# The interval analysis of multilinear expressions

Cosimo Laneve[1] Tudor Lascu[1] Vania Sordoni[2]

[1] Dipartimento di Scienze dell'Informazione, Università di Bologna
[2] Dipartimento di Matematica, Università di Bologna

June 2, 2010

**Abstract**

Expressions are multilinear when variable occurrences are linear and products have factors using different variables. We demonstrate that multilinear expressions are either constant or have never a local minimum or a local maximum. Therefore the interval of multilinear expressions may be computed precisely studying their values at the bounds of the variables therein. We then propose a technique for the interval analysis of generic expressions that transforms them into multilinear ones and compute the interval of the latters.

## 1 Introduction

Interval analysis is a static analysis technique that abstractly computes programs using ranges of variables' values rather than specific values. For example, if $x \in [1,2]$ and $y \in [0,4]$ and one performs the assignment $z := x+y$, then, in the final state, $z \in [1,6]$, which is the outcome of the interval addition $[1,2]+[0,4]$.

It is well-known that the lost of precision in interval computations is due to the evaluation of nonlinear expressions such as $x*(y-z)+z$. In this case, when $x \in [0,1]$, $y \in [0,10]$ and $z \in [0,10]$, the current interval techniques compute the expression $[0,1]*([0,10]-[0,10])+[0,10]$ and yield the interval $[-10,20]$, which is a strict over-approximation of the precise result $[0,10]$. This lost of precision is caused by the double occurrence of the variable $z$ that, in a precise evaluation, may reduce the range of the result (because every occurrence of a variable must be replaced by the same value). In an interval evaluation the two occurrences of $z$ disappear because they are replaced by the corresponding interval. The same argument may be used for polynomials in several variables, which is a computational equivalent problem.

In facts, finding the (precise) range of an expression in several variables has been proved to be NP-hard by Gaganov [2] with respect to the number of variables and the degree of the expression. The problem seems practically unfeasible even if we constrain either the degree or the number of variables of the expression. In particular, the problem is NP-hard as long as the degree is greater than 1. If the number $n$ of variables is fixed, Grigoriev and Vorobjov have designed an $O(d^k)$ algorithm, where $d$ is the degree of the expression [3]. Unfortunately, the constant $k$ is equal to $n^2$, which makes the algorithm expensive even for small values of $n$. The survey [6] reports the main results about this problem.

Constraining either the degree or the number of variables are two somehow extreme restrictions. Other restrictions, retaining simple and even more performant algorithms than $O(d^{n^2})$, may be proposed. To this aim, we have parsed several programs that have been developed for different purposes (several thousands lines) and we noticed that a large number of expressions were *multilinear*. An expression is multilinear when variable occurrences are linear – the exponent is 1 – and products have factors using different variables. For example $x * (y - z) + z$ and $x * (y * z - u) + y * u$ are multilinear (for simplicity we are omitting constant coefficients) and $x * (y + x)$ is not.

We demonstrate that multilinear expressions never manifest a local minimum or a local maximum. That is, if a multilinear expression $E$ has variables $x_1, \cdots,$ $x_n$ that ranges over $[a_1, b_1], \cdots, [a_n, b_n]$, respectively, then the least and greatest values of $E$ can be found at the vertices of the hypercube $[a_1, b_1] \times \cdots \times [a_n, b_n]$. Therefore, the range of multilinear expressions may be computed in a precise way by collecting the values of $E$ at the $2^n$ vertices and taking the least and greatest ones. This simple algorithm has a computational complexity $O(n \cdot 2^{2n})$.

We then use this result to design a technique for evaluating generic expressions. The idea is to transform an expression into a multilinear one and then compute the range of the latter. The transformation amounts to replace nonlinear variables with fresh linear ones whose interval is defined by the corresponding exponential variable (directly, rather than as a sequence of products). In doing this replacement, we keep the dependencies between variable's occurrences as much as possible. For example, the expression $x^3y + x^2z + xyz$, with $x \in [-1, 1]$, is transformed into $uxy + uz + xyz$ by letting $u = x^2$ and $u \in [0, 1]$. The technique is sound, *i.e.* it introduces over-approximations as in the transformation of the expression $x^4y + x^3z + xyz$, with $x \in [-2, 2]$. In this case we obtain $uxy + uz + xyz$, by letting $u = x^3$ and $u \in [-8, 8]$, and we notice that the range of the subexpression $ux$ is $[-16, 16]$, whilst it is $[0, 16]$ in the original expression.

