}_1(S) + \beta_2.\text{delay}_2(S)$. This yields an equation $S = (\alpha_0 + \alpha_1 z + \alpha_2 z^2)E + (\beta_1 z + \beta_2 z^2)S$. This equation is easily solved into $S = (\alpha_0 + \alpha_1 z + \alpha_2 z^2)(1 - \beta_1 z - \beta_2 z^2)^{-1}.E$.

In Fig. 2, we first analyze the two internal second order IIR filters separately and obtain $Q_1 = \frac{\alpha_0+\alpha_1 z+\alpha_2 z^2}{1-\beta_1 z-\beta_2 z^2}$ and $Q_2 = \frac{a_0+a_1 z+a_2 z^2}{1-b_1 z-b_2 z^2}$. We then analyze the feedback loop and obtain for the whole filter a rational function with a 6th degree dominator: $S = \frac{Q_1.Q_2}{1+kz^2.Q_1.Q_2}.E$ where Q_1 and Q_2 are the transfer function of the TF2 filters (form $(\alpha_0 + \alpha_1 z + \alpha_2 z^2)(1 - \beta_1 z - \beta_2 z^2)^{-1}$), which we computed earlier.

4.3 Practical computations

To avoid problems during matrix inversion, we perform all our computations over the ring $\mathbb{Q}[z]_{(z)}$ of rational functions over the rational numbers. In §8.2 we explain how to use controlled approximation to reduce the size of the rationals and thus ensure good computation speed even with complex filters.

An alternative, at least for the filters on real numbers, is to perform all computations in $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)[z]_{(z)}$: all the coefficients of the rational functions are themselves rational functions whose variables represent the various constant coefficients inside the filter. This makes it possible to perform one computation with one particular shape of filter (i.e. class of equivalence of filters up to difference of coefficients), then use the results for concrete filters, replacing the variables by the values of the coefficients.

5 Bounding the l_∞ and l_1-norms of rational functions

In §3.3 and 4.1, we used l_1-norms of expansions of rational functions to bound the gain of filters. In this section, we explain how to over-approximate these.

Let $P(z)/Q(z) \in \mathbb{R}[z]_{(z)}$ be a rational function representing a power series by its development $(u_n)_{n\in\mathbb{N}}$ around 0. We wish to bound $\|u\|_1$, which we shall note $\|P/Q\|_1$. As we said before, most of the mass of the development of P/Q lies in its initial terms, whereas the "tail" of the series is negligible (but must be accounted for for reasons of soundness). We thus split P/Q into an initial development of N terms and a tail, and use $\|P/Q\|_1 = \|P/Q\|_1^{<N} + \|P/Q\|_1^{\geq N}$. $\|P/Q\|_1^{<N}$ is computed by computing explicitly the N first terms of the development of P/Q. We shall see in Sect. 9.2 the difficulties involved in performing such a computation soundly using interval arithmetics.

Let d_Q be the degree of Q. The development D of P/Q yields an equation $P(z) = D(z).Q(z) + R(z).z^N$. We have $P(z)/Q(z) = D(z) + R(z)/Q(z).z^N$, thus $\|P/Q\|_1^{\geq N} = \|R/Q\|_1 \leq \|R\|_1.\|1/Q\|_1$.

There exist a variety of methods for bounding $\|1/Q\|_1$ using the zeroes of $Q(z)$. One uses the following lemma:

Lemma 3. *If $P(z)/Q(z)$ is a rational function such that $Q(0) \neq 0$ and Q is monic (leading coefficient equal to 1), with roots (counted with their multiplicity) $\xi_1, \ldots, \xi_n$, then $\|P/Q\|_1 \leq \|P\|_1.(|\xi_1| - 1)^{-1} \ldots (|\xi_n| - 1)^{-1}$.*

$\|R\|_\infty$ is bounded by explicit computation of R using interval arithmetics; as we shall see (§9.2), we compute D until the sign of the terms is unknown — that is, when the norm of the developed signal is on the same order of magnitude as the numerical error on it, which happens, experimentally, when the terms are very small in absolute values. Therefore, $\|R\|_\infty$ is small, and thus the roughness of the approximation used for $\|1/Q\|_1$ does not matter much in practice.

