Implementation of Taylor models in CORA 2018 (Tool Presentation)

Matthias Althoff[1], Dmitry Grebenyuk[2], and Niklas Kochdumper[1]

[1] Technische Universität München, Department of Informatics, Munich, Germany
althoff@in.tum.de
niklas.kochdumper@tum.de
[2] University of Melbourne, School of Computing and Information Systems, Melbourne, Australia
dgrebenyuk@student.unimelb.edu.au

Abstract

Computing guaranteed bounds of function outputs when their input variables are bounded by intervals is an essential technique for many formal methods. Due to the importance of bounding function outputs, several techniques have been proposed for this problem, such as interval arithmetic, affine arithmetic, and Taylor models. While all methods provide guaranteed bounds, it is typically unknown to a formal verification tool, which approach is best suitable for a given problem. For this reason, we present an implementation of the aforementioned techniques in CORA—a MATLAB tool—so that advantages and disadvantages of different techniques can be quickly explored without having to compile code. In this work we present the implementation of Taylor models and affine arithmetic; our interval arithmetic implementation has already been published. We evaluate the performance of our implementation using a set of benchmarks against Flow* and INTLAB. To the best of our knowledge, we have also evaluated for the first time how a combination of interval arithmetic and Taylor models performs—our results indicate that this combination is faster and more accurate than only using Taylor models.

1 Introduction

Providing guaranteed bounds of multi-dimensional functions $r = f(x)$ ($f : \mathbb{R}^n \to \mathbb{R}^m$) subject to inputs $x \in [\underline{x}, \overline{x}]$, $\underline{x} \leq \overline{x}$, $\underline{x}, \overline{x} \in \mathbb{R}^n$ bounded by a multi-dimensional interval is essential in many formal verification techniques. For instance, in [41], the error of an approximate solution of ordinary differential equations is obtained from function bounds. In [11, 13, 30, 39], bounds on functions are used to compute reachable sets via Picard iteration or a truncated Lie series in combination with computing guaranteed bounds of functions. The remainder of Taylor series is computed in [3, 20, 21] to abstract arbitrary differential equations to polynomial differential equations for reachability analysis. The Jacobian of linearized systems is bounded in [17] to compute reachable sets. Other approaches for reachability analysis use interval constraint propagation [24, 40]. A combination of Taylor models with SAT solving is presented in [1]. Function values are also bounded for verified runtime validation [33, Sec. 3.2]. Further tools requiring bounds of functions can be found in [5, 12, 43].

Due to the importance of being able to bound values of arbitrary functions $r = f(x)$, many techniques have been developed for this problem. Among them are interval arithmetic [23], generalized interval arithmetic [19], affine arithmetic [15], and Taylor models [8, 16]. Since generalized interval arithmetic and affine arithmetic are conceptually identical [15, Sec. 5], we will only consider affine arithmetic from now on. The most general technique for inputs bounded by intervals are Taylor models since interval arithmetic is a zeroth-order Taylor model and affine arithmetic is a first-order Taylor model when the inputs are bounded by intervals. In general, interval arithmetic is the fastest technique, but it typically results in the largest over-approximation. On the other end are Taylor models, which typically require the most computation time, but typically provide the tightest solutions. However, there exist quite many cases in which interval arithmetic provides tighter results [37]. Affine arithmetic is in most cases a good compromise between interval arithmetic (efficiency) and Taylor models (accuracy).

Taylor models cannot be directly evaluated as it is done in interval arithmetic or affine arithmetic. For this reason, we present the implementation of different techniques to obtain the bounds of Taylor models in CORA [4,6]. Not only is it unclear for a given problem, whether interval arithmetic, affine arithmetic, or Taylor models provide a tighter bound—it is also unknown what Taylor model evaluation technique is best for a given problem [37, Sec. 4]. This motivated us to introduce the method *zoo*, which evaluates several techniques in parallel and then intersects all results. While this method is quite trivial, we obtained superior results and can show that a mix of methods is faster and more precise than using Taylor models with high accuracy, which are slower and less precise in our tested examples. Our evaluation is performed on examples taken from [22] to remove bias from choosing the examples considered. We also compare our results with the established tools Flow* [11] for Taylor models and INTLAB [42] for interval arithmetic. Another work that has compared interval arithmetic, affine arithmetic, and Taylor models is [35], but this work investigated whether other useful representations exist theoretically (e.g. a combination of trigonometric functions), without comparing the methods experimentally. A further work compared Taylor models with centered and mean value forms [28], but this work has evaluated the performance on rather small input uncertainties, while e.g. reachability analysis typically evaluates larger input uncertainties.