We finally compare our technique with standard interval analysis and with a recent technique proposed by Miné [5]. This comparison is rather preliminary: a thorough study is delayed to the next future.

**Related works.** Several proposals for reducing the loss of precision of interval arithmetics may be found in the literature. As usual, in every proposal there is a trade-off between computational cost and precision.

A recent proposal, which has been integrated in ASTREÉ [1], is due to Miné [5]. In this technique an expression is transformed into an expression of degree 1, called *affine*. Intervals of affine expressions are then computed without loss of precision. The problem of this technique is that the affine transformation is not unique and may introduce over-approximations (Miné's technique, applied to $x * (y - z) + z$, when $x \in [0, 1]$, $y \in [0, 10]$ and $z \in [0, 10]$, yields $[0, 20]$, which is an interval (twice) longer than the precise result $[0, 10]$).

The solution in [4] requires advanced computational techniques such as the study of monotony and the analysis of sub-intervals.

**Overview of the paper.** We recall the mathematical background and define multilinear expressions and demonstrate our results in Section 2. The algorithm for the interval analysis of multilinear expressions and the study of its computational complexity are in Section 3. In Section 4 we study the extension of our technique to generic expression. Section 5 reports our conclusions.

## 2 Multilinear functions

A *polynomial function* is a function defined by a polynomial. For example, the function $f$ from real numbers $\mathbb{R}$ to $\mathbb{R}$, defined by $f(x) = 5x^3 + 2x + 7$ is a polynomial function of one argument. Polynomial functions of multiple arguments can be defined, using polynomials in multiple variables, as $f(x, y) = xy^2 + 2xy + y + 1$.

The gradient of a (differentiable) function $f : \mathbb{R}^n \to \mathbb{R}$, noted $\nabla f$, is the $n$-uple $(\frac{\partial f}{\partial x_1}, \cdots, \frac{\partial f}{\partial x_n})$. For example, when $g(x, y) = x^2 + y^2 - 2xy^2 - y^4$, $\nabla g(x, y) = (2x - 2y^2, 2y - 4xy - 4y^3)$. The gradient of a function is relavant because a 0-gradient, *i.e.* $(\nabla g)(a_1, \cdots, a_n) = (0, \cdots, 0)$, is a necessary condition for $(a_1, \cdots, a_n)$ being a local minimum or a local maximum. This condition is in general not sufficient: in the case of the above function $g$, $(0, 0)$, $(\frac{1}{4}, \frac{1}{2})$ and $(\frac{1}{4}, -\frac{1}{2})$ are the zeros of the gradient, however only $(0, 0)$ is a local minimum.

**Definition 2.1.** A polynomial function $f : \mathbb{R}^n \to \mathbb{R}$ is called *multilinear* if

$$f(x_1, \cdots, x_n) = \sum_{S \subseteq \{1,2,\cdots,n\}} c_S \cdot \prod_{i \in S} x_i$$

where every $c_S$ is a constant in $\mathbb{R}$.

The Taylor series of a multilinear function $f : \mathbb{R}^n \to \mathbb{R}$ in a neighborhood of a point $(a_1, \cdots, a_n)$ is a polynomial of the form

$$\begin{array}{rl} f(a_1, \cdots, a_n) + & \sum_{i \in \{1,\cdots,n\}} \frac{\partial f}{\partial x_i}(a_1, \cdots, a_n)(x_i - a_i) \\ + & \sum_{i_1, i_2 \in \{1,\cdots,n\}, i_1 \neq i_2} \frac{1}{2!} \frac{\partial f}{\partial x_{i_1} \partial x_{i_2}}(a_1, \cdots, a_n)(x_{i_1} - a_{i_1})(x_{i_2} - a_{i_2}) \\ + & \cdots \end{array}$$

For example, the Taylor series of $f(x, y) = 2xy + 7y + 12$ in a neighborhood of a point $(a, b)$ is

$$f(a, b) + 2b(x - a) + (2a + 7)(y - b) + (x - a)(y - b) .$$

The Taylor series approximate the value of functions at given points. We will use them in the following theorem.