The same method way be used for bounding the l_∞-norm: explicit computation of the norm over a finite development, and bounding of the (negligible) tail, if necessary by $\|R\|_\infty \leq \|R\|_1$.

6 Complex nonlinear iterated filter

We now consider a nonlinear, iterated filter due to Roozbehani et al. [11][§5]. We first analyze separately `filter1()` (2nd-order linear filter) and `filter2()` (2nd-order affine filter). So as to simplify matters, we do not give the transfer functions using matrices, matrices inverses etc. but as the solution of a system of linear equations over polynomials in z. We obtain that system very simply from the program: whenever we see an assignment $x := e$, we turn it into an equation $x = e$ (we assume without loss of generalities that variables are only assigned once in a single iteration step), where e is the original expression where a variable v that has not yet been assigned in the current iteration is replaced by $i_v + z.v$, i_v standing for the initialization value of v.

```
void filter1 () {
  static float E[2], S[2];
  if (INIT1) {
    S[0] = X; P = X;
    E[0] = X; E[1]=0; S[1]=0;
  } else {
    P =0.5*X-0.7*E[0] +0.4*E[1]
      +1.5*S[0]-S[1]*0.7;
    E[1] = E[0];
    E[0] = X;
    S[1] = S[0];
    S[0] = P;
    X=P/6+S[1]/5;
  }
}
```

$$\begin{aligned}
&p = 0.5e - 0.7(i_{e_0} + z.e_0)\\
&\quad +0.4(i_{e_1} + z.e_1) + 1.5(i_{s_0} + z.s_0) - 0.7(i_{s_1} + z.s_1)\\
&e_1 = i_{e_0} + z.e_0\\
&e_0 = e\\
&s_1 = i_{s_1} + z.e_1\\
&s_0 = p\\
&x = p/6 + s_1/5
\end{aligned}$$

We call e the input value for `X`. We solve the system and obtain $x = Q.e + Q_{i_{e_0}}.i_{e_0} + Q_{i_{e_1}}.i_{e_1} + Q_{i_{s_0}}.i_{s_0} + Q_{i_{s_1}}.i_{s_1}$. The common denominator of the Q fractions is $10 - 15z + 7z^2$, which has complex conjugate roots z such that $|z| \simeq 1.2$. $i_{e_1} = i_{s_1} = 0$ and $i_{e_0} = i_{s_0} = \iota$ (the last value for input e such that `INIT1` is true), thus $\|x\|_\infty \leq \|Q\|_1.\|e\|_\infty + \|Q_{i_{e_0}} + Q_{i_{s_0}}\|_\infty.\|\iota\|$. With a precondition $\|e\|_\infty \leq 400$, this yields $\|x\|_\infty < 339$. If we take the coarser inequality $\|x\|_\infty \leq \|Q\|_1.\|e\|_\infty + (\|Q_{i_{e_0}}\|_\infty + \|Q_{i_{s_0}}\|_\infty).\|\iota\|$ we get $\|x\|_\infty < 528$. Roozbehani et al. find a bound $\simeq 531$.

```
void filter2 () {
  static float E2[2], S2[2];
  if (INIT2) {
    S2[0] =0.5*X; P = X;
    E2[0] = 0.8*X; E2[1]=0; S2[1]=0;
  } else {
    P =0.3*X-E2[0]*0.2+E2[1]*1.4
      +S2[0]*0.5-S2[1]*1.7;
    E2[1] = 0.5*E2[0];
    E2[0] = 2*X;
```

$$\begin{aligned}
&p = 0.3e - 0.2(i_{e_0} + z.e_0)\\
&\quad +1.4(i_{e_1} + z.e_1) + 0.5(i_{s_0} + z.s_0) + 1.7(i_{s_1} + z.s_1)\\
&e_1 = 0.5(i_{e_0} + z.e_0)\\
&e_0 = 2e
\end{aligned}$$

```
    S2[1] = S2[0]+10;                s_1 = i_{s_0} + z.s_0 + τ
    S2[0] = P/2+S2[1]/3;             s_0 = p/2 + s_1/3
    X=P/8+S2[1]/10;                  x = p/8 + s_1/10
  }
}
```

