Implementation of Interval Arithmetic in CORA 2016 (Tool Presentation)

Matthias Althoff[1] and Dmitry Grebenyuk[2]

[1] Technische Universität München,
Department of Informatics,
Munich, Germany
althoff@in.tum.de
[2] Ludwig-Maximilians-Universität München,
Department of Physics,
Munich, Germany
Dmitry.Grebenyuk@physik.uni-muenchen.de

Abstract

Interval arithmetic can be seen as one of the workhorses for formal verification approaches. The popularity of interval arithmetic stems from the fact that the possible outcomes of almost all frequently occurring mathematical expressions can be bounded. A disadvantage of interval arithmetic is that due to the negligence of dependencies of variables in expressions, results can be overly conservative. For this reason, interval arithmetic is typically used in formal verification tools when formulas are not contributing much to the accuracy of the overall approach, but do not belong to a restrictive class of expressions and are thus hard to evaluate. Although a lot of textbooks and software manuals for interval arithmetic exist, we have not found a complete and detailed description of how all standard mathematical functions are evaluated. This work changes this situation and describes concisely the evaluation of all standard mathematical functions. The described techniques are implemented as a class in CORA, a free MATLAB tool for continuous reachability analysis. Previously, CORA used the MATLAB toolbox INTLAB, but this tool is no longer freely available. Thus, our interval arithmetic class is currently the only freely available implementation of interval arithmetic that runs under current MATLAB versions. We have thoroughly tested our implementation against INTLAB and present the results.

1 Introduction

Cyber-physical systems arise across many innovative sectors in growth markets, such as automated vehicles, smart grids, and collaborative human-robot manufacturing to name only a few [29]. One of the main characteristics of cyber-physical systems is the tight interconnection of software with physical elements [19, 23]. Thus, a cyber-physical development process has to consider the discrete dynamics typically originating from computation and the continuous dynamics typically arising from physical components. Cyber-physical systems clearly perform better when the discrete and continuous dynamics is jointly considered in the modeling, the analysis, and the control design [13, 30].

Since analytical methods are basically non-existent for the analysis of mixed discrete and continuous dynamics, which are typically referred to as hybrid dynamics, algorithmic analysis techniques have been developed [21]. One of the most frequently used technique for analyzing hybrid systems is reachability analysis [27]. A variety of tools have been developed that

can perform reachability analysis as listed in the references of e.g. [2, 5, 6, 11, 17, 27]. Our tool *COntinuous Reachability Analyzer* (CORA) [4] realizes techniques for reachability analysis with a special focus on developing scalable solutions for verifying hybrid systems with non-linear continuous dynamics. Due to the modular design of CORA, much functionality can be used for other purposes that requires resource efficient representations of multi-dimensional sets and operations on them. The set representations that are currently supported are intervals, zonotopes, zonotope bundles, polynomial zonotopes, and polytopes. CORA also realizes the conversion between the aforementioned set representations, while some conversions are realized in an over-approximative fashion since not all representations are equally expressive. Methods for polytopes are not implemented in CORA, but realized by the MPT toolbox [10]. The same holds for intervals, for which CORA 2015 integrated the INTLAB toolbox [25]. However, INTLAB has become commercial and thus CORA 2015 is no longer a toolbox that solely relies on non-commercial software, which is not the intention of the developers. For this reason, we have implemented interval arithmetic on our own and have integrated it in CORA 2016. Since the newly implemented interval arithmetic is the main novelty in CORA 2016, we focus on this aspect.

Interval arithmetic is not new and has been developed over the last decades. Although numerous textbooks (see e.g. [9,12,18,22,24]), tool papers (see e.g. [3,7,14,15,26]), and software manuals (see e.g. [16,28]) on interval arithmetic exist, we have not found a detailed description of how a rather complete list of mathematical functions is implemented. It should also be noted that since 2015 an IEEE standard for interval arithmetic exists [1]. This standard discusses many important cases for which multiple outcomes are thinkable, e.g. how division by an interval including zero should be treated. However, the standard does not provide methods on how to evaluate mathematical functions. In contrast to previous work, we provide a detailed description of our implementation, which supports other researchers to implement their own interval arithmetic toolbox. Furthermore, one can only discuss in depth better implementations, if the used algorithm is described in detail. Since CORA does not consider rounding errors of floating point numbers, the first version of the interval arithmetic implemented in CORA also neglects this effect. Rounding effects are difficult to consider for transcendental functions due to the *table maker's dilemma* [8, Sec. 2.1.4], [20]. Computation of rounding errors for transcendental functions has been taken special care of in INTLAB, but details of those algorithms, which are specific to each function evaluation, are not published to the best knowledge of the authors. In the first version, we also restrict ourselves to intervals of real numbers and exclude intervals of complex numbers.

In Sec. 2 we describe how to compute the bounds of the outcome of scalar operations, such as the addition of two intervals. The computation of results for interval vectors and matrices is described in Sec. 3. Finally, in Sec. 4, we describe the implementation in CORA and compare the results with INTLAB.

2 Scalar Operations

A real-valued interval $[x] = [\underline{x}, \overline{x}] = \{x \in \mathbb{R} | \underline{x} \leq x \leq \overline{x}\}$ is a connected subset of $\mathbb{R}$ and can be specified by a left bound $\underline{x} \in \mathbb{R}$ and right bound $\overline{x} \in \mathbb{R}$, where $\underline{x} \leq \overline{x}$. Alternatively, one can specify an interval by its center $\text{mid}([x]) = 0.5(\overline{x}+\underline{x})$ and its radius $\text{rad}([x]) = 0.5(\overline{x}-\underline{x})$. After introducing set-based addition $A \oplus B = \{a+b | a \in A, b \in B\}$ and multiplication $A \otimes B = \{a\, b | a \in A, b \in B\}$, where multiplication binds stronger than addition, an interval can be obtained using

the center and radius as $[x] = \text{mid}([x]) \oplus \text{rad}([x]) \otimes [-1, 1]$. In this work, we use reversed square brackets to show that a bound is excluded from an interval, e.g. $[\underline{x}, \overline{x}[= \{x \in \mathbb{R} | \underline{x} \leq x < \overline{x}\}$. We further denote the empty set by $\emptyset$ and the numeric data type *not a number* by `NaN`. If `NaN` appears syntactically as the left or right bound of an interval, the result is no longer an interval. Nevertheless, we use `NaN` within the interval notation (e.g. $[\texttt{NaN}, 1]$) to indicate whether the computation of the left or right bound failed. The union and intersection are implemented in CORA as (see [1, Sec. 9.3])

$$\begin{aligned} [x] \cap [y] &= \begin{cases} [\max(\underline{x}, \underline{y}), \min(\overline{x}, \overline{y})] & \text{if } \max(\underline{x}, \underline{y}) \leq \min(\overline{x}, \overline{y}), \\ \emptyset & \text{otherwise,} \end{cases} \\ [x] \cup [y] &= \begin{cases} [\min(\underline{x}, \underline{y}), \max(\overline{x}, \overline{y})] & \text{if } \max(\underline{x}, \underline{y}) \leq \min(\overline{x}, \overline{y}), \\ [\texttt{NaN}, \texttt{NaN}] & \text{otherwise.} \end{cases} \end{aligned} \tag{1}$$