**Theorem 2.2.** *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a multilinear function. If $f$ has a local minimum or a local maximum then $f$ is constant.*

*Proof.* Let $f(x_1, \cdots, x_n) = \sum_{S \subseteq \{1,2,\cdots,n\}} c_S \cdot \prod_{i \in S} x_i$ be a multilinear function and let $(a_1, \cdots, a_n)$ be such that $\nabla f(a_1, \cdots, a_n) = 0$. Then there are constants $d_S$, with $S \subseteq \{1, 2, \cdots, n\}$, such that $f$ may be rewritten as:

$$f(x_1, \cdots, x_n) = \sum_{S \subseteq \{1,2,\cdots,n\}} d_S \cdot \prod_{i \in S} (x_i - a_i) .$$

where $d_\emptyset = f(a_1, \cdots, a_n)$ (*c.f.* the Taylor series in a neighborhood of $(a_1, \cdots, a_n)$). We have the following cases:

1. Let $T = \{i_1, i_2, \cdots, i_\ell\}$ be a minimal set with $\ell > 0$ such that $d_T \neq 0$. We demonstrate that $(a_1, \cdots, a_n)$ cannot be a local minimum or a local maximum. We observe that $T$ cannot be a singleton because $\nabla f(a_1, \cdots, a_n) = 0$.

   There are two subcases: $\ell$ odd and $\ell$ even:

   – when $\ell$ is odd, consider $h_{(a_1,\cdots,a_n)} : \mathbb{R} \to \mathbb{R}^n$ defined as follows:

   $$(h_{(a_1,\cdots,a_n)}(t))_j = \begin{cases} t + a_j & \text{if } j \in T \\ a_j & \text{otherwise} \end{cases}$$

   Then we have $f(h_{(a_1,\cdots,a_n)}(t)) = f(a) + d_S \cdot t^\ell$ and $f(h_{(a_1,\cdots,a_n)}(-t)) = f(a) - d_S \cdot t^\ell$. Hence, for $t > 0$:

   $$(f(h_{(a_1,\cdots,a_n)}(t)) - f(a_1, \cdots, a_n))(f(h_{(a_1,\cdots,a_n)}(-t)) - f(a_1, \cdots, a_n)) < 0\ .$$

   Since, for every $\varepsilon > 0$, $h_{(a_1,\cdots,a_n)}(]-\varepsilon, \varepsilon[)$ is contained in an hypercube $H = \prod_{i=1}^n ]a_i - \varepsilon, a_i + \varepsilon[$ of $\mathbb{R}^n$, then there are two points $\widetilde{x}, \widetilde{y} \in H$ such that $(f(\widetilde{x}) - f(a_1, \cdots, a_n))(f(\widetilde{y}) - f(a_1, \cdots, a_n)) < 0$ and then one can conclude that the point $(a_1, \cdots, a_n)$ cannot be a local maximum or a local mimimum.

   – when $\ell$ is even, consider $h^-_{(a_1,\cdots,a_n)} : \mathbb{R} \to \mathbb{R}^n$ defined as follows

   $$(h^-_{(a_1,\cdots,a_n)}(t))_j = \begin{cases} a_j - t & \text{if } j = i_1; \\ a_j + t & \text{if } j \in T \setminus \{i_1\}; \\ a_j & \text{if } j \notin T. \end{cases}$$

   Then we have $f(h^-_{(a_1,\cdots,a_n)}(t)) = f(a) - d_S \cdot t^\ell$. Hence, for $t > 0$:

   $$(f(h_{(a_1,\cdots,a_n)}(t)) - f(a))(f(h^-_{(a_1,\cdots,a_n)}(t)) - f(a)) < 0$$

   and, as before, it is possible to conclude that $(a_1, \cdots, a_n)$ cannot be a point of local maximum or local mimimum.

