How to Check, While Performing Straightforward Interval Computations, Whether the Resulting Function is Continuous, Everywhere Defined, etc.: Towards Foundations of Decorations

Vladik Kreinovich
Department of Computer Science
University of Texas at El Paso
500 W. University
El Paso, TX 79968, USA
vladik@utep.edu

Abstract

One of the main problems of interval computations is, given a function $f(x_1, \ldots, x_n)$ and n intervals, to find the range

$$\mathbf{y} = f(\mathbf{x}_1, \ldots, \mathbf{x}_n) = \{f(x_1, \ldots, x_n) : x_1 \in \mathbf{x}_1, \ldots, x_n \in \mathbf{x}_n\}.$$

One method of estimating this range is *straightforward interval computations*, in which we use the fact that a compiler represents the algorithm for computing $f(x_1, \ldots, x_n)$ as a sequence of elementary operations (arithmetic operations and elementary functions), and replace each elementary operation with numbers by the corresponding operation of interval arithmetic. In this case, on each computation step, we get an interval. It is known that the interval $\mathbf{Y}$ obtained on the final step is an *enclosure*, i.e., it is contains the desired range $\mathbf{y}$. This result is sometimes called the *main theorem of interval computations.*

By itself, the straightforward interval computations method is not a very good one: the resulting enclosure is often too wide. There exist much more efficient techniques (such as the mean value form). However, straightforward interval computation is important, since most efficient techniques for computing the desired range $\mathbf{y}$ use straightforward interval computations on intermediate steps.

Many implementations of interval techniques include, on each step, not only the interval, but also one or several bits (called *decorations*) that describe whether the resulting function is continuous, everywhere defined, etc. John Pryce has shown that the main theorem of interval computations can be extended to such *decorated intervals*. The purpose of this paper is to reformulate this result (and its proof) so as to make it as clear as possible for the most general readers interested in interval computations.

1 Interval Computations and Straightforward Interval Computations: Reminder

Need for interval computations. In many practical situations, we know the relation $y = f(x_1, \ldots, x_n)$ between quantities $x_1, \ldots, x_n$ and a quantity y. For example, Ohm's law says that the voltage y is equal to the product of the current x_1 and the resistance x_2. In this example, the dependence $f(x_1, \ldots, x_n)$ is very simple. In other cases, we have a very complex algorithm for computing $f(x_1, \ldots, x_n)$ – e.g., in geosciences, we have algorithms that come from solving the corresponding non-linear partial differential equations.

When we know the exact values of x_i, then we can simply use the known relation to estimate the value of y. In practice, our information about each quantity x_i comes from measurement, and measurement is never absolutely accurate: the measurement result $\widetilde{x}_i$ is, in general, different from the actual (unknown) value x_i of this quantity. In science and engineering practice, it is often assumed that we know the probability distribution of measurement errors $\Delta x_i \stackrel{\text{def}}{=} \widetilde{x}_i - x_i$. However, in many practical situations, we only know the upper bound Δ_i on the (absolute value of) the measurement error: $|\Delta x_i| \le \Delta_i$ [3]. In such situations, after we perform the measurement of the i-th quantity, the only information that we have about the actual value x_i is that this value belongs to the interval

$$\mathbf{x}_i = [\underline{x}_i, \overline{x}_i] \stackrel{\text{def}}{=} [\widetilde{x}_i - \Delta_i, \widetilde{x}_i + \Delta_i].$$

Different values $x_i \in \mathbf{x}_i$ lead, in general, to different values of $y = f(x_1, \ldots, x_n)$. It is therefore desirable to find the *range* of possible values of y:

$$\mathbf{y} = f(\mathbf{x}_1, \ldots, \mathbf{x}_n) \stackrel{\text{def}}{=} \{f(x_1, \ldots, x_n) : x_1 \in \mathbf{x}_1, \ldots, x_n \in \mathbf{x}_n\}.$$

In this problem, inputs are intervals; because of this, the problem of estimating the range $\mathbf{y}$ is called *interval computations*; see, e.g., [1] are references therein.

How algorithms are performed inside a computer: reminder. Most methods for solving the above interval computations problem are based on the following observation. In a computer, several "elementary" operations are directly implemented: usually, the basic arithmetic operations (addition, subtraction, multiplication, division) and several elementary functions such as square root, logarithms, trigonometric functions, etc.

Every algorithm that we write for a computer is translated (by a compiler) into a sequence of elementary steps, on each of which the computer applies one of these elementary operations. For example, when we write an expression $x_1 \cdot (1.0 - x_1)$, the computer first computes the constant $x_2 = 1.0$, then computes an auxiliary value $x_3 = 1.0 - x_1$, and then multiplies x_1 by x_3 resulting in the desired value $x_4 = x_1 \cdot x_3$.