We introduce the interval hull of two intervals as (see [1, Sec. 9.3])

$$[x] \underline{\cup} [y] = [\min(\underline{x}, \underline{y}), \max(\overline{x}, \overline{y})].$$

The only binary operations typically required for scalar intervals and implemented in CORA are addition, subtraction, multiplication, and division. The result for a binary operation $\circ \in \{+, -, \cdot, /\}$ is defined as $[x] \circ [y] = \{x \circ y | x \in [x], y \in [y]\}$ and implemented in a straightforward way, except for division, as presented in previously mentioned textbooks (see e.g. [12, Sec. 2.3.3]):

$$\begin{aligned} [x] + [y] &= [\underline{x} + \underline{y}, \overline{x} + \overline{y}], \\ [x] - [y] &= [\underline{x} - \overline{y}, \overline{x} - \underline{y}], \\ [x] \cdot [y] &= [\min(\underline{xy}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{xy}), \max(\underline{xy}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{xy})]. \end{aligned} \tag{2}$$

Division is implemented differently depending on the software package. We use [12, eq. 2.54]:

$$[x]/[y] = [x] \cdot (1/[y]), \quad [1]/[y] = \begin{cases} \emptyset & \text{if } y = [0, 0], \\ [1/\overline{y}, 1/\underline{y}] & \text{if } 0 \notin [y], \\ [1/\overline{y}, \infty[& \text{if } (\underline{y} = 0) \wedge (\overline{y} > 0), \\ ]-\infty, 1/\underline{y}] & \text{if } (\underline{y} < 0) \wedge (\overline{y} = 0), \\ ]-\infty, \infty[& \text{if } (\underline{y} < 0) \wedge (\overline{y} > 0). \end{cases} \tag{3}$$

The solution is empty for $y = [0, 0]$ since there exists no x that would solve $1 = 0 \cdot x$. When we have the situation that $0 \in [y]$, the result can approach infinity since one can compute the limit for approaching 0. This, however, is not possible when the interval only consists of 0.

Unary operations are typically evaluations of elementary functions $f(x)$ defined as $f([x]) = \{f(x) | x \in [x]\}$. In the following, we present the interval evaluation of elementary functions implemented in CORA. We first discuss non-periodic function, followed by periodic functions.

2.1 Non-Periodic Functions

The easiest category of elementary functions with respect to interval evaluation, are monotonic functions. A function $f_{mi}(x)$ is monotonically increasing if $\forall x, y : x \leq y \Rightarrow f_{mi}(x) \leq f_{mi}(y)$

and a function $f_{md}(x)$ is monotonically decreasing if $\forall x, y : x \leq y \Rightarrow f_{md}(x) \geq f_{md}(y)$. Thus, it suffices to evaluate the left and right bound of such functions as for the exponential function:

$$e^{[x]} = [e^{\underline{x}}, e^{\overline{x}}]. \tag{4}$$

The same technique can be used for $\log(\cdot)$ and $\sqrt{(\cdot)}$, but they are only defined for positive values (when considering real interval arithmetic):

$$\log([x]) = \begin{cases} [\log \underline{x}, \log \overline{x}] & \text{if } \underline{x} \geq 0, \\ [\texttt{NaN}, \log \overline{x}] & \text{if } (\underline{x} < 0) \wedge (\overline{x} \geq 0), \\ [\texttt{NaN}, \texttt{NaN}] & \text{if } \overline{x} < 0, \end{cases} \quad \sqrt{[x]} = \begin{cases} [\sqrt{\underline{x}}, \sqrt{\overline{x}}] & \text{if } \underline{x} \geq 0, \\ [\texttt{NaN}, \sqrt{\overline{x}}] & \text{if } (\underline{x} < 0) \wedge (\overline{x} \geq 0), \\ [\texttt{NaN}, \texttt{NaN}] & \text{if } \overline{x} < 0. \end{cases} \tag{5}$$

Please note that other works (e.g. [12, eq. 2.48]), given the domain D of a function, compute $f([x]) = \{f(x) | x \in [x] \cap D\}$. Thus, based on the domain $D = [0, \infty[$ of $\sqrt{(\cdot)}$, one would obtain $\sqrt{[-1, 4]} = [0, 2]$ instead of $[\texttt{NaN}, 2]$. However, we use the latter solution to indicate that the input interval is improper. The inverse trigonometric functions $\arcsin(\cdot)$, $\arccos(\cdot)$, and $\arctan(\cdot)$ are all monotonic. However, $\arcsin(\cdot)$ and $\arccos(\cdot)$ are only defined for values within $[-1, 1]$. If the argument is not within this interval, we return $\texttt{NaN}$ to be consistent with $\log(\cdot)$ and $\sqrt{(\cdot)}$:

$$\begin{aligned} \arcsin([x]) &= \begin{cases} [\arcsin(\underline{x}), \arcsin(\overline{x})] & \text{if } (\underline{x} \geq -1) \wedge (\overline{x} \leq 1), \\ [\arcsin(\underline{x}), \texttt{NaN}] & \text{if } (\underline{x} \in [-1, 1]) \wedge (\overline{x} > 1), \\ [\texttt{NaN}, \arcsin(\overline{x})] & \text{if } (\underline{x} < -1) \wedge (\overline{x} \in [-1, 1]), \\ [\texttt{NaN}, \texttt{NaN}] & \text{if } (\underline{x} < -1) \wedge (\overline{x} > 1), \end{cases} \\ \arccos([x]) &= \begin{cases} [\arccos(\overline{x}), \arccos(\underline{x})] & \text{if } (\underline{x} \geq -1) \wedge (\overline{x} \leq 1), \\ [\arccos(\overline{x}), \texttt{NaN}] & \text{if } (\underline{x} < -1) \wedge (\overline{x} \in [-1, 1]), \\ [\texttt{NaN}, \arccos(\underline{x})] & \text{if } (\underline{x} \in [-1, 1]) \wedge (\overline{x} > 1), \\ [\texttt{NaN}, \texttt{NaN}] & \text{if } (\underline{x} < -1) \wedge (\overline{x} > 1), \end{cases} \\ \arctan([x]) &= [\arctan(\underline{x}), \arctan(\overline{x})]. \end{aligned} \tag{6}$$