2. $S_\ell = \emptyset$ (that is $\ell = 0$) and $d_{S_\ell} \neq 0$ and every other $S$ is such that $d_S = 0$. Then $f(x_1, \cdots, x_n) = d_\emptyset$, that is $f$ is constant.

□

An immediate consequence of Theorem 2.2 is the following.

**Corollary 2.3.** *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a multilinear function. The least and upper bounds of $f$ in the hypercube $H = [a_1, b_1] \times [a_2, b_2] \times \cdots \times [a_n, b_n]$ occur at the vertices of $H$.*

*Proof.* By Theorem 2.2, the least and upper bounds cannot be found inside $H$. Therefore they are on the borders with respect to some variable. Let it be $x_1$. This means that least and upper bounds are either in $f(a_1, x_2, \cdots, x_n)$ or in $f(b_1, x_2, \cdots, x_n)$, which are multilinear functions from $\mathbb{R}^{n-1}$ to $\mathbb{R}$. We reiterate the argument on these functions. The process terminates with constant functions. □

Theorem 2.2 and Corollary 2.3 may be generalized as follows. Let $f : \mathbb{R}^{n+m} \to \mathbb{R}$ be *almost-multilinear* if

$$f(x_1, \cdots, x_n, y_1, \cdots, y_m) = c_\emptyset(y_1, \cdots, y_m) + \sum_{S \subseteq \{1,2,\cdots,n\}} c_S(y_1, \cdots, y_m) \cdot \prod_{i \in S} x_i$$

and every $c_S : \mathbb{R}^m \to \mathbb{R}$ is a polynomial function. Interestingly, the bounds of almost-multilinear functions may be found by studying the non-multilinear parts.

**Proposition 2.4.** *Let $f : \mathbb{R}^{n+m} \to \mathbb{R}$ be almost-multilinear with linear variables $x_1$, $\cdots$, $x_n$. If $f$ has a local minimum or a local maximum at a point $(c_1, \cdots, c_n, c'_1, \cdots, c'_m)$ then $c_1, \cdots, c_n$ are bounds (either lower or upper) of $x_1$, $\cdots$, $x_n$.*

*Proof.* Let $(c_1, \cdots, c_n, c'_1, \cdots, c'_m)$ be such that $\nabla f(c_1, \cdots, c_n, c'_1, \cdots, c'_m) = 0$. Since the function $f(x_1, \cdots, x_n, c'_1, \cdots, c'_m)$ is multilinear then, by Corollary 2.3, its least and upper bounds may be found at the bounds of the linear variables, let them be $[a_1, b_1]$, $\cdots$, $[a_n, b_n]$. This means that, in order to compute the minimum and maximum values of $f(x_1, \cdots, x_n, y_1, \cdots, y_m)$, it is possible to reduce the calculus to the minimum and maximum of the sets of functions $\{f(d_1, \cdots, d_n, y_1, \cdots, y_m) \mid d_i \in \{a_i, b_i\},\ 1 \leq i \leq n\}$. $\square$

For example, let $f(x, y) = 2y^2 + xy^2$, with $x, y \in [-1, 1]$. Then $\nabla f(x, y) = (y^2, 2y(2x + 1))$, which is 0 when $y = 0$ (and every $x$). By Proposition 2.4, in order to compute the least and upper bound of $f$, we may reduce to computing the minimum and maximum of the functions $f(-1, y) = y^2$ and $f(1, y) = 3y^2$, that are 0 and 3, respectively (when $y \in [-1, 1]$).

## 3 Interval arithmetics for multilinear functions

We apply the results of the previous section to design a new algorithm for the interval analysis of expressions. In this section we focus on multilinear expressions (see below).

We use an infinite set of *identifiers*, ranged over by $x$, $y$, $z$; constants in $\mathbb{R}$ are ranged over by $c$, $d$, $\cdots$. *Polynomial expressions* $E$ are defined by the following grammar:

$$E \ ::= \quad c \ \mid \ x \ \mid \ -E \ \mid \ E + E \ \mid \ E - E \ \mid \ E * E$$

Let $\texttt{id}(E)$ be the set of identifiers in $E$. It is evident that an expression $E$ represents a polynomial in $\texttt{id}(E)$ variables. Therefore it is a functions from $\mathbb{R}^n$ to $\mathbb{R}$, where $n$ is the cardinality of $\texttt{id}(E)$. An expression is called *multilinear* if the corresponding polynomial is multilinear.