In general, we start with the inputs $x_1, \ldots, x_n$, and with whatever constants we need in our computations. We will denote the number of these constants by

c and the constants themselves by $x_{n+1}, \ldots, x_{n+c}$. Then, we compute intermediate results – which will be denoted by $x_{n+c+1}, x_{n+2}, \ldots, x_N$ – until we reach the final result x_N. Each intermediate result x_j is obtained by applying one of the functions g from the list G of elementary functions to some previously computed values, i.e., as $x_j = g(x_{i_1}, \ldots, x_{i_k})$ for some $i_1, \ldots, i_k < j$. Thus, by an *algorithm*, we can understand a sequence of such statements. This can be formally described as follows:

Definition 1. *Let G be a finite set whole elements are functions $g : \mathbb{R}^k \to \mathbb{R}$ (these functions may be defined only for some k-tuples of real numbers). For each function $g \in G$, the corresponding value k will be called its* arity *and denoted by* $\mathrm{ar}(g)$.

- *By an* elementary computation step *s, we mean a pair consisting of a function $g \in G$ and a tuple $\langle i_1, \ldots, i_k \rangle$. consisting of $k = \mathrm{ar}(g)$ positive integers; this step will also be denoted by $x = g(x_{i_1}, \ldots, x_{i_k})$.*
- *By an* algorithm *f, we mean a tuple consisting of:*
  - *a positive integer n (called* number of inputs*),*
  - *a finite list of real numbers called* constants*; the number of these constants will be denoted by c, and the numbers themselves by $x_{n+1}, \ldots, x_{n+c}$; and*
  - *a sequence of elementary computation steps $s_{n+c+1}, s_{n+2}, \ldots, s_N$ for which, for every step $s_j = \langle g_j, \langle i_{j,1}, \ldots, i_{j,k_j} \rangle \rangle$, all the integers $i_{j,1}, \ldots, i_{j,k_j}$ are smaller than j; the j-th step will be also denoted by $x_j = g_j(x_{i_{j,1}}, \ldots, x_{i_{j,k_j}})$.*

Let $f = \langle n, x_{n+1}, \ldots, x_{n+c}, s_{n+1}, \ldots, s_N \rangle$ be an algorithm with n inputs, and let $x_1, \ldots, x_n$ be n real numbers. Then, for each $j \le N$, by the j-th intermediate result *f_j, we mean a value that is described by the following inductive definition:*

- *for $j \le n + c$, we define $f_j = x_j$;*
- *once the values $f_1, \ldots, f_{j-1}$ are defined and $j > n+c$, we take the j-th computation step $x_j = g_j(x_{i_{j,1}}, \ldots, x_{i_{j,k_j}})$ and define f_j as $g_j(f_{i_{j,1}}, \ldots, f_{i_{j,k_j}})$.*

The last (N-th) intermediate result is called the result *of applying the algorithm f to the values $x_1, \ldots, x_n$; it will be denoted by $f(x_1, \ldots, x_n)$.*

Comment. Of course, if a function g_j is not defined for the values $f_{i_{j,1}}$, …, $f_{i_{j,k_j}}$, then the value f_j is not defined – and thus, the following values are not defined as well. For example, when $x_1 = 1.0$ and $x_2 = 0.0$, the value $x_3 = g(x_1, x_2) = x_1/x_2 = 1.0/0.0$ is undefined. It is worth mentioning that in the IEEE arithmetic implemented in many computers, the $1.0/0.0$ is explicitly defined as ∞; however, we decided not to consider infinities in this text, since the inclusion of infinite values makes all arithmetic operations much more complex – and besides, some operations results like $\infty - \infty$ are still undefined.

In the above example of computing $f(x_1) = x_1 \cdot (1.0 - x_1)$, the set G of elementary operations contains two 2-ary functions: subtraction $-$ and multiplication $\cdot$. Here, $f = \langle 1, 1.0, s_3, s_4 \rangle$, with a single input $n = 1$, a single constant $x_2 = 1.0$, and two elementary computation steps:

- a step s_3 of the form $x_3 = g(x_2, x_1) = x_2 - x_1$, with $g_3 = -$, $i_{3,1} = 2$, and $i_{3,2} = 1$; and
- a step s_4 of the form $x_3 = g(x_1, x_3) = x_1 \cdot x_3$, with $g_4 = \cdot$, $i_{4,1} = 1$, and $i_{4,2} = 3$.

In this example, the first intermediate result is simply the input $f_1 = x_1$, the second is the constant $f_2 = x_2 = 1.0$, the third result is $f_3 = f_2 - f_1 = 1 - x_1$, and the final result is $f_4 = f_1 \cdot f_3 = x_1 \cdot (1.0 - x_1)$.

Straightforward interval computations. Let us assume that for each elementary operation $g(x_1, \ldots, x_k) \in G$, we have an *interval enclosure*, i.e., a function $\mathbf{G}$ that transforms each tuple of k intervals $\mathbf{x}_1, \ldots, \mathbf{x}_k$, into an interval $\mathbf{G}(\mathbf{x}_1, \ldots, \mathbf{x}_k)$ that is an enclosure for the range of g on these intervals. For elementary functions, such enclosure are easy to produce – usually, we can even compute the exact range.