We proceed similarly (with the introduction of $\tau = 10/(1-z)$ and obtain $x = Q.e + Q_{i_{e_0}}.i_{e_0} + Q_{i_{e_1}}.i_{e_1} + Q_{i_{s_0}}.i_{s_0} + Q_{i_{s_1}}.i_{s_1} + Q_c$. The common denominator of the Q is $60 + 35z + 51z^2$, with complex conjugate roots z such that $|z| \simeq 1.08$. Then $\|x\|_\infty \leq \|Q\|_1.\|e\|_\infty + \|0.8Q_{i_{e_0}} + 0.5Q_{i_{s_0}}\|_\infty.\|\iota\| + \|Q_c\|_\infty$. This yields $\|x\|_\infty \leq 1105$.

The two linear filters are combined into an iterated nonlinear filter. `filter1()` (resp. `filter2()`) is run with a pre-condition of `X` $\in [-400, 400]$ (resp. $[-800, 800]$). We replace the call to the filter by its postcondition `X` $\in [-339, 339]$ (resp. `X` $\in [-1105, 1105]$).

The program then can be abstracted into:

```
while (TRUE) {
  X = 0.98 * X + 85;
  maybe choose X in [−1155, 1055]; }
```

We obtain `X` $\in [-1155, 4250.02]$ by running Astrée with a large number of narrowing iterations, whereas Astrée cannot analyze the original program precisely and cannot bound `X`. In this case, the exact solution $[-1155, 4250]$ ($x = 0.98x + 85$ has for unique solution $x = 4250$) could have been computed algebraically, but in more complex filters this would not have been the case. Roozbehani et al. have a bound of 4560.

```
void main () {
  X = 0;
  INIT1 = TRUE; INIT2=TRUE;
  while (TRUE) {
    X = 0.98 * X + 85;
    if (abs(X)<= 400) {
      filter1 ();
      X=X+100;
      INIT1=FALSE;
    } else
    if (abs(X)<=800) {
      filter2();
      X=X-50;
      INIT2=FALSE;
    }
}}
```

Note that the non-abstracted program converges to a value $\simeq 205$, with `X` $\in [0, 209]$. However, this very simple program illustrates our methodology for compositional analysis: finding the optimal solution is possible here because the program is simple, but would not be possible in practice if we had added more nonlinear behavior and nondeterministic inputs, as in real-life reactive code; whereas by analyzing precisely each linear filter and plugging the results back into a generic analyzer, we get reasonable results.

7 Precision properties of fixed- or floating-point operations

Most types of numerical arithmetics, including the widely used IEEE-754 floating-point arithmetic, implemented in hardware in all current microcomputers, define the result of elementary operations as follows: if f is the ideal operation (addition, subtraction, multiplication, division etc.) over the real numbers and $\tilde{f}$ is

the corresponding floating-point operation, then $\tilde{f} = r \circ f$ where r is a *roundoff* function, depending on the current rounding mode.

In this description, we leave out the possible generation of special values such as infinities ($+\infty$ and $-\infty$) and *not-a-number* (NaN). We assume as a precondition to the numerical filters that we analyze that they are not fed infinities or NaNs.

Our framework provides constructive methods for bounding *any* floating-point quantity x inside the filters as $\|x\|_\infty \leq c_0 + \sum_{k=1}^{n} c_k . \|e_k\|_\infty$ where the e_k are the input streams of the system; it is quite easy to check that the system does not overflow ($\|x\| < M$); one can even easily provide some very wide sufficient conditions on the input ($\|e_k\|_\infty \leq (M - c_0)/(\sum_{k=1}^{n} c_k)$). We will not include such conditions in our description, for the sake of simplicity.