The hyperbolic functions $\sinh(\cdot)$ and $\tanh(\cdot)$ are monotonic and $\cosh(\cdot)$ is monotonic within $]-\infty, 0]$ and $[0, \infty[$:

$$\begin{aligned} \sinh([x]) &= [\sinh(\underline{x}), \sinh(\overline{x})], \\ \cosh([x]) &= \begin{cases} [\cosh(\overline{x}), \cosh(\underline{x})] & \text{if } \overline{x} < 0, \\ [1, \cosh(\max(|\underline{x}|, |\overline{x}|))] & \text{if } (\underline{x} \leq 0) \wedge (\overline{x} \geq 0), \\ [\cosh(\underline{x}), \cosh(\overline{x})] & \text{if } \underline{x} > 0, \end{cases} \\ \tanh([x]) &= [\tanh(\underline{x}), \tanh(\overline{x})]. \end{aligned} \tag{7}$$

The inverse hyperbolic functions arcsinh(·), arccosh(·), and arctanh(·) are all monotonically increasing, where arccosh(·) is only defined for $[1,\infty[$ and arctanh(·) is only defined for $]-1,1[$.

$$\begin{aligned}
\operatorname{arcsinh}([x]) &= [\operatorname{arcsinh}(\underline{x}), \operatorname{arcsinh}(\overline{x})], \\
\operatorname{arccosh}([x]) &= \begin{cases} [\operatorname{arccosh}(\underline{x}), \operatorname{arccosh}(\overline{x})] & \text{if } \underline{x} \geq 1, \\ [\texttt{NaN}, \operatorname{arccosh}(\overline{x})] & \text{if } (\underline{x} < 1) \wedge (\overline{x} \geq 1), \\ [\texttt{NaN}, \texttt{NaN}] & \text{if } \overline{x} < 1, \end{cases} \\
\operatorname{arctanh}([x]) &= \begin{cases} [\operatorname{arctanh}(\underline{x}), \operatorname{arctanh}(\overline{x})] & \text{if } (\underline{x} > -1) \wedge (\overline{x} < 1), \\ [\operatorname{arctanh}(\underline{x}), \texttt{NaN}] & \text{if } (\underline{x} \in]-1,1[) \wedge (\overline{x} \geq 1), \\ [\texttt{NaN}, \operatorname{arctanh}(\overline{x})] & \text{if } (\underline{x} \leq -1) \wedge (\overline{x} \in]-1,1[), \\ [\texttt{NaN}, \texttt{NaN}] & \text{if } (\underline{x} \leq -1) \wedge (\overline{x} \geq 1). \end{cases}
\end{aligned} \tag{8}$$

The power $[x]^n$, where $n \in \mathbb{N}$, is monotonic for uneven n and monotonic for even n within $]-\infty,0]$ and $[0,\infty[$ (see [22, Appendix B]):

$$[x]^n = \begin{cases} [\underline{x}^n, \overline{x}^n] & \text{if } (\underline{x} > 0) \vee (n \text{ uneven}), \\ [\overline{x}^n, \underline{x}^n] & \text{if } (\overline{x} < 0) \wedge (n \text{ even}), \\ [0, \max(|\underline{x}|, |\overline{x}|)^n] & \text{if } (0 \in [x]) \wedge (n \text{ even}). \end{cases} \tag{9}$$

The absolute value function is monotonic for $]-\infty,0]$ and $[0,\infty[$:

$$|[x]| = \begin{cases} [|\overline{x}|, |\underline{x}|] & \text{if } \overline{x} < 0, \\ [\underline{x}, \overline{x}] & \text{if } \underline{x} > 0, \\ [0, \max(|\underline{x}|, |\overline{x}|)] & \text{if } 0 \in [x]. \end{cases} \tag{10}$$

Please note that this result is in line with the previous definition $f([x]) = \{f(x) | x \in [x]\}$. However, many textbooks (see e.g. [22,24]) define $|[x]| = \max(|\underline{x}|, |\overline{x}|)$. In [9, Example 3.4], the syntax $abs([x])$ is used for the absolute value function, which is evaluated as in this work, and $||[x]||$ is used for the largest absolute value. To avoid any confusion, the IEEE standard uses the syntax $abs([x])$ for the absolute value function (see [1, Tab. 9.1]) and $mag([x]) = \max(|\underline{x}|, |\overline{x}|)$ for the largest absolute value (see [1, Tab. 9.2]).

2.2 Periodic Functions

Monotonicity is also exploited in CORA for trigonometric functions as suggested by standard textbooks. However, in none of the checked textbooks [9,12,18,22,24] we have found a detailed description of how all trigonometric functions are implemented. Throughout this subsection, we assume that the arguments of trigonometric functions are in radians. If they are in degree, they can be easily converted to radians upfront. Due to the periodic nature of trigonometric functions, it suffices to only consider intervals whose radius $\operatorname{rad}([x])$ is below a certain threshold.

Please note that it is not required to provide implementations for all trigonometric functions, since they depend on each other, e.g. $\sin([x]) = \cos(\frac{\pi}{2} - [x])$. This, however, would require additional operations, such as $\frac{\pi}{2} - [x]$ in the previous example, which reduces the computational efficiency. For this reason, we suggest the implementations as presented subsequently. We begin with the sin(·) function (see Fig. 2), for which holds

$$\sin([x]) = [-1,1] \text{ if } \overline{x} - \underline{x} \geq 2\pi$$

so that we can focus on the case $\overline{x} - \underline{x} < 2\pi$. We further simplify the problem by considering only values within the interval $[0, 2\pi[$ by using the modulo function from which we obtain

$$\underline{y} = \text{mod}(\underline{x}, 2\pi), \quad \overline{y} = \text{mod}(\overline{x}, 2\pi). \tag{11}$$

The following proposition helps distinguishing the different cases for obtaining $\sin([x])$:

Proposition 1 (Interval bound difference after applying the modulo operator)**.** *There only exist two cases for* $\overline{x} - \underline{x} < 2\pi$ *after applying* (11)*:*

$$\overline{x} - \underline{x} = \overline{y} - \underline{y} \text{ (case 1)}$$
$$\overline{x} - \underline{x} = (\overline{y} - \underline{y}) + 2\pi \text{ (case 2)}$$