For example, for elementary arithmetic operations, the corresponding ranges are described by the following formulas of *interval arithmetic*. For $g(x_1, x_1) = x_1 + x_2$, we have

$$g([\underline{x}_1, \overline{x}_1], [\underline{x}_2, \overline{x}_2]) = [\underline{x}_1, \overline{x}_1] + [\underline{x}_2, \overline{x}_2] = [\underline{x}_1 + \underline{x}_2, \overline{x}_1 + \overline{x}_2].$$

Similarly, for other arithmetic operations, we have

$$[\underline{x}_1, \overline{x}_1] - [\underline{x}_2, \overline{x}_2] = [\underline{x}_1 - \overline{x}_2, \overline{x}_1 - \underline{x}_2];$$

$$[\underline{x}_1, \overline{x}_1] \cdot [\underline{x}_2, \overline{x}_2] =$$

$$[\min(\underline{x}_1 \cdot \underline{x}_2, \underline{x}_1 \cdot \overline{x}_2, \overline{x}_1 \cdot \underline{x}_2, \overline{x}_1 \cdot \overline{x}_2), \max(\underline{x}_1 \cdot \underline{x}_2, \underline{x}_1 \cdot \overline{x}_2, \overline{x}_1 \cdot \underline{x}_2, \overline{x}_1 \cdot \overline{x}_2)];$$

$$[\underline{x}_1, \overline{x}_1] / [\underline{x}_2, \overline{x}_2] = [\underline{x}_1, \overline{x}_1] \cdot \frac{1}{[\underline{x}_2, \overline{x}_2]},$$

where

$$\frac{1}{[\underline{x}_2, \overline{x}_2]} = [1/\overline{x}_2, 1/\underline{x}_2] \text{ if } 0 \notin [\underline{x}_2, \overline{x}_2].$$

For elementary functions, it is also usually possible to compute either the exact range, or at least a reasonable enclosure for this range; see, e.g., [1].

Then, to find the enclosure for the range $\mathbf{y} = f(\mathbf{x}_1, \ldots, \mathbf{x}_n)$, we we replace each elementary operation $x_j = g_j(x_{i_{j,1}}, \ldots, x_{i_{j,k_j}})$ from the algorithm f with the corresponding interval operation $\mathbf{x}_j = \mathbf{G}_j(\mathbf{x}_{i_{j,1}}, \ldots, \mathbf{x}_{i_{j,k_j}})$. The "main theorem of interval computations" (see, e.g., [1]) states that for each intermediate result, the interval $\mathbf{x}_j$ computed by this algorithm encloses the actual range $f_j(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ of the corresponding intermediate function $f_j(x_1, \ldots, x_n)$.

In particular, for the last step, we conclude that the interval $\mathbf{x}_N$ computed on this step is an enclosure for the desired range $\mathbf{y} = f(\mathbf{x}_1, \ldots, \mathbf{x}_n)$.

Comment. The resulting enclosure is often too wide. There exist much more efficient techniques (such as the mean value form); see, e.g., [1]. However, straightforward interval computation is important, since most efficient techniques for computing the desired range $\mathbf{y}$ use straightforward interval computations on intermediate steps.

Since our objective is to extend the "main theorem of interval computations", let us provide exact definitions and the corresponding proof.

Definition 2. *Let $\mathbb{IR}$ denote the set of all intervals (including an empty set). An everywhere defined function $\mathbf{G} : \mathbb{IR}^k \to \mathbb{IR}$ is called an* interval extension *of a (maybe partially defined) function $g : \mathbb{R}^k \to \mathbb{R}$ if for all possible intervals $\mathbf{x}_1, \ldots, \mathbf{x}_k$ and for all values $x_1 \in \mathbf{x}_1$, $\ldots$, $x_k \in \mathbf{x}_k$ for which the value $g(x_1, \ldots, x_k$ is defined, we have $g(x_1, \ldots, x_k) \in \mathbf{G}(\mathbf{x}_1, \ldots, \mathbf{x}_k)$.*

Comment. This definition is equivalent to requiring that the result $\mathbf{G}(\mathbf{x}_1, \ldots, \mathbf{x}_k)$ of the interval extension always contain the range $g(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ of the function g.

Definition 3. *Let G be a finite set of functions $g : \mathbb{R}^k \to \mathbb{R}$; for each of these functions g, we have an interval extension $\mathbf{G} : \mathbb{IR}^k \to \mathbb{IR}$. Let $f = \langle n, x_{n+1}, \ldots, x_{n+c}, s_{n+1}, \ldots, s_N \rangle$ be an algorithm with n inputs, and let $\mathbf{x}_1, \ldots, \mathbf{x}_n$ be n real numbers. Then, for each $j \le N$, by the j-th* intermediate result $\mathbf{f}_j$*, we mean a value that is described by the following inductive definition:*

- *for $j \le n$, we define $\mathbf{f}_j = \mathbf{x}_j$;*
- *for $n + 1 \le j \le n + c$, we define $\mathbf{f}_j = [x_j, x_j]$;*
- *once the intervals $\mathbf{f}_1, \ldots, \mathbf{f}_{j-1}$ are defined and $j > n + c$, we take the j-th computation step $x_j = g_j(x_{i_{j,1}}, \ldots, x_{i_{j,k_j}})$ and define $\mathbf{f}_j$ as $\mathbf{G}_j(\mathbf{f}_{i_{j,1}}, \ldots, \mathbf{f}_{i_{j,k_j}})$.*

The last (N-th) intermediate result is called the result *of applying straightforward interval computations to the problem of estimating the range $f(\mathbf{x}_1, \ldots, \mathbf{x}_n)$; this result will be denoted by $\mathbf{F}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$.*