For any arithmetic operation, the discrepancy between the ideal result x and the floating-point result $\tilde{x}$ is bounded, in absolute value, by $\max(\varepsilon_{\text{rel}}|x|, \varepsilon_{\text{abs}})$ where ε_{abs} is the *absolute error* (the least positive floating-point number)[3] and ε_{rel} is the *relative error* incurred. ε_{abs} and ε_{rel} depend on the floating-point type used and possible rounding modes. We actually take the coarser inequality $|x - \tilde{x}| \leq \varepsilon_{\text{rel}}|x| + \varepsilon_{\text{abs}}$. See [1] for more details on floating-point numbers and [9] for more about the affine bound on the error.

In the case of fixed-point arithmetics, we have $\varepsilon_{\text{rel}} = 0$ and $\varepsilon_{\text{abs}} = \delta$ (δ is the smallest positive fixed-point number) if the rounding mode is unknown (round to $+\infty$, $-\infty$ etc.) and $\delta/2$ is it is the rounding mode is known to be round-to-nearest.

8 Compositional semantics: fixed- and floating-point

8.1 Constraint on the errors

We now enrich our compositional abstract semantics to reflect numerical errors. Our enriched semantics characterizes a fixed- or floating-point filter $\tilde{F}$ by the exact semantics of the associated filter F over the real numbers and a bound on the discrepancy $\Delta(I) = \tilde{F}(I) - F(I)$ between the ideal and floating-point filters.

Assuming for the sake of simplicity a single input and a single output and no initialization conditions, we obtain an *affine*, *almost linear* constraint on $\|\Delta(I)\|\infty$: $\|\Delta(I)\|_\infty \leq \varepsilon_{\text{rel}}^F \|I\|_\infty + \varepsilon_{\text{abs}}^F$. In short: since the filter is linear, the magnitude of the error is (almost) linear. We generalize this idea to the case of multiple inputs and outputs. The abstract semantics characterizing Δ is given by matrices $\varepsilon_{\text{rel},T}^F \in \mathcal{M}_{n_o,n_i}(\mathbb{R}_+)$ and $\varepsilon_{\text{rel},D}^F \in \mathcal{M}_{n_o,n_r}(\mathbb{R}_+)$ and a vector $\varepsilon_{\text{abs}}^F \in \mathbb{R}_+^{n_o}$ such that $\|F(I,R) - \tilde{F}(I,R)\|_\infty \leq \varepsilon_{\text{rel},T}^F . N_\infty(I) + \varepsilon_{\text{rel},D}^F . N_\infty(R) + \varepsilon_{\text{abs}}$. where $\tilde{F}(I,R)$ is the output on the stream computed upon the *floating-point* numbers

[3] The absolute error results from the *underflow* condition: a number close to 0 is rounded to 0. Contrary to overflow (which generates infinities, or is configured to issue an exception), underflow is generally a benign condition. However, it precludes merely relying on relative error bounds if one wants to be sound.

on input streams I and initial values I. As before, the matrices for a complex filter may be computed compositionally from the matrices for the sub-filters.

Let us for instance consider the instruction `t = (x + y) + z` where all variables are from the same fixed- or floating-point type (say, IEEE double precision). We shall note $\oplus$ (resp. $\otimes$) the machine operation corresponding to the ideal $+$ (resp $\times$) on reals. Then $x \oplus y = x + y + \varepsilon_1$, with $\varepsilon_1 \leq \varepsilon_{\text{rel}}.|x+y| + \varepsilon_{\text{abs}} \leq \varepsilon_{\text{rel}}.|x| + \varepsilon_{\text{rel}}.|y| + \varepsilon_{\text{abs}}$. Also, $(x \oplus y) \oplus z = (x \oplus y) + z + \varepsilon_2$, with $\varepsilon_2 \leq \varepsilon_{\text{rel}}.|(x \oplus y) + z| + \varepsilon_{\text{abs}} \leq \varepsilon_{\text{rel}}.|x+y+z+\varepsilon_1| + \varepsilon_{\text{abs}} \leq \varepsilon_{\text{rel}}.(1+\varepsilon_{\text{rel}}).|x| + \varepsilon_{\text{rel}}.(1+\varepsilon_{\text{rel}}).|y| + \varepsilon_{\text{rel}}.|z| + \varepsilon_{\text{abs}}.(1+\varepsilon_{\text{rel}})$. Then $(x \oplus y) \oplus z = x + y + z + (\varepsilon_1 + \varepsilon_2)$ with $|\varepsilon_1 + \varepsilon_2| \leq \varepsilon_{\text{rel}}.(2+\varepsilon_{\text{rel}}).|x| + \varepsilon_{\text{rel}}.(2+\varepsilon_{\text{rel}}).|y| + \varepsilon_{\text{rel}}.|z| + \varepsilon_{\text{abs}}.(2+\varepsilon_{\text{rel}})$.