The algorithm `multilinearExp_range` in Table 1 computes the bounds of multilinear expressions by taking the minimum and maximum values of the expression when instantiated with the least and upper bounds of variables. More precisely, the algorithm gets (the syntax tree of) an expression $E$, an array of intervals, *i.e.* pairs $(a, b)$, and a natural number specifying the number of free variables of $E$. It is assumed that variables are totally ordered, *i.e.* $x_1$, $x_2$, $x_3$, $\cdots$, and $E$ contains the first $n$ variables. So, the interval of $x_i$ is defined

```
1: multilinearExp_range(E,Bounds[],n)
2:   for (1<= i <= n) do $a_i$ := proj1(Bounds[i]);
3:   m := E$\{{}^{a_1,\cdots,a_n}/_{x_1,\cdots,x_n}\}$ ;
4:   M := m ;
5:   for $y_1 \in$ { proj1(Bounds[1], proj2(Bounds[2] }, ...,
              $y_n \in$ { proj1(Bounds[n], proj2(Bounds[n] } do
6:       t := $E\{{}^{y_1,\cdots,y_n}/_{x_1,\cdots,x_n}\}$ ;
7:       m := min(m, t) ;
8:       M := max(M, t) ;
9:   return([m,M])
```

Table 1: The algorithm `multilinearExp_range`.

in `Bounds[i]` and we get the lower-bound and the upper-bound by means of `proj1` and `proj2`, respectively.

The computational complexity of `multilinearExp_range` is determined as follows:

- statements at lines 2, 3 and 4 do not play any relevant role;
- the `for` statement at line 5 has $2^n$ iterations;
- at every iteration, the cost of line 6 is computed as follows:
  - every monomial with $k$ variables has $k$ products (including the constant) and, in the worst case, there are $\binom{n}{k}$ of such monomials;
  - the total number of operations is

    $$\sum_{k=1}^{n} \binom{n}{k} \cdot k \quad = \quad n \cdot 2^{n-1}$$

  Therefore line 6 costs $O(n \cdot 2^{n-1})$.
- the overall complexity of `multilinearExp_range` is $O(2^n \cdot n \cdot 2^{n-1}) = O(n \cdot 2^{2n})$.

We observe that our algorithm has a better computational complexity than the one designed by Grigoriev and Vorobjov – that has a cost $O(d^{n^2})$. (Actually our algorithm is better when $d > 1$. When $d = 1$ we use the standard interval analysis, which returns the precise range.) We also observe that, while an $O(n \cdot 2^{2n})$ algorithm is prohibitive, in general, it is more affordable in the case of expressions occurring in programs that very rarely retain more than 4 variables. Last, we remind that interval arithmetics of multilinear expressions is computed statically for correctness purposes. In this context it is reasonable to pay more for an accurate analysis.

## 4 The general case

The technique of Section 3 may be extended to generic, polynomial expressions. Since Theorem 2.2 cannot be generalized to non-multilinear polynomials, and

```
 1: Exp_range(E,Bounds[],n)
 2:     (x,i) := nonlinearvar(E) ;
 3:     while (x $\neq$ $) do
 4:         (k,h) := getexponents(E,x) ;
 5:         if (h>1) then
 6:           E := replace(E,$x^h$, $x_{n+1}$) ; r:= h
 7:         else E := replace(E,$x^k$, $x_{n+1}$) ; r:= k;
 8:         Bounds[n+1] := intv(x,r,Bounds[i]);
 9:         n := n+1;
10:         (x,i) := nonlinearvar(E) ;
11:     return(multilinearExp_range(E,Bounds[],n))
```

Table 2: The algorithm `Exp_range`.

Proposition 2.4 does not help very much from the algorithmic point of view, we decided to define a "reduction" technique. That is, we reduce generic expressions to multilinear ones and compute the intervals of the latters. The ambition is to return more precise intervals than the standard interval arithmetics or other techniques.