Example. Let us illustrate this definition on the example of computing the range of the function $f(x_1) = x_1 \cdot (1.0 - x_1)$ on the interval $\mathbf{x}_1 = [0.0, 1.0]$. In this case, for both elementary arithmetic operations $g = -$ and $g = \cdot$, the corresponding interval extension is described above. Here, $f = \langle 1, 1.0, s_3, s_4 \rangle$, with a single input $n = 1$, a single constant $x_2 = 1.0$, and two elementary computation steps:

- a step s_3 of the form $x_3 = g(x_2, x_1) = x_2 - x_1$, with $g_3 = -$, $i_{3,1} = 2$, and $i_{3,2} = 1$; and

- a step s_4 of the form $x_3 = g(x_1, x_3) = x_1 \cdot x_3$, with $g_4 = \cdot$, $i_{4,1} = 1$, and $i_{4,2} = 3$.

In this example:

- the first intermediate result is simply the input $\mathbf{f}_1 = \mathbf{x}_1 = [0.0, 1.0]$,
- the second is the constant $\mathbf{f}_2 = [x_2, x_2] = [1.0, 1.0]$,
- the third result is

$$\mathbf{f}_3 = \mathbf{f}_2 - \mathbf{f}_1 = [1.0.1.0] - [0.0, 1.0] = [1.0 - 1.0, 1.0 - 0.0] = [0.0, 1.0];$$

- the final result is

$$\mathbf{f}_4 = \mathbf{f}_1 \cdot \mathbf{f}_3 = [0.0, 1.0] \cdot [0.0, 1.0] =$$

$$[\min(0.0{\cdot}0.0, 0.0{\cdot}1.0, 1.0{\cdot}0.0, 1.0{\cdot}1.0), \max(0.0{\cdot}0.0, 0.0{\cdot}1.0, 1.0{\cdot}0.0, 1.0{\cdot}1.0)] =$$

$$[\min(0.0, 0.0, 0.0, 1.0), \max(0.0, 0.0, 0.0, 1.0)] = [0.0, 1.0].$$

Let us compare this result with the actual range., According to calculus, the minimum and the maximum of a differentiable function on an interval are attained either at one of the endpoints, or an internal point – in which case $\dfrac{df}{dx_1} = 0$. Thus, to find the minimum and the maximum of a function on a given interval, it is sufficient to compute its value on both endpoints and on all the internal points where the derivative is 0: the smallest of these values is the desired minimum, and the largest of these values is the desired maximum.

For the function $f(x_1) = x_1 \cdot (1.0 - x_1)$, the derivative is $\dfrac{df}{dx_1} = 1 - 2x_1$, so it is equal to 0 when $x_1 = 0.5$. Thus, to find the actual range, we need to compute three values: $f(0.0) = 0.0$, $f(1.0) = 0.0$, and $f(0.5) = 0.25$. The smallest of these three values is 0.0, the largest is 0.25, so the actual range is $\mathbf{y} = [0.0, 0.25]$. We see that the result $[0.0, 1.0]$ of straightforward interval computations encloses this range (and also that it has "excess width" in comparison with the actual range). Let us describe the proof that this is always the case.

Proposition 1. *For every algorithm f with n inputs and for every tuple $\mathbf{x}_1$, ..., $\mathbf{x}_n$, the result $\mathbf{Y}$ of applying straightforward interval computations encloses the range $\mathbf{y} = f(\mathbf{x}_1, \ldots, \mathbf{x}_n)$.*

Proof. To prove this proposition, let us first prove the following two lemmas:

Lemma 1. *Let $g : \mathbb{R}^k \to \mathbb{R}$ be a function, let $\mathbf{G} : \mathbb{IR}^k \to \mathbb{IR}$ its interval extension, let $\mathbf{X}_1$, ..., $\mathbf{X}_n$ be intervals, and let $x_1 \in \mathbf{X}_1$, ..., $x_n \in \mathbf{X}_n$ be numbers. Then,*

$$g(x_1, \ldots, x_n) \in \mathbf{G}(\mathbf{X}_1, \ldots, \mathbf{X}_n).$$

Proof of Lemma 1. By definition of the range, the fact that $x_1 \in \mathbf{X}_1$, ..., and $x_n \in \mathbf{X}_n$ implies that value $g(x_1, \ldots, x_n)$ belongs to the range:

$$g(x_1, \ldots, x_n) \in g(\mathbf{X}_1, \ldots, \mathbf{X}_n).$$

By definition of an interval extension, we conclude that

$$g(\mathbf{X}_1, \ldots, \mathbf{X}_n) \subseteq \mathbf{G}(\mathbf{X}_1, \ldots, \mathbf{X}_n),$$

i.e., that the range is a subset of the result of applying the interval extension. An element of a subset is also an element of the set itself. The Lemma is proven.

Lemma 2. *For every algorithm f with n inputs, for every tuple $\mathbf{x}_1$, ..., $\mathbf{x}_n$, and for every j, the j-th intermediate result $\mathbf{f}_j$ of applying straightforward interval computations encloses the range $\mathbf{y}_j = f_j(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ of the function $f_j(x_1, \ldots, x_n)$ describing the j-th intermediate result.*

Proof of Lemma 2. This lemma can be easily proven by induction. The base case is $j \le n + c$, when, by definition of the j-result, it coincides with the desired range.