8.2 Trading accuracy for speed; nonlinear elements

We have split the behavior of the filter into the sum of the convolution of the input signal by the power development of a rational function, representing the exact behavior, and some error term. If we compute the rational functions exactly over $\mathbb{Q}[z]_{(z)}$, then the rational coefficients might grow expensively large. We can actually take shorter approximations of these coefficients and absorb the error that we introduce into the error term.

An ideal filter of Z-transform P/Q with no initialization condition, $P(z) = \sum_{k=0}^{k} \alpha_k z^k$ and $Q(z) = \sum_{k=0}^{k} \beta_k z^k$ is equivalent to a filter as described in §3.1. Such a filter may be soundly approximated by a non-ideal feedback filter $F^\sharp$ with $T_I^{F^\sharp} = P^\sharp$, $T_O^{F^\sharp} = Q^\sharp$, $\varepsilon_{\text{rel},I} = \|P^\sharp - P\|_1$, $\varepsilon_{\text{rel},O} = \|Q^\sharp - Q\|_1$, $\varepsilon_{\text{abs}} = 0$.

More generally: a filter F (Z-transform P/Q) may be approximated by a filter $F^\sharp$ ($P^\sharp/Q^\sharp$) with transfer function $T^{F^\sharp} = T^G$, $\varepsilon_{\text{rel},T}^{F^\sharp} = \varepsilon_{\text{rel},T}^{F} + \varepsilon_{\text{rel},T}^{G}$, $\varepsilon_{\text{rel},D}^{F^\sharp} = \varepsilon_{\text{rel},D}^{F} + \varepsilon_{\text{rel},D}^{G}$, $\varepsilon_{\text{abs}}^{F^\sharp} = \varepsilon_{\text{abs}}^{F}$ where G is the feedback filter whose internal filter H is given by $T_I^H = P^\sharp$, $T_O^H = Q^\sharp$, $\varepsilon_{\text{rel},I}^H = \|P^\sharp - P\|_1$, $\varepsilon_{\text{rel},I}^H = \|Q^\sharp - Q\|_1$, $\varepsilon_{\text{abs}}^H = 0$. In this way, a nonlinear sub-filter can be approximated by a linear part and a nonlinear part, the latter being constrained by ε_{rel} and ε_{abs}.

9 Numerical considerations and implementation

We have so far given many mathematical formulas that are exact in the *real* field. In this section, we explain how to obtain sound abstractions for these formulas using floating-point arithmetics.

We implemented the algorithms described here. As an example, the serial composition of the filter in Fig. 1 and another TF2 filter, all with realistic coefficients, is analyzed in about 0.04 s on a recent PC; the analyzer finds that $\|S\| \leq g\|E\|$ with $g \simeq 2$, with $\varepsilon_{\text{rel}} \simeq 10^{-12}$ and $\varepsilon_{\text{abs}} \simeq 10^{-305}$.

For filters implemented over the real numbers, the computation of the rational fractions representing the convolution kernels can be performed using arbitrary precision arithmetics; no loss of precision is entailed. When computing the formal development and its sum, one can use floating-point numbers in round-to-$+\infty$ and round-to-$-\infty$ modes and obtain lower and upper bounds, thus also deriving

a bound on the computation error; similar bounds may be obtained for the estimate of the norm of the tail. We applied the method to filters extracted from industrial codes; in all cases, the error bounds were small. In practice, outside of artificial cases, floating-point arithmetics does not add significantly to the bounds.