Proof. After applying the modulo operator, we have that $\underline{y} \in [0, 2\pi[$ and $\overline{y} \in [0, 2\pi[$ so that $\overline{y} - \underline{y} \in]-2\pi, 2\pi[$. If $\overline{y} - \underline{y} \in [0, 2\pi[$, we obtain case 1 since $\overline{x} - \underline{x} < 2\pi$ and the difference $(\overline{x} - \underline{x}) - (\overline{y} - \underline{y})$ can only be a multiple of 2π. The remaining interval is $\overline{y} - \underline{y} \in]-2\pi, 0[$, which results in case 2, since only the addition of 2π ensures $\overline{x} - \underline{x} = (\overline{y} - \underline{y}) + 2\pi \in [0, 2\pi[$. □

Thus, it suffices to check whether $\underline{y} > \overline{y}$ for necessary corrections, since when $\underline{y} \leq \overline{y}$ we know that $\overline{x} - \underline{x} = \overline{y} - \underline{y}$. To compute the $\sin(\cdot)$ function, we separate the interval $[0, 2\pi[$ into three monotonic regions (see Fig. 1) with

$$Rs_1 = [0, \frac{\pi}{2}[, \quad Rs_2 = [\frac{\pi}{2}, \frac{3\pi}{2}[, \quad Rs_3 = [\frac{3\pi}{2}, 2\pi[.$$

If the regions of $\underline{y}$ and $\overline{y}$ are equal, we additionally have to check whether $\underline{y} > \overline{y}$ holds – this, of course, follows directly when the regions are unequal. Thus, in total, we have 6 combinations of unequal regions and 3 combinations of equal regions, for which we have to check whether $\underline{y} > \overline{y}$. Thus, in total we have $6 + 2 \cdot 3 = 12$ cases:

$$\sin([x]) = \begin{cases} [-1, 1] & \text{if} \quad (\overline{x} - \underline{x} \geq 2\pi) \vee \\ & \quad (\underline{y} \in Rs_1 \wedge \overline{y} \in Rs_1 \wedge \underline{y} > \overline{y}) \vee \\ & \quad (\underline{y} \in Rs_1 \wedge \overline{y} \in Rs_3) \vee \\ & \quad (\underline{y} \in Rs_2 \wedge \overline{y} \in Rs_2 \wedge \underline{y} > \overline{y}), \\ & \quad (\underline{y} \in Rs_3 \wedge \overline{y} \in Rs_3 \wedge \underline{y} > \overline{y}), \\ [\sin(\underline{y}), \sin(\overline{y})] & \text{if} \quad (\underline{y} \in Rs_1 \wedge \overline{y} \in Rs_1 \wedge \underline{y} \leq \overline{y}) \vee \\ & \quad (\underline{y} \in Rs_3 \wedge \overline{y} \in Rs_1) \vee \\ & \quad (\underline{y} \in Rs_3 \wedge \overline{y} \in Rs_3 \wedge \underline{y} \leq \overline{y}), \\ [\min(\sin(\underline{y}), \sin(\overline{y})), 1] & \text{if} \quad (\underline{y} \in Rs_1 \wedge \overline{y} \in Rs_2), \\ & \quad (\underline{y} \in Rs_3 \wedge \overline{y} \in Rs_2), \\ [-1, \max(\sin(\underline{y}), \sin(\overline{y}))] & \text{if} \quad (\underline{y} \in Rs_2 \wedge \overline{y} \in Rs_1) \vee \\ & \quad (\underline{y} \in Rs_2 \wedge \overline{y} \in Rs_3), \\ [\sin(\overline{y}), \sin(\underline{y})] & \text{if} \quad (\underline{y} \in Rs_2 \wedge \overline{y} \in Rs_2 \wedge \underline{y} \leq \overline{y}). \end{cases} \tag{12}$$

The correctness of each case can be easily checked via visual inspection in Fig. 1; if $\underline{y} > \overline{y}$, the inspection has to be performed as demonstrated in Fig. 2.

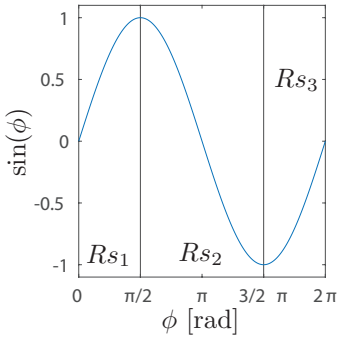

Figure 1: Regions of sine.

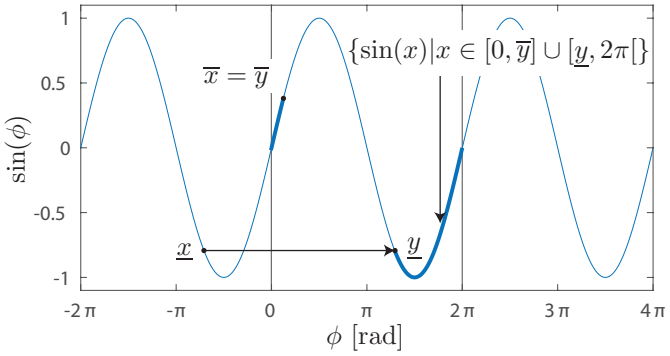

Figure 2: Movement of infimum and supremum after applying the modulo operator.

For computing the $\cos(\cdot)$ function, we only require two monotonic regions (see Fig. 3) with

$$Rc_1 = [0, \pi[, \quad Rc_2 = [\pi, 2\pi[.$$

Thus, in total, we have 2 cases with unequal regions and 4 cases with equal regions, yielding 6 cases:

$$\cos([x]) = \begin{cases} [-1,1] & \text{if} \quad (\overline{x} - \underline{x} \geq 2\pi) \vee \\ & \quad (\underline{y} \in Rc_1 \wedge \overline{y} \in Rc_1 \wedge \underline{y} > \overline{y}) \vee \\ & \quad (\underline{y} \in Rc_2 \wedge \overline{y} \in Rc_2 \wedge \underline{y} > \overline{y}), \\ [\cos(\underline{y}), \cos(\overline{y})] & \text{if} \quad (\underline{y} \in Rc_2 \wedge \overline{y} \in Rc_2 \wedge \underline{y} \leq \overline{y}), \\ [\min(\cos(\underline{y}), \cos(\overline{y})), 1] & \text{if} \quad (\underline{y} \in Rc_2 \wedge \overline{y} \in Rc_1), \\ [-1, \max(\cos(\underline{y}), \cos(\overline{y}))] & \text{if} \quad (\underline{y} \in Rc_1 \wedge \overline{y} \in Rc_2), \\ [\cos(\overline{y}), \cos(\underline{y})] & \text{if} \quad (\underline{y} \in Rc_1 \wedge \overline{y} \in Rc_1 \wedge \underline{y} \leq \overline{y}). \end{cases} \tag{13}$$