As for `multilinearExp_range`, we assume that variables are totally ordered and $E$ contains the first $n$ variables, called $x_1$, $x_2$, $\cdots$, $x_n$. We also assume the presence of the following identifiers and functions:

– `$` is a dummy identifier, different from any other occurring in expressions;

– `nonlinearvar(E)` returns either a pair `($,-1)`, if the expression `E` is multilinear, or a pair `(x,i)`, where `x` is the first nonlinear variable (in the total ordering) occurring in `E` and `i` is its ordinal;

– `getexponents(E,x)` returns a pair `(k,h)` of naturals, with $\mathtt{k} > \mathtt{h}$, where `k` is the greatest exponent of `x` in `E` and `h` is the exponent of an occurrence of `x` that is immediately smaller than `k`. In case all the occurrences of `x` have exponent `k` then $\mathtt{h} = 0$.

– `intv(x,r,(a,b))` returns the lower-bound and upper-bound of the expression $\mathtt{x}^{\mathtt{r}}$, when $\mathtt{x} \in [\mathtt{a}, \mathtt{b}]$.

– `replace(E,`$\mathtt{x}^{\mathtt{h}}$`, z)` replaces the occurrences of $\mathtt{x}^{\mathtt{h}+\mathtt{h}'}$ in `E` with $\mathtt{z}\mathtt{x}^{\mathtt{h}'}$.

The algorithm `Exp_range` is defined in Table 2. `Exp_range` takes a nonlinear variable (line 2). In case no variable is found, then the bounds of the expressions are computed with `multilinearExp_range` (line 11). Otherwise, let the variable be `x`. The algorithm grabs the maximum exponent `k` of `x` the exponent `h`, with $\mathtt{k} > \mathtt{h}$, such that the other occurrences of `x` have either exponent `k` or exponent lesser or equal to `h` (line 4). When $\mathtt{h} > 1$, all the powers $\mathtt{x}^{\mathtt{k}}$ and $\mathtt{x}^{\mathtt{h}}$ in $E$ are replaced by $\mathtt{x}_{n+1}\mathtt{x}^{\mathtt{k}-\mathtt{h}}$ and $\mathtt{x}_{n+1}$, respectively, where $\mathtt{x}_{n+1}$ is the first fresh variable (line 6), and the interval of $\mathtt{x}_{n+1}$ is computed in a precise way (line 8). When $\mathtt{h} = 1$ or $\mathtt{h} = 0$ the replacement only concerns the terms $x_i^{\mathtt{k}}$. The iteration at line 3 terminates when the expression becomes multilinear. For example, let $E = x^5 - x^3 z + xy - xz + z$, with $x \in [-1, 2]$, $y \in [1, 3]$, and $z \in [2, 3]$. Here, the unique nonlinear variable is $x$ and the two values returned by `getexponents` are 5 and 3. Therefore, the expression $E$ becomes $vx^2 - vz + xy - xz + z$ with $v \in [-1, 8]$. In turn, the evaluation of this expression reduces to computing `multilinearExp_range` on $vu - vz + xy - xz + z$, with $u \in [0, 4]$.

We notice that the replacement of line 6 keeps the dependencies between the occurrences of $\mathtt{x^h}$, but definitely breaks those between (i) $\mathtt{x^h}$ and the exponent $\mathtt{k} - \mathtt{h}$ and (ii) those exponents lower than `h`. Such a rupture causes an over-approximation in the interval computation. In the above expression $E = x^5 - x^3z + xy - xz + z$, the dependency of $x$ between the first and second monomial is retained in $vx^2 - vz + xy - xz + z$ by means of the variable $v$. (A greedy strategy replacing $x^5$ with a variable would break this dependency.) This is a choice among several ones and it is not clear to us whether other choices may return better results. We leave this issue to future work.

Next, we analyze the computational complexity of `Exp_range`. Let $d$ be the greatest exponent in the input expression $E$ and let $n$ be the number of variables therein.