The induction case is $j > n + c$. In this case, $x_j = g_j(x_{i_{j,1}}, \ldots, x_{i_{j,k_j}})$ and $\mathbf{f}_j = \mathbf{G}_j(\mathbf{f}_{i_{j,1}}, \ldots, \mathbf{f}_{i_{j,k_j}})$. By induction, we assume that the lemma holds for all previous intermediate results $i_{j,\ell} < j$. In particular, this means that for every ℓ, for all possible values $x_1 \in \mathbf{x}_1$, ..., $x_n \in \mathbf{x}_n$, we have:

$$f_{i_{j,\ell}}(x_1, \ldots, x_n) \in \mathbf{f}_{i_{j,\ell}}.$$

Thus, by Lemma 1, we have

$$f_j(x_1, \ldots, x_n) = g_j(f_{i_{j,1}}, \ldots, f_{i_{j,k_j}}) \in \mathbf{G}_j(\mathbf{f}_{i_{j,1}}, \ldots, \mathbf{f}_{i_{j,k_j}}) = \mathbf{f}_j.$$

So, every possible value of f_j belongs to the interval $\mathbf{f}_j$. This means that the interval $\mathbf{f}_j$ encloses the range $\mathbf{y}_j = f_j(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ of the function $f_j(x_1, \ldots, x_n)$. The lemma is proven.

Proof of the Proposition itself. By definition, the result of the straightforward interval computations is the N-th intermediate result; thus, the proposition follows from Lemma 2 for $j = N$.

2 Checking Continuity etc.: Decorations

Need for checking. Some elementary functions are continuous everywhere they are defined: e.g., all arithmetic operations. Some elementary functions have points of discontinuity: e.g., a function $\lfloor x \rfloor$ (that returns the integer part of a given real number x):

- is equal to 1.0 for all the values x from the semi-closed interval $[1.0, 2.0)$,

- but for $x = 2.0$, its value "jumps" from 1.0 to 2.0.

Similarly, some elementary functions are everywhere defined, e.g, addition, substraction, and multiplication. On the other hand, division $g(x_1, x_2) = x_1/x_2$ is not defined when $x_2 = 0$.

It is therefore desirable to check whether the function $f(x_1, \ldots, x_n)$ computed by an algorithm is continuous and/or everywhere defined.

Idea of decorations. For this checking, some implementations of interval computations use the following idea: on each step of straightforward interval computations, we compute not only the interval, but also one or several bits (called *decorations*) that describe whether the resulting function is continuous, everywhere defined, etc. The corresponding tuple consisting of the interval and these decoration bits is called a *decorated interval*. The interval part is computed the same way as before.

What we do in this paper. John Pryce has shown [2] that the main theorem of interval computations can be extended to such decorated intervals. The purpose of this paper is to reformulate this result (and its proof) so as to make it as clear as possible for the most general readers interested in interval computations. Let us start with the case of continuity.

Definition 4. *Let G be a finite set of functions $g : \mathbb{R}^k \to \mathbb{R}$; for each of these functions g, we have:*

- *an interval extension $\mathbf{G} : \mathbb{IR}^k \to \mathbb{IR}$, and*
- *a function $\mathrm{cont}_g : \mathbb{IR}^k \to \{0, 1\}$ such that if $\mathrm{cont}_g(\mathbf{x}_1, \ldots, \mathbf{x}_k) = 1$ (= "true"), then the function $g(x_1, \ldots, x_k)$ is continuous at every point $(x_1, \ldots, x_k) \in \mathbf{x}_1 \times \ldots \times \mathbf{x}_k$.*

Let $f = \langle n, x_{n+1}, \ldots, x_{n+c}, s_{n+1}, \ldots, s_N \rangle$ be an algorithm with n inputs, and let $\mathbf{x}_1, \ldots, \mathbf{x}_n$ be n real numbers. Then, for each $j \le N$, by the j-th intermediate result *$\langle \mathbf{f}_j, c_j \rangle$, we mean a tuple consisting of an interval and a boolean value which is described by the following inductive definition:*

- *for $j \le n$, we define $\mathbf{f}_j = \mathbf{x}_j$ and $c_j = 1$;*
- *for $n + 1 \le j \le n + c$, we define $\mathbf{f}_j = [x_j, x_j]$ and $c_j = 1$;*
- *once the intervals $\mathbf{f}_1, \ldots, \mathbf{f}_{j-1}$ are defined and $j > n + c$, we take the j-th computation step $x_j = g_j(x_{i_{j,1}}, \ldots, x_{i_{j,k_j}})$ and define*

$$\mathbf{f}_j = \mathbf{G}_j(\mathbf{f}_{i_{j,1}}, \ldots, \mathbf{f}_{i_{j,k_j}});$$

$$c_j = \mathrm{cont}_{g_j}(\mathbf{f}_{i_{j,1}}, \ldots, \mathbf{f}_{i_{j,k_j}}) \,\&\, c_{i_{j,1}} \,\&\, \ldots \,\&\, c_{i_{j,k_j}}.$$