The $\tan(\cdot)$ function has a shorter period of π so that we use slightly different auxiliary angles compared to $\sin(\cdot)$ and $\cos(\cdot)$:

$$\underline{z} = \mathrm{mod}(\underline{x}, \pi), \quad \overline{z} = \mathrm{mod}(\overline{x}, \pi).$$

For $\tan(\cdot)$ we require two monotonic regions (see Fig. 4) with

$$Rt_1 = [0, \pi/2[, \quad Rt_2 = [\pi/2, \pi[,$$

resulting in 6 cases as for $\cos(\cdot)$:

$$\tan([x]) = \begin{cases}]-\infty, \infty[& \text{if} \quad (\overline{x} - \underline{x} \geq \pi) \vee \\ & \quad (\underline{z} \in Rt_1 \wedge \overline{z} \in Rt_1 \wedge \underline{z} > \overline{z}) \vee \\ & \quad (\underline{z} \in Rt_2 \wedge \overline{z} \in Rt_2 \wedge \underline{z} > \overline{z}) \vee \\ & \quad (\underline{z} \in Rt_1 \wedge \overline{z} \in Rt_2), \\ [\tan(\underline{z}), \tan(\overline{z})] & \text{if} \quad (\underline{z} \in Rt_1 \wedge \overline{z} \in Rt_1 \wedge \underline{z} \leq \overline{z}) \vee \\ & \quad (\underline{z} \in Rt_2 \wedge \overline{z} \in Rt_2 \wedge \underline{z} \leq \overline{z}). \end{cases} \tag{14}$$

The $\cot(\cdot)$ function is monotonically decreasing in $[0, \pi[$ so that we directly obtain

$$\cot([x]) = \begin{cases}]-\infty, \infty[& \text{if} \quad (\overline{x} - \underline{x} \geq \pi) \ \vee \ (\underline{z} > \overline{z}), \\ [\cot(\overline{z}), \cot(\underline{z})] & \text{if} \quad (\underline{z} \leq \overline{z}). \end{cases} \tag{15}$$

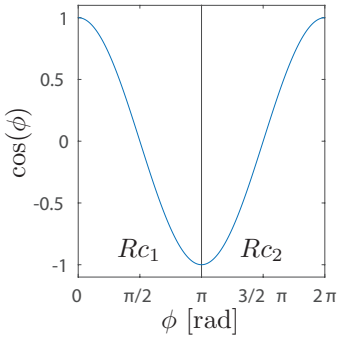

Figure 3: Regions of cosine.

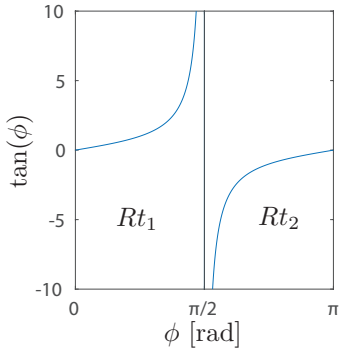

Figure 4: Regions of tangent.

3 Vector/Matrix Operations

This section addresses how CORA handles interval vectors and matrices. An interval vector $[\mathbf{x}] \subseteq \mathbb{R}^n$, which we denote by a small bold symbol, can be defined as the Cartesian product of n intervals ($[\mathbf{x}] = [x_1] \times [x_2] \ldots \times [x_n]$) or as the set bounded by a left vector $\underline{\mathbf{x}} \in \mathbb{R}^n$ and a right vector $\overline{\mathbf{x}} \in \mathbb{R}^n$ ($[\mathbf{x}] = \{\mathbf{x} | \underline{x}_i \leq x_i \leq \overline{x}_i, i \in \{1, \ldots, n\}\}$). Analogously, a $(m \times n)$-dimensional interval matrix $[\mathbf{X}] \subseteq \mathbb{R}^{m \times n}$, which we denote by a capital bold symbol, can be defined as the Cartesian product of $m \times n$ intervals ($[\mathbf{X}] = [x_{11}] \times [x_{12}] \ldots \times [x_{21}] \ldots \times [x_{mn}]$) or as the set bounded by a left matrix $\underline{\mathbf{X}} \in \mathbb{R}^{m \times n}$ and a right matrix $\overline{\mathbf{X}} \in \mathbb{R}^{m \times n}$ ($[\mathbf{X}] = \{\mathbf{X} | \underline{x}_{ij} \leq x_{ij} \leq \overline{x}_{ij}, i \in \{1, \ldots, m\}, j \in \{1, \ldots, n\}\}$).

Matrix multiplication and the multiplication of a scalar with a matrix is defined and implemented in CORA for $[\mathbf{X}] \subseteq \mathbb{R}^{o \times n}$, $[\tilde{\mathbf{X}}] \subseteq \mathbb{R}^{o \times n}$, $[\mathbf{Y}] \subseteq \mathbb{R}^{n \times p}$, and $[a] \subseteq \mathbb{R}$ as

$$([\mathbf{X}][\mathbf{Y}])_{ij} = \sum_{k=1}^{n} [\mathbf{X}]_{ik} [\mathbf{Y}]_{kj}, \quad ([a][\mathbf{X}])_{ij} = [a][\mathbf{X}]_{ij}. \tag{16}$$

Matrix addition is straightforwardly implemented as

$$[\mathbf{X}] + [\tilde{\mathbf{X}}] = [\underline{\mathbf{X}} + \underline{\tilde{\mathbf{X}}}, \overline{\mathbf{X}} + \overline{\tilde{\mathbf{X}}}]. \tag{17}$$

All unary operations are realized in CORA elementwise as

$$f([\mathbf{X}])_{ij} = f([\mathbf{X}]_{ij})$$

Note that we typically compute with larger intervals. For smaller intervals, there exist techniques that directly operate on the center matrix and the radius matrix for computational efficiency [26], but obtain results that are not tight anymore.

4 Implementation in CORA

This section lists the functions realized in CORA. Since CORA is implemented in MATLAB, the function names are chosen such that they overload the built-in MATLAB functions. The

currently implemented functions are listed in Tab. 1 and Tab. 2. The list in Tab. 1 shows which mathematical functions have been realized and Tab. 2 shows all other functions, which are required to specify intervals, arrange them in matrices, access values, perform set operations, display the intervals, among others.