This paper is organized as follows: Sec. 2 defines Taylor models and how to perform operations on them. Affine arithmetic is presented in Sec. 3. In Sec. 4 we detail our implementation in CORA. We evaluate the implemented methods in Sec. 5 and also compare the performance with Flow* and INTLAB. Sec. 6 provides some final conclusions.

2 Taylor Models

In this section we briefly recall *Taylor models* [8,9,28,30,31]. An earlier development of Taylor models, which was not introduced under that name can be found in [16]. Taylor models are implemented in the tools Flow* [11] and COSY INFINITY [29]. Other tools also use concepts from Taylor models, such as VNODE-LP [34].

To define Taylor models, we first introduce an n-dimensional interval $[x] := [\underline{x}, \overline{x}]$, $\forall i: \quad \underline{x}_i \leq \overline{x}_i$, $\underline{x}, \overline{x} \in \mathbb{R}^n$ and define Taylor polynomials:

Definition 1 (Taylor polynomial)**.** We introduce $P^q(x-x_0)$ as the q-th order Taylor polynomial of $f(x)$ around x_0 $(x, x_0 \in \mathbb{R}^n)$:

$$P^q(x - x_0) = \sum_{l_1=0}^{q} \dots \sum_{l_n=0}^{q} \frac{(x_1 - x_{0,1})^{l_1} \dots (x_n - x_{0,n})^{l_n}}{l_1! \dots l_n!} \left(\frac{\partial^{l_1+\dots+l_n} f}{\partial x_1^{l_1} \dots \partial x_n^{l_n}} \right) \Bigg|_{x=x0} . \qquad (1)$$

Definition 2 (Taylor model)**.** Let $f : \mathbb{R}^n \to \mathbb{R}^m$ be a function that is $(q+1)$ times continuously differentiable in an open set $\mathcal{D} \subset \mathbb{R}^n$ containing the n-dimensional interval $[x]$. Given $P^q(x-x_0)$

as the q-th order Taylor polynomial of $f(x)$ around $x_0 \in [x]$, we choose an n-dimensional interval $[I]$ such that

$$\forall x \in [x]: \quad f(x) \in P^q(x - x_0) + [I]. \tag{2}$$

The pair $T = (P^q(x - x_0), I)$ is called an q-th order Taylor model of $f(x)$ around x_0.

From now we use the shorthand notation (P, I) and we omit q, x, and x_0 when it is self-evident. Further information can be found in [31, Sec. 2] and [28, p. 384]. An illustration of a fourth-order Taylor model is shown in Fig. 1 for $r = \cos(x)$ and the range $[x] = [-\pi/3, \pi/2]$.

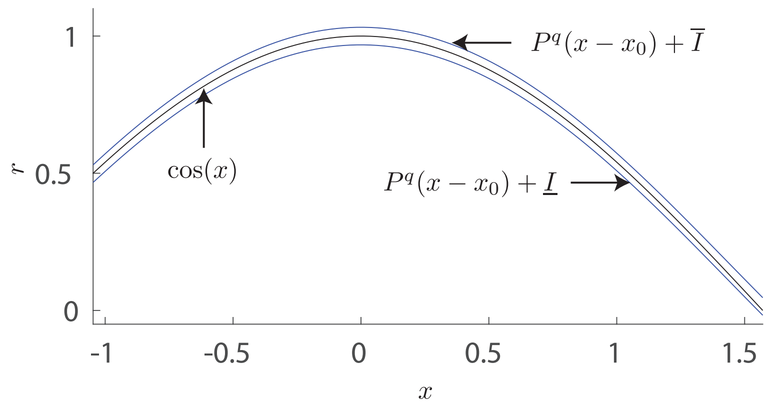


Figure 1: Fourth-order Taylor model for $\cos(x)$ with $[x] = [-\pi/3, \pi/2]$.

2.1 Scalar Operations

Since Taylor models are essentially polynomials with an interval uncertainty, one requires interval arithmetic to perform operations on Taylor models. We briefly recall interval arithmetic and refer to [6] for information on its implementation in CORA. Binary operations typically required for scalar intervals are addition, subtraction, multiplication, and division:

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

Similarly, bounds on standard functions, such as $\sin([x])$ can be obtained [6]. Interval arithmetic treats each variable as an independent entity, causing over-approximative results. For instance, the intervals in the expression $[x] - [x]$ are treated independently as

$$[x] - [x] = \{x_1 \in [x]\} - \{x_2 \in [x]\} = [\underline{x} - \overline{x}, \overline{x} - \underline{x}] \neq \{0\}. \tag{4}$$

This phenomenon is often referred to as the dependency problem [23].