The last (N-th) intermediate result is called the result *of applying straightforward interval computations to the problem of estimating the range $f(\mathbf{x}_1, \ldots, \mathbf{x}_n)$; this result will be denoted by*

$$\langle \mathbf{F}(\mathbf{x}_1, \ldots, \mathbf{x}_n), \mathrm{cont}_f(\mathbf{x}_1, \ldots, \mathbf{x}_n) \rangle.$$

Comment. Since the intervals $\mathbf{f}_j$ are computed exactly the same way as before, we know that the resulting interval $\mathbf{F}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ is an enclosure. Thus, all we need to do is to describe the meaning of the additional bit:

Proposition 2. *For every algorithm f with n inputs and for every tuple $\mathbf{x}_1$, ..., $\mathbf{x}_n$, if the resulting bit $\mathrm{cont}_f(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ is true, then the function $f(x_1, \ldots, x_n)$ is continuous at every point from the box $\mathbf{x}_1 \times \ldots \times \mathbf{x}_n$ at which it is defined.*

Comment. In Proposition 1, we proved, in effect, that every possible value of $y = f(x_1, \ldots, x_n)$ is contained in the result $\mathbf{Y}$ of straightforward interval computations. We also had an example when the inverse is not true: there are some values from the result which are not possible values of $y = f(x_1, \ldots, x_n)$ for $x_i \in \mathbf{x}_i$.

Similarly here, the fact that the bit is false ($= 0$) does not necessarily mean that the function is discontinuous. For example, the function $f(x) = \lfloor x - x \rfloor$ defined on the interval $[0, 1]$ is constant ($= 0$) for all x and thus, continuous. On the other hand, when we perform straightforward interval computations with decorations, we get $x_2 = x_1 - x_1$, $x_3 = \lfloor x_2 \rfloor$, which leads to $\mathbf{f}_2 = [0, 1] - [0, 1] = [-1, 1]$. Since the integer part function is not everywhere continuous on the interval $[-1, 1]$, the above algorithm returns the false value of the continuity bit.

Proof of Proposition 2. Similarly to Proposition 1, we can prove this proposition by induction. For $j \le n + c$, all the functions are either projections (for $j \le n$ or constants – in both cases, they are continuous.

Let us assume that we have proved it for all intermediate values $< j$. Let us prove it for the j-th result $x_j = g_j(x_{i_{j,1}}, \ldots, x_{i_{j,k_j}})$. According to our procedure, the only case when the j-th continuity bit is true is when the bit $\mathrm{cont}_{g_j}(\mathbf{f}_{i_{j,1}}, \ldots, \mathbf{f}_{i_{j,k_j}})$ is true. In this case, by definition of cont_g, the function $g_j(x_{i_{j,1}}, \ldots, x_{i_{j,k_j}})$ is continuous for all possible values $x_{i_{j,1}} \in \mathbf{f}_{i_{j,1}}$, ..., $x_{i_{j,k_j}} \in \mathbf{f}_{i_{j,k_j}}$. Since we have proven, in Lemma 2, that for $x_i \in \mathbf{x}_i$, all possible values of $x_{i_{j,\ell}}$ belong to the set $\mathbf{f}_{i_{j,\ell}}$, we thus conclude that g_j is always continuous on the possible values. By induction assumption, we already know that the functions $f_{i_{j,\ell}}(x_1, \ldots, x_n)$ are continuous on the given box $\mathbf{x}_1 \times \ldots \times \mathbf{x}_n$. Since composition of continuous functions is continuous, the function

$$f_j(x_1, \ldots, x_n) = g_j(f_{i_{j,1}}(x_1, \ldots, x_n), \ldots, f_{i_{j,k_j}}(x_1, \ldots, x_n))$$

is also continuous on this box. The proposition is proven.

Application to connectedness. Let us describe a situation in which proving continuity may be important. In addition to knowing the interval $\mathbf{x}_i$ of possible values for each i, we may know some relation between the variables: e.g., we may know that $x_1 + x_2 \le 1.0$. In general, we may know the set $X \subseteq \mathbf{x}_1 \times \ldots \times \mathbf{x}_n$ of possible values of the tuple $(x_1, \ldots, x_n)$. In this case, we are interested in learning about the range $\{f(x_1, \ldots, x_n) : (x_1, \ldots, x_n) \in X\}$. Since $X \subseteq \mathbf{x}_1 \times \ldots \times \mathbf{x}_n$, this range is contained in the range $f(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ and thus, contained in the result $\mathbf{Y}$ of straightforward interval computations.

If the function $f(x_1, \ldots, x_n)$ is continuous, then we can provide more information about the desired range. For example, if the set X is *connected* (in the usual topological sense of this word, that it cannot be partitioned into two nonempty subsets with disjoint closures), then its range is also connected – and is, thus, an *interval*.

Beyond continuity. The only property of continuity that we used in the above proof is that the composition of continuous functions is continuous. Thus, similar results hold for every property that is preserved under composition: e.g., for the property of being differentiable, for the property of being everywhere defined on a given box, etc.