9.1 Interval arithmetics

IEEE floating-point arithmetics [1] and good extended precision libraries such as MPFR[4] provide functions computing *upward rounded* (or *rounded-to-*$+\infty$) and *downward rounded* (or *rounded-to-*$-\infty$) results: that is, if $f(x_1,\ldots,x_n)$ is the exact operation on real numbers and $\tilde{f}^-$ and $\tilde{f}^+$ are the associated floating-point downward and upward operations, then $f(x_1,\ldots,x_n)$ is guaranteed to be in the interval $[\tilde{f}^-(x_1,\ldots,x_n),\tilde{f}^+(x_1,\ldots,x_n)]$, which will guarantee the *soundness* of our approach. Furthermore, for many operations, $\tilde{f}^-(x_1,\ldots,x_n)$ and $\tilde{f}^+(x_1,\ldots,x_n)$ are guaranteed to be optimal; that is, no better bounds can be provided within the desired floating-point format; this will guarantee local *optimality* of certain of our elementary operations.

9.2 Computation of developments

When bounding the norm $\|P/Q\|_1$ of a series quotient of two polynomials, we split the series into its N initial terms of development, which we compute explicitly, and a tail whose norm we bound. The first idea is to compute the N first terms of the series by quotienting the series, as explained in Sect. 3.2 or, equivalently, by running the filter for N iterations on the Dirac input $1,0,0,\ldots$. In order to provide a sound result, one would work using interval arithmetics over floating-point numbers. However, as already noted by Feret, after some number of iterations the sign of the terms becomes unknown and then the magnitude of the terms increase fast; it is therefore indicated to compute the development until the first term of unknown sign is reached, and assign N accordingly (one may still also enforce a maximal number of iterations $N_{\max}$). In order to be able to develop the quotient further with good precision, one can use a library of extended-precision floating-point computations.

9.3 Bounding the roots

In order to bound $\|P/Q\|_1$, we have to get lower bounds of the absolute values of the roots of Q. For this, we want to obtain discs $D(x_j,\rho_j)$ such that $|x_j-\xi_j|\leq\rho_j$ where the ξ_j are the roots of Q counted with their multiplicities.

Our polynomial coefficients turned into floating-point intervals $[l_k,h_k]$; it is expected that the h_k-l_k are small. This suggests to us a two-step method for obtaining the desired bounds:

[4] `http://www.mpfr.org`

1. Use an efficient and, in practice, very accurate algorithm to obtain *approximations* x_j to the roots of $\sum_{k=1}^{n} \frac{l_k+h_k}{2} z^k$ (the "midpoint polynomial"). We used `gsl_poly_complex_solve` of the GNU Scientific Library [5], which is based on an eigenvalue decomposition of the companion matrix.
2. From those approximations, obtain bounds on the radius of the error committed. There exist a variety of bounding methods [12] which take a polynomial and approximate roots as an input and output error radii; these methods may be performed using interval arithmetics. We implemented the simplest and roughest one [12, Th. 3.1]: ξ_j is in a closed disc of center $x_j - \rho_j$ and radius $|\rho_j|$ where $\rho_j = (nP(x_j))/(2p_n \prod_{k \neq j} x_j - x_k)$.

10 Related works and applications

In the field of digital signal processing, some sizable literature has been devoted to the study of the effects of fixed-point and floating-point errors on numerical filters. While the fact that the l_1-norm of the convolution kernel is what matters for judging overflow, it is argued that this norm is "overly pessimistic" [7, §11.3] [6, eq 13], not to mention the difficulties in estimating it. In practice, filter designers have preferred criteria that indicate no saturation for most "commonplace" inputs, excluding pathological inputs. As a consequence, most studies model the errors as random sources of known distribution, independent of each other and with no temporal correlation [2,10]. This allows estimating the energy spectrum (l_2-norm) of the *typical* numeric noise; however, this does not work for our purpose, which is to provide sound bounds valid in all circumstances.