- In order to turn $E$ into an expression that is linear in $x$, we need, in the worst case, $d/2$ iteration of lines 4-10. These iterations introduce $d/2$ fresh variables. Similarly for every other nonlinear variable. In the worst case, the iterations are $n(d/2)$.
- At every iteration, the cost of `getexponents(E,x)`, $\mathtt{replace(E,x^h,x}_{n+1}\mathtt{)}$ and `nonlinearvar(E)` depend on the size of $E$, let it be $|E|$. In the worst case, this size grows linearly with respect to $E$ (because of the products $\mathtt{x}_{n+1}\mathtt{x^{k-h}}$ that are inserted). That is, the size $|E|$ at the beginning, $2|E|$ at the second iteration, $3|E|$ at the third one, and so on.
- Therefore, the cost of lines 4-10 is $O(|E| \cdot nd(nd + 2))$.
- Since the cost of invoking `multilinearExp_range` at line 11 with an expression of $n + n(d/2)$ variables is $O(2^{2n\cdot(1+d/2)})$, we obtain a complexity $O(|E| \cdot nd(nd + 2) + 2^{2n\cdot(1+d/2)})$, which is equal to $O(2^{2n\cdot(1+d/2)})$.

We conclude our analysis by comparing the outputs of `Exp_range` with other techniques. We consider the standard interval analysis and Miné's symbolic technique [5].

The expression we consider is $x^5 - x^3 * z + x * y - x * z + z$ with $x \in [-1, 2]$, $y \in [0, 2]$, and $z \in [0, 2]$. The precise interval of this expression is $[-3, 36]$. The following table sums up the results:

| **algorithm** | **output** |
|---|---|
| Interval Analysis | $[-26, 42]$ |
| Miné's technique | $[-21, 42]$ |
| `Exp_range` | $[-18, 36]$ |

If the intervals of variables are $x \in [0, 1]$, $y \in [0, 2]$, and $z \in [0, 3]$ then the precise interval of the above expression is $[-2, 3]$. The following table sums up the results:

| **algorithm** | **output** |
|---|---|
| Interval Analysis | $[-6, 6]$ |
| Miné's technique | $[-5, 4]$ |
| `Exp_range` | $[-2, 3]$ |

We notice that (in these cases) our techniques gives a better precision than the other ones. While this is reasonable because `Exp_range` has a higher computational cost than interval analysis or Minè's technique, it is unclear, as we said, whether this is always the case or not.

## 5 Conclusions

A new algorithm for the interval analysis of polynomial expressions has been proposed and studied. The algorithm is precise when the expressions ate multilinear. A preliminary assessment of the algorithm with respect to other techniques has begun and a thorough study is planned.

Our algorithm has been already successfully used for the interval analysis of codes of control switchboards, where every expression turns out to be multilinear. In facts, non-multilinear expressions are quite infrequent in programs. That is, the message conveyed by this paper is that interval analysis of expressions may be carried out without lost of precision in almost all the cases.

## References

[1] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. The astreé analyzer. In S. Sagiv, editor, *ESOP*, volume 3444 of *Lecture Notes in Computer Science*, pages 21–30. Springer, 2005.

[2] A. Gaganov. Computational complexity of the range of the polynomial in several variables. *Cybernetics and Systems Analysis*, 21(4):418–421, July 1985.

[3] D. Grigoriev and N. Vorobjov. Solving systems of polynomial inequalities in subexponential time. *J. Symb. Comput.*, 5(1/2):37–64, 1988.

[4] E. Hansen. Generalized interval arithmetic. In *Interval Mathematics*, volume 29 of *Lecture Notes in Computer Science*, pages 7–18. Springer Berlin / Heidelberg, 1975.

[5] A. Miné. Symbolic methods to enhance the precision of numerical abstract domains. In *Proc. of the 7th Int. Conf. on Verification, Model Checking, and Abstract Interpretation (VMCAI'06)*, volume 3855 of *Lecture Notes in Computer Science*, pages 348–363, Charleston, South Carolina, USA, January 2006. Springer.

[6] A. L. V. Kreinovich and J. Rohn. *Scientific Computing and Validated Numerics*, chapter Computational Complexity of Interval Algebraic Problems: Some Are Feasible and Some Are Computationally Intractable - A Survey, pages 293–306. Akademie Verlag, 1996.