One important property is the property of being everywhere defined on a box. This property is often implicitly assumed. For example, in control, stability is often described in terms of inequalities on the eigenvalues of the corresponding matrices – with the implicit understanding that the formulas describing these eigenvalues are everywhere defined. In the simplified example, if we need to check that the condition $y \stackrel{\text{def}}{=} f(x) = \sqrt{x} \le 1.0$ holds for all $x \in [\underline{x}, \overline{x}]$, then a seemingly natural idea is to find the range of y

$$[\underline{y}, \overline{y}] = \{\sqrt{x} : x \in [\underline{x}, \overline{x}]\}$$

and then to check whether $\overline{y} \le 1.0$. This works if the function $f(x)$ is everywhere defined on the interval $[\underline{x}, \overline{x}]$. However, e.g., for $[\underline{x}, \overline{x}] = [-1, 1]$, the above-described range is $[0, 1]$, so $\overline{y} = 1.0 \le 1.0$. At first glance, it may seem that the desired condition is satisfied, but in reality, it is *not* satisfied – because for values $x \in [-1, 0] \subseteq [-1, 1]$, the function $f(x) = \sqrt{x}$ is not defined.

Because checking whether a function is everywhere defined is important, let us explicitly describe the corresponding result.

Definition 5. *Let G be a finite set of functions $g : \mathbb{R}^k \to \mathbb{R}$; for each of these functions g, we have:*

- *an interval extension $\mathbf{G} : \mathbb{IR}^k \to \mathbb{IR}$, and*
- *a function $\text{def}_g : \mathbb{IR}^k \to \{0, 1\}$ such that if $\text{def}_g(\mathbf{x}_1, \ldots, \mathbf{x}_k) = 1$ (= "true"), then the function $g(x_1, \ldots, x_k)$ is defined at every point $(x_1, \ldots, x_k) \in \mathbf{x}_1 \times \ldots \times \mathbf{x}_k$.*

Let $f = \langle n, x_{n+1}, \ldots, x_{n+c}, s_{n+1}, \ldots, s_N\rangle$ be an algorithm with n inputs, and let $\mathbf{x}_1, \ldots, \mathbf{x}_n$ be n real numbers. Then, for each $j \le N$, by the j-th intermediate result *$\langle \mathbf{f}_j, d_j\rangle$, we mean a tuple consisting of an interval and a boolean value which is described by the following inductive definition:*

- *for $j \le n$, we define $\mathbf{f}_j = \mathbf{x}_j$ and $d_j = 1$;*
- *for $n+1 \le j \le n+c$, we define $\mathbf{f}_j = [x_j, x_j]$ and $d_j = 1$;*
- *once the intervals $\mathbf{f}_1, \ldots, \mathbf{f}_{j-1}$ are defined and $j > n+c$, we take the j-th computation step $x_j = g_j(x_{i_{j,1}}, \ldots, x_{i_{j,k_j}})$ and define*

$$\mathbf{f}_j = \mathbf{G}_j(\mathbf{f}_{i_{j,1}}, \ldots, \mathbf{f}_{i_{j,k_j}});$$

$$d_j = \mathrm{def}_{g_j}(\mathbf{f}_{i_{j,1}}, \ldots, \mathbf{f}_{i_{j,k_j}}) \,\&\, d_{i_{j,1}} \,\&\, \ldots \,\&\, d_{i_{j,k_j}}.$$

The last (N-th) intermediate result is called the result *of applying straightforward interval computations to the problem of estimating the range $f(\mathbf{x}_1, \ldots, \mathbf{x}_n)$; this result will be denoted by*

$$\langle \mathbf{F}(\mathbf{x}_1, \ldots, \mathbf{x}_n), \mathrm{def}_f(\mathbf{x}_1, \ldots, \mathbf{x}_n)\rangle.$$

Proposition 3. *For every algorithm f with n inputs and for every tuple $\mathbf{x}_1, \ldots, \mathbf{x}_n$, if the resulting bit $\mathrm{def}_f(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ is true, then the function $f(x_1, \ldots, x_n)$ is defined everywhere on the box $\mathbf{x}_1 \times \ldots \times \mathbf{x}_n$.*

A similar result holds when we take into account that sometimes, functions are nowhere defined on boxes. For example, the square root is nowhere defined on the interval $[-2.0, -1.0]$.

Definition 6. *Let G be a finite set of functions $g : \mathbb{R}^k \to \mathbb{R}$; for each of these functions g, we have:*

- *an interval extension $\mathbf{G} : \mathbb{IR}^k \to \mathbb{IR}$, and*
- *a function $\mathrm{def}_g : \mathbb{IR}^k \to \{0, 1\}$ such that if $\mathrm{nowhere}_g(\mathbf{x}_1, \ldots, \mathbf{x}_k) = 1$ (= "true"), then the function $g(x_1, \ldots, x_k)$ is not defined at any point $(x_1, \ldots, x_k) \in \mathbf{x}_1 \times \ldots \times \mathbf{x}_k$.*

Let $f = \langle n, x_{n+1}, \ldots, x_{n+c}, s_{n+1}, \ldots, s_N\rangle$ be an algorithm with n inputs, and let $\mathbf{x}_1, \ldots, \mathbf{x}_n$ be n real numbers. Then, for each $j \le N$, by the j-th intermediate result *$\langle \mathbf{f}_j, n_j\rangle$, we mean a tuple consisting of an interval and a boolean value which is described by the following inductive definition:*