Table 1: Methods of the class `interval` that realize mathematical functions. All functions can be applied to scalars, vectors, or matrices.

name	description
`abs`	returns the absolute value as defined in (10)
`acos`	arccos(·) function as defined in (6)
`acosh`	arccosh(·) function as defined in (8)
`asin`	arcsin(·) function as defined in (6)
`asinh`	arcsinh(·) function as defined in (8)
`atan`	arctan(·) function as defined in (6)
`atanh`	arctanh(·) function as defined in (8)
`cos`	cos(·) function as defined in (13)
`cosh`	cosh(·) function as defined in (7)
`exp`	exponential function as defined in (4)
`log`	natural logarithm function as defined in (5)
`minus`	overloaded '-' operator, see (2)
`mpower`	overloaded '^' operator (power), see (9)
`mrdivide`	overloaded '/' operator (division), see (3)
`mtimes`	overloaded '*' operator (multiplication), see (2) for scalars and (16) for matrices
`plus`	overloaded '+' operator (addition), see (2) for scalars and (17) for matrices
`rdivide`	overloads the './' operator: provides elementwise division of two matrices
`sin`	sin(·) function as defined in (12)
`sinh`	sinh(·) function as defined in (7)
`sqrt`	$\sqrt{(\cdot)}$ function as defined in (5)
`tan`	tan(·) function as defined in (14)
`tanh`	tanh(·) function as defined in (7)
`times`	overloaded '.*' operator for elementwise multiplication of matrices
`uminus`	overloaded '-' operator for a single operand
`uplus`	overloaded '+' operator for single operand

4.1 Examples

An interval matrix can be generated by first generating scalar intervals and then assembling them using the same MATLAB syntax as for real-valued variables, or by directly specifying the left and right bounds:

```
a_11 = interval(-1,1); % generate scalar interval
a_12 = interval(-1,2); % generate scalar interval
a_21 = interval(0.5,1); % generate scalar interval
a_22 = interval(-2,-1); % generate scalar interval

A_variant1 = [a_11 a_12; a_21 a_22] % generate matrix and display it

A_inf = [-1 -1; 0.5 -2] % specify infimum
A_sup = [1 2; 1 -1] % specify supremum

A_variant2 = interval(A_inf, A_sup) % generate matrix and display it

firstElement = A_variant2(1,1) % obtain and display first element
```

Table 2: Methods of the class `interval` that do not realize mathematical functions. All functions can be applied to scalars, vectors, or matrices.

name	description
`and`	returns the intersection as defined in (1)
`display`	displays the values of the `interval` object in the MATLAB workspace
`horzcat`	overloads the operator for horizontal concatenation, e.g. `a = [b,c,d]`
`infimum`	returns the infimum of an interval
`interval`	constructor of the `interval` class
`isempty`	returns 1 if interval is empty and 0 otherwise
`isscalar`	returns 1 if interval is scalar and 0 otherwise
`length`	overloads the operator that returns the length of the longest array dimension
`mid`	returns the center of an interval
`rad`	returns the radius of an interval
`size`	overloads the operator that returns the size of the object, i.e., length of an array in each dimension
`subsasgn`	overloads the operator that assigns elements of an interval matrix `I`, e.g. `I(1,2)=value`, where the element of the first row and second column is set
`subsref`	overloads the operator that selects elements of an interval matrix `I`, e.g. `value=I(1,2)`, where the element of the first row and second column is read
`supremum`	returns the supremum of an interval
`transpose`	overloads the '.'' operator to compute the transpose of an interval matrix
`vertcat`	overloads the operator for vertical concatenation, e.g. `a = [b;c;d]`

```
14 firstRow = A_variant2(1,:) % obtain and display first row
15 secondColumn = A_variant2(:,2) % obtain and display second column
```

This produces the workspace output

```
A_variant1 =
   [-1.00000,1.00000] [-1.00000,2.00000]
   [0.50000,1.00000] [-2.00000,-1.00000]

A_variant2 =
   [-1.00000,1.00000] [-1.00000,2.00000]
   [0.50000,1.00000] [-2.00000,-1.00000]

firstElement =
   [-1.00000,1.00000]

firstRow =
   [-1.00000,1.00000] [-1.00000,2.00000]

secondColumn =
   [-1.00000,2.00000]
   [-2.00000,-1.00000]
```

Mathematical functions are evaluated using the MATLAB syntax as for real-valued variables:

```
1 A_inf = [-1 -1; 0.5 -2]; % specify infimum
2 A_sup = [1 2; 1 -1]; % specify supremum
3 A = interval(A_inf, A_sup); % generate matrix
4
5 B_inf = [0 -1; 0 1]; % specify infimum
6 B_sup = [1 3; 0.5 1.2]; % specify supremum
```

```
B = interval(B_inf, B_sup); % generate matrix

A+B % addition
A*B % multiplication
A.*B % pointwise multiplication
A/interval(1,2) % division
A./B % pointwise division
A^3 % power function
sin(A) % sine function
sin(A(1,1)) + A(1,1)^2 - A(1,1)*B(1,1) % scalar combination of functions
sin(A) + A^2 - A*B % matrix combination of functions
```

This produces the workspace output

```
A+B =
   [-1.00000,2.00000] [-2.00000,5.00000]
   [0.50000,1.50000] [-1.00000,0.20000]

A*B =
   [-1.50000,2.00000] [-4.20000,5.40000]
   [-1.00000,1.00000] [-3.40000,2.00000]

A.*B =
   [-1.00000,1.00000] [-3.00000,6.00000]
   [0.00000,0.50000] [-2.40000,-1.00000]

A/interval(1,2) =
   [-1.00000,1.00000] [-1.00000,2.00000]
   [0.25000,1.00000] [-2.00000,-0.50000]

A./B =
   [-Inf,Inf] [-Inf,Inf]
   [1.00000,Inf] [-2.00000,-0.83333]

A^3 =
   [-9.00000,7.00000] [-12.00000,18.00000]
   [-3.00000,9.00000] [-18.00000,3.00000]

sin(A) =
   [-0.84147,0.84147] [-0.84147,1.00000]
   [0.47943,0.84147] [-1.00000,-0.84147]

sin(A(1,1)) + A(1,1)^2 - A(1,1)*B(1,1) =
   [-1.84147,2.84147]

sin(A) + A^2 - A*B =
   [-4.84147,5.34147] [-12.24147,9.10930]
   [-3.52057,2.34147] [-2.90930,8.55853]
```

4.2 Comparison with INTLAB

In this subsection, we compare the speed and accuracy of our implementation with INTLAB. Please note that we do not compare the results with b4m [31] since this tool no longer runs on current MATLAB versions. Since INTLAB considers rounding errors of floating-point numbers, the results of INTLAB are slightly larger intervals due to outwards rounding. As mentioned in the introduction, the first version of interval arithmetic realized in CORA does not consider floating-point errors, since they are also not considered in other computations of CORA. The main purpose of CORA is to prototypically realize algorithms for reachable set computations. Once an algorithm is mature, one should consider rounding errors when aiming for a commercial tool. Since we neglect floating-point errors, our implementation is faster for almost all functions compared to INTLAB as shown in Tab. 3. The computation times have been averaged from 10^4 runs for each function and have been carried out on an Intel Core i5-3230M processor running at 2.60GHz. Please note that this is an old processor and computation times have been observed to be more than 100 times faster on a modern processor.