J. Feret has proposed an abstract domain for analyzing programs comprising digital linear filters [4]. He provides effective bounds for first and second degree filters. In comparison, we consider more complex filter networks, in a compositional fashion; but we analyze specifications, and not C code (which is usually compiled from those specifications, with considerable loss of structure). Another difference is that we do not perform abstract iterations. Feret's method currently considers only second-order filters (i.e. TF2), though it may be possible to adapt it to higher-order filters. On second-order filters, the bounds computed by Feret's method and the method in this paper are very close (since both are based on a development of the convolution kernel, though they use different methods of tail estimation).

Lamb et al. [8] have proposed effective methods, based on linear algebra, for computing equivalent filters for DSP optimization. They do not compute bounds, nor do they study floating-point errors.

Roozbehani et al. [11] find program invariants by Lagrangian relaxation and semidefinite programming, with quadratic invariants. In order to make problems tractable, they too apply a blockwise abstraction. The class of programs that they may analyze directly is potentially larger, but the results are less precise than our method on some linear filters. They do not handle floating-point imprecisions (though this can perhaps be added to their framework).

One possible application of our method would be to integrate it as a pre-analysis pass of a tool such as Astrée [3]. Astrée computes bounds on all floating-point variables inside the analyzed program, in order to prove the absence of errors such as overflow. In order to do so, it needs to compute reasonably accurate bounds on the behavior of linear filters. A typical fly-by-wire controller contains dozens of TF2 filters, some of which may be integrated into more complex feedback loops; in some cases, separate analysis of the filters may yield too coarse bounds.

11 Conclusions and future works

We have proposed effective methods for providing sound bounds on the outcome of complex linear filters from their flow-diagram specifications, as found in many applications. Computation times are modest; furthermore, the nature of the results of the analysis may be used for modular analyses — the analysis results of a sub-filter can be stored and never be recomputed until the sub-filter changes.

In the future, we plan to provide abstract domains suitable for the analysis of source code as written in an imperative language such as C, in order to extract the filter specification and arithmetic errors from the source.

References

1. *IEEE Standard for Binary Floating-Point Arithmetic*. IEEE 754.
2. Bruce W. Bomar et al. Roundoff noise analysis of state-space digital filters implemented on floating-point digital signal processors. *IEEE Trans. on Circuits and Systems II*, 44(11):952–955, 1997.
3. P. Cousot et al. The ASTRÉE analyzer. In *ESOP*, number 3444 in LNCS, pages 21–30, 2005.
4. Jérôme Feret. Static analysis of digital filters. In *ESOP '04*, number 2986 in Lecture Notes in Computer Science. Springer-Verlag, 2004.
5. Free Software Foundation. *GSL — GNU scientific library*, 2004.
6. Leland B. Jackson. On the interaction of roundoff noise and dynamic range in digital filters. *The Bell System Technical J.*, 49(2):159–184, February 1970.
7. Leland B. Jackson. *Digital Filters and Signal Processing*. Kluwer, 1989.
8. Andrew A. Lamb, William Thies, and Saman Amarasinghe. Linear analysis and optimization of stream programs. In *PLDI '03*, pages 12–25. ACM, 2003.
9. A. Miné. Relational abstract domains for the detection of floating-point run-time errors. In *ESOP'04*, volume 2986 of *LNCS*, pages 3–17. Springer, 2004.
10. Bhaskar D. Rao. Floating point arithmetic and digital filters. *IEEE Trans. on Signal Processing*, 40(1):85–95, January 1992.
11. M. Roozbehani, E. Feron, and A. Megretski. Modeling, optimization and computation for software verification. In *HSCC*, number 3414 in Lecture Notes in Computer Science, page 606. Springer Verlag, 2005.
12. Siegfried M. Rump. Ten methods to bound multiple roots of polynomials. *J. of Computational and Applied Math.*, 156(2):403–432, 2003.