- *for $j \le n$, we define $\mathbf{f}_j = \mathbf{x}_j$ and $n_j = 1$;*
- *for $n+1 \le j \le n+c$, we define $\mathbf{f}_j = [x_j, x_j]$ and $n_j = 1$;*

- *once the intervals* $\mathbf{f}_1, \ldots, \mathbf{f}_{j-1}$ *are defined and* $j > n + c$*, we take the* j*-th computation step* $x_j = g_j(x_{i_{j,1}}, \ldots, x_{i_{j,k_j}})$ *and define*

$$\mathbf{f}_j = \mathbf{G}_j(\mathbf{f}_{i_{j,1}}, \ldots, \mathbf{f}_{i_{j,k_j}}); \quad n_j = \text{nowhere}_{g_j}(\mathbf{f}_{i_{j,1}}, \ldots, \mathbf{f}_{i_{j,k_j}}).$$

The last (N*-th) intermediate result is called the* result *of applying straightforward interval computations to the problem of estimating the range* $f(\mathbf{x}_1, \ldots, \mathbf{x}_n)$*; this result will be denoted by*

$$\langle \mathbf{F}(\mathbf{x}_1, \ldots, \mathbf{x}_n), \text{nowhere}_f(\mathbf{x}_1, \ldots, \mathbf{x}_n) \rangle.$$

Proposition 4. *For every algorithm* f *with* n *inputs and for every tuple* $\mathbf{x}_1$*,* $\ldots$*,* $\mathbf{x}_n$*, if the resulting bit* $\text{nowhere}_f(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ *is true, then the function* $f(x_1, \ldots, x_n)$ *is not defined anywhere on the box* $\mathbf{x}_1 \times \ldots \times \mathbf{x}_n$*.*

Comment. We assumed that computations of the intervals $\mathbf{f}_j$ are not affected by the computations of the corresponding bits. However, for nowhere-defined bits, it may be beneficial to take the bits into account when computing intervals. Indeed, if a function describing the j-th intermediate result is nowhere defined, then its range is empty. So, we arrive at the following modification of the above procedure:

Definition 7. *Let* G *be a finite set of functions* $g : \mathbb{R}^k \to \mathbb{R}$*; for each of these functions* g*, we have:*

- *an interval extension* $\mathbf{G} : \mathbb{IR}^k \to \mathbb{IR}$*, and*
- *a function* $\text{def}_g : \mathbb{IR}^k \to \{0, 1\}$ *such that if* $\text{nowhere}_g(\mathbf{x}_1, \ldots, \mathbf{x}_k) = 1$ *(= "true"), then the function* $g(x_1, \ldots, x_k)$ *is not defined at any point* $(x_1, \ldots, x_k) \in \mathbf{x}_1 \times \ldots \times \mathbf{x}_k$*.*

Let $f = \langle n, x_{n+1}, \ldots, x_{n+c}, s_{n+1}, \ldots, s_N \rangle$ *be an algorithm with* n *inputs, and let* $\mathbf{x}_1, \ldots, \mathbf{x}_n$ *be* n *real numbers. Then, for each* $j \le N$*, by the* j-th intermediate result $\mathbf{f}_j$*, we mean an interval maybe empty) which is described by the following inductive definition:*

- *for* $j \le n$*, we define* $\mathbf{f}_j = \mathbf{x}_j$*;*
- *for* $n + 1 \le j \le n + c$*, we define* $\mathbf{f}_j = [x_j, x_j]$*;*
- *once the intervals* $\mathbf{f}_1, \ldots, \mathbf{f}_{j-1}$ *are defined and* $j > n + c$*, we take the* j*-th computation step* $x_j = g_j(x_{i_{j,1}}, \ldots, x_{i_{j,k_j}})$ *and define:*

  $\mathbf{f}_j = \emptyset$ *if* $\text{nowhere}_{g_j}(\mathbf{f}_{i_{j,1}}, \ldots, \mathbf{f}_{i_{j,k_j}}) = 1$ *otherwise* $\mathbf{f}_j = \mathbf{G}_j(\mathbf{f}_{i_{j,1}}, \ldots, \mathbf{f}_{i_{j,k_j}})$.

The last (N*-th) intermediate result is called the* result *of applying straightforward interval computations to the problem of estimating the range* $f(\mathbf{x}_1, \ldots, \mathbf{x}_n)$*; this result will be denoted by* $\mathbf{F}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$*.*

Proposition 5. *For every algorithm f with n inputs and for every tuple $\mathbf{x}_1$, ..., $\mathbf{x}_n$, the result $\mathbf{Y}$ of applying the above procedure encloses the range $\mathbf{y} = f(\mathbf{x}_1, \ldots, \mathbf{x}_n)$.*

References

[1] R. E. Moore, R. B. Kearfott, and M. J. Cloud, *Introduction to Interval Analysis*, SIAM Press, Philadelphia, Pennsylvania, 2009.

[2] J. Pryce, *Motion to Propose a "Discontinuous" Decoration Bit. Version 3*, Motion 22, IEEE Interval Standard Working Group P1788, 2010.

[3] S. Rabinovich, *Measurement Errors and Uncertainties: Theory and Practice*, Springer Verlag, New York, 2005.