Let us introduce the left and right bound of the j^{th} out of $N = 10^4$ evaluations of a particular function in CORA as $\underline{x}_{C,j}$ and $\overline{x}_{C,j}$. Likewise, the bound of the j^{th} evaluation using INTLAB is denoted by $\underline{x}_{I,j}$ and $\overline{x}_{I,j}$. The maximum relative error over all runs is defined as

$$\epsilon = \max(\mu_1, \ldots, \mu_N), \quad \mu_j = \frac{\max(|\underline{x}_{C,j} - \underline{x}_{I,j}|, |\overline{x}_{C,j} - \overline{x}_{I,j}|)}{\overline{x}_{C,j} - \underline{x}_{C,j}}. \tag{18}$$

We have used the same 10^4 runs for each function as for measuring the average computation time. We introduce the random variables $\mathcal{U}$ and $\mathcal{U}_\Delta$, which are uniformly distributed within an interval. The interval for $\mathcal{U}$ is $[\underline{u}, \overline{u}]$ and the one for $\mathcal{U}_\Delta$ is $[\underline{u}_\Delta, \overline{u}_\Delta]$. We construct the following random input intervals for unary operations: $[\mathcal{U}, \mathcal{U} + \mathcal{U}_\Delta]$. For binary operations, this random interval is computed twice. For our implementation, we have used the standard precision of MATLAB floating point numbers, which is the double-precision data type according to the IEEE 754 standard. The bound of the random variables vary depending on the tested function and are listed in Tab. 3. The results of the evaluation in Tab. 3 show that the maximum relative error ϵ over all runs is marginal compared to the range of intervals $\overline{x}_{C,j} - \underline{x}_{C,j}$.

5 Conclusions

This paper has three main contributions. First, we present how we integrate interval arithmetic into CORA and list all implemented methods. Currently, we provide the only freely available MATLAB implementation of interval arithmetic running under the latest MATLAB versions. Second, we compare our implementation to INTLAB; the main result of our comparison is that our tool obtains results faster, but does not consider floating-point errors. These, however, are also not yet considered in the other methods provided by CORA, mainly since the tool is designed for prototypically testing different approaches for reachability analysis. The resulting error from neglecting floating-point arithmetic is marginal for the tested functions. Finally, we provide all required formulas for evaluating the range of functions – all previous works referenced in here do not provide such an exhaustive list, but only point out that monotonicity should be considered when evaluating functions. We admit that the evaluations of functions are not difficult, but especially when considering cases where the input interval is not within the

Table 3: Comparison with INTLAB.

		avg. comp. time [s]		input intervals			
function	**error ϵ; see (18)**	**INTLAB**	**CORA**	$\underline{u}$	$\overline{u}$	$\underline{u}_\Delta$	$\overline{u}_\Delta$
`abs`	0	$7.388 \cdot 10^{-5}$	$8.503 \cdot 10^{-5}$	-100	100	0	100
`acos`	$4.711 \cdot 10^{-12}$	$4.881 \cdot 10^{-4}$	$1.346 \cdot 10^{-4}$	-1	0	0	1
`acosh`	$8.208 \cdot 10^{-13}$	$5.433 \cdot 10^{-4}$	$1.279 \cdot 10^{-4}$	1	10	0	10
`asin`	$1.948 \cdot 10^{-12}$	$4.318 \cdot 10^{-4}$	$1.282 \cdot 10^{-4}$	-1	0	0	1
`asinh`	$2.304 \cdot 10^{-12}$	$5.196 \cdot 10^{-4}$	$1.131 \cdot 10^{-4}$	-100	100	0	100
`atan`	$5.915 \cdot 10^{-11}$	$3.179 \cdot 10^{-4}$	$1.081 \cdot 10^{-4}$	-100	100	0	100
`atanh`	$4.801 \cdot 10^{-12}$	$4.711 \cdot 10^{-4}$	$1.271 \cdot 10^{-4}$	-1	0	0	1
`cos`	$8.515 \cdot 10^{-13}$	$4.857 \cdot 10^{-4}$	$1.086 \cdot 10^{-4}$	-100	100	0	100
`cosh`	$1.865 \cdot 10^{-14}$	$3.971 \cdot 10^{-4}$	$1.203 \cdot 10^{-4}$	-100	100	0	100
`exp`	$2.627 \cdot 10^{-14}$	$2.262 \cdot 10^{-4}$	$6.840 \cdot 10^{-5}$	-100	100	0	100
`log`	$2.028 \cdot 10^{-11}$	$3.090 \cdot 10^{-4}$	$1.127 \cdot 10^{-4}$	0	100	0	100
`minus`	$7.583 \cdot 10^{-15}$	$9.846 \cdot 10^{-5}$	$8.425 \cdot 10^{-5}$	-100	100	0	100
`mpower` $(()^3)$	$1.873 \cdot 10^{-12}$	$2.993 \cdot 10^{-4}$	$1.550 \cdot 10^{-4}$	0	100	0	100
`mrdivide`	$4.656 \cdot 10^{-15}$	$2.300 \cdot 10^{-4}$	$3.449 \cdot 10^{-4}$	-100	100	0	100
`mtimes`	$4.053 \cdot 10^{-15}$	$1.817 \cdot 10^{-4}$	$1.208 \cdot 10^{-4}$	-100	100	0	100
`plus`	$3.327 \cdot 10^{-15}$	$9.453 \cdot 10^{-5}$	$5.625 \cdot 10^{-5}$	-100	100	0	100
`rdivide`	$4.896 \cdot 10^{-15}$	$2.056 \cdot 10^{-4}$	$2.735 \cdot 10^{-4}$	-100	100	0	100
`sin`	$3.139 \cdot 10^{-13}$	$4.093 \cdot 10^{-4}$	$9.839 \cdot 10^{-5}$	-100	100	0	100
`sinh`	$1.902 \cdot 10^{-12}$	$3.224 \cdot 10^{-4}$	$1.044 \cdot 10^{-4}$	-100	100	0	100
`sqrt`	$1.407 \cdot 10^{-12}$	$1.524 \cdot 10^{-4}$	$1.163 \cdot 10^{-4}$	0	100	0	100
`tan`	$5.627 \cdot 10^{-12}$	$5.371 \cdot 10^{-4}$	$1.267 \cdot 10^{-4}$	$-\frac{\pi}{2} + 0.1$	0	0	$\frac{\pi}{2} - 0.1$
`tanh`	$1.418 \cdot 10^{-12}$	$2.432 \cdot 10^{-4}$	$1.050 \cdot 10^{-4}$	-1	1	0	1
`times`	$9.434 \cdot 10^{-15}$	$1.828 \cdot 10^{-4}$	$1.257 \cdot 10^{-4}$	-100	100	0	100
`uminus`	0	$1.880 \cdot 10^{-5}$	$3.452 \cdot 10^{-5}$	-100	100	0	100
`uplus`	0	$1.078 \cdot 10^{-5}$	$1.631 \cdot 10^{-5}$	-100	100	0	100

domain of a function, we believe that a complete documentation of the implemented algorithms is beneficial.

Acknowledgment

The authors gratefully acknowledge financial support by the European Commission project UnCoVerCPS under grant number 643921.

References

[1] IEEE standard for interval arithmetic. *IEEE Std 1788-2015*, pages 1–97, June 2015.

[2] A.Eggers, N. Ramdani, N. S. Nedialkov, and M. Fränzle. Improving the sat modulo ode approach to hybrid systems analysis by combining different enclosure methods. *Software & Systems Modeling*, 14(1):121–148, 2012.

[3] G. Alefeld and G. Mayer. Interval analysis: Theory and applications. *Computational and Applied Mathematics*, 121:421–464, 2000.

[4] M. Althoff. An introduction to CORA 2015. In *Proc. of the Workshop on Applied Verification for Continuous and Hybrid Systems*, pages 120–151, 2015.

[5] E. Asarin, T. Dang, G. Frehse, A. Girard, C. Le Guernic, and O. Maler. Recent progress in continuous and hybrid reachability analysis. In *Proc. of the 2006 IEEE Conference on Computer Aided Control Systems Design*, pages 1582–1587, 2006.

[6] X. Chen, E. Ábrahám, and S. Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *Proc. of Computer-Aided Verification*, LNCS 8044, pages 258–263. Springer, 2013.

[7] J. S. Ely. The VPI software package for variable precision interval arithmetic. *Interval Computations*, 2:135–153, 1993.

[8] D. Goldberg. What every computer scientist should know about floating point arithmetic. *ACM Computing Surveys*, 23(1):5–48, 1991.

[9] R. Hammer, M. Hocks, U. Kulisch, and D. Ratz. *C++ Toolbox for Verified Computing I*. Springer, 1995.

[10] M. Herceg, M. Kvasnica, C.N. Jones, and M. Morari. Multi-Parametric Toolbox 3.0. In *Proc. of the European Control Conference*, pages 502–510, Zürich, Switzerland, July 17–19 2013. http://control.ee.ethz.ch/~mpt.

[11] F. Immler. Tool presentation: Isabelle/HOL for reachability analysis of continuous systems. In *Proc. of the 2nd Workshop on Applied Verification for Continuous and Hybrid Systems.*, pages 180–187, 2015.

[12] L. Jaulin, M. Kieffer, and O. Didrit. *Applied Interval Analysis*. Springer, 2006.

[13] J. C. Jensen, D. H. Chang, and E. A. Lee. A model-based design methodology for cyber-physical systems. In *Proc. of the 7th International Wireless Communications and Mobile Computing Conference*, pages 1666–1671, 2011.

[14] R. B. Kearfott. Algorithm 763: INTERVAL_ARITHMETIC: A Fortran 90 module for an interval data type. *ACM Transactions on Mathematical Software*, 22(4):385–392, 1996.

[15] R. B. Kearfott, M. Dawande, K. Du, and C. Hu. Algorithm 737: INTLIB: A portable Fortran 77 interval standard-function library. *ACM Transactions on Mathematical Software*, 20(4):447–459, 1994.

[16] O. Knüppel. PROFIL/BIAS V 2.0. Technical report, Technische Universität Hamburg-Harburg, 1999.

[17] S. Kong, S. Gao, W. Chen, and E. Clarke. dReach: δ-reachability analysis for hybrid systems. In *Proc. of Tools and Algorithms for the Construction and Analysis of Systems, 200-205*, pages 200–205, 2015.

[18] U. W. Kulisch and W. L. Miranker. *Computer Arithmetic in Theory and Practice*. Academic Press, 1981.

[19] E. A. Lee. CPS foundations. In *Proc. of the 47th Design Automation Conference*, pages 737–742, 2010.

[20] V. Lefèvre, J.-M. Muller, and A. Tisserand. Toward correctly rounded transcendentals. *IEEE Transactions on Computers*, 47(11):1235–1243, 1998.

[21] S. Mitra, T. Wongpiromsarn, and R. M. Murray. Verifying cyber-physical interactions in safety-critical systems. *IEEE Security and Privacy*, 11(4):28–37, 2013.

[22] R. E. Moore, R. B. Kearfott, and M. J. Cloud. *Introduction to Interval Analysis*. Society for Industrial and Applied Mathematics, 2009.

[23] R. Poovendran. Cyber-physical systems: Close encounters between two parallel worlds. *Proceedings of the IEEE*, 98(8):1363–1366, 2010.

[24] H. Ratschek and J. Rokne. *Computer Methods for the Range of Functions*. Halsted Pr, 1984.

[25] S. M. Rump. *Developments in Reliable Computing*, chapter INTLAB - INTerval LABoratory,

pages 77–104. Kluwer Academic Publishers, 1999.

[26] S. M. Rump. Fast and parallel interval arithmetic. *BIT Numerical Mathematics*, 39(3):534–554, 1999.

[27] S. Schupp, E. Ábrahám, X. Chen, I. Ben Makhlouf, G. Frehse, S. Sankaranarayanan, and S. Kowalewski. Current challenges in the verification of hybrid systems. In *Proc. of the Fifth Workshop on Design, Modeling and Evaluation of Cyber Physical Systems*, pages 8–24, 2015.

[28] Sun Microsystems, Inc., 901 San Antonio Road Palo Alto, CA 94303 U.S.A. 650-960-1300. *C++ Interval Arithmetic Programming Reference*, 2001.

[29] J. Sztipanovits, S. Ying, I. Cohen, D. Corman, J. Davis, H. Khurana, P. J. Mosterman, V. Prasad, and L. Stormo. Strategic R&D opportunities for 21st century cyber-physical systems. Technical report, National Institute of Standards and Technology (NIST), 2013.

[30] W. Wolf. Cyber-physical systems. *Computer*, 42(3):88–89, 2009.

[31] J. Zemke. b4m — A free interval arithmetic toolbox for Matlab based on BIAS. Technical report, Arbeitsbereich Technische Informatik III 4-04, Technische Universität Hamburg-Harburg, 1999.