Taylor models are particularly designed to address the dependency problem. Due to the use of polynomials, dependent variables are considered by so-called cancellation effects [37]. Let us first introduce P_e as the cut-off part of the polynomial $P_1 \cdot P_2$ of orders $(q+1)$ to $2q$. We also require the operator $B(P^q(x-x_0))$ returning an over-approximation of the exact bounds $[\min_{x\in[x]} P^q(x-x_0), \max_{x\in[x]} P^q(x-x_0)]$. Let us further introduce $c_f = f(x_0)$ and $\tilde{T} = T - c_f$. Addition, subtraction, multiplication, and division are computed for $T_1 = (P_1, I_1)$ and $T_2 = (P_2, I_2)$, and constants $\alpha, \beta \in \mathbb{R}$ as [28, Sec. 2]:

$$\alpha T_1 + \beta T_2 = (\alpha P_1 + \beta P_2,\ \alpha[I_1] + \beta[I_2]), \tag{5}$$

$$\alpha T_1 - \beta T_2 = (\alpha P_1 - \beta P_2,\ \alpha[I_1] - \beta[I_2]), \tag{6}$$

$$T_1 \cdot T_2 = (P_1 \cdot P_2,\ B(P_e) + B(P_1)\cdot I_2 + B(P_2)\cdot I_1 + I_1 \cdot I_2), \tag{7}$$

$$\frac{T_1}{T_2} = T_1 \cdot \Big(\frac{1}{T_2}\Big), \tag{8}$$

$$\frac{1}{T} = \frac{1}{c_f}\Big[1 - \frac{\tilde{T}}{c_f} + \frac{\tilde{T}^2}{c_f^2} \dots + (-1)^q \frac{\tilde{T}^q}{c_f^q}\Big] + (-1)^{q+1} \frac{B(\tilde{T})^{q+1}}{c_f^{q+2}} \frac{1}{\Big(1 + [0,1]\cdot \frac{B(\tilde{T})}{c_f}\Big)^{q+2}}. \tag{9}$$

Please note that $1/T$ as well as $[1]/[y]$ is only defined if the division by zero is excluded. We have also improved the computation of $1/T$ as presented in Appendix B. It is now obvious that the previous example $[x] - [x] = \sum_{i=1}^q a_i x^i - \sum_{i=1}^q a_i x^i = 0$ returns the exact result due to cancellations of polynomials.

Unary operations, such as e^x, $\sin(x)$, etc. are handled similarly. A list of unary operations implemented in CORA can be found in Appendix A. Obviously, interval arithmetic is identical to Taylor models of zeroth order. It remains to obtain the bounds $B(\cdot)$, which is addressed next.

2.2 Conversion of an Interval to a Taylor Model

Converting an interval to a Taylor model is trivial [36, Sec. 2.4]. Let $[a]$ be an interval then the corresponding Taylor model T is

$$T = P(x) + I = 0.5(\overline{a} + \underline{a}) + 0.5(\overline{a} - \underline{a})\,x + [0,0],\ \text{where } x \in [-1,1]. \tag{10}$$

Using the above conversion with $x \in [-1,1]$ has the effect that all operations result in Taylor models whose variables are also bounded by $[-1,1]$ (ranges of variables are not affected by operations on them). This does not only simplify the implementation since ranges do not have to be stored, but also provides tighter over-approximations (see Sec. 2.3.1). The positive effects of a so-called centered form or mean-value form have been well studied, but mostly for polynomials of order 1 [2,25]. From now on, we refer to Taylor models with $[x] = [-\mathbf{1}, \mathbf{1}]$, where $\mathbf{1}$ is a vector of ones of proper dimension, as *normalized Taylor models*.

2.3 Over-approximation of Bounds

In contrast to interval arithmetic and affine arithmetic, Taylor models do not directly provide a range of possible values. Let us first introduce the operator $B(f) := [\min_{x\in[x]} f(x), \max_{x\in[x]} f(x)]$.

The bounds of a Taylor model T with a polynomial P and interval $[I]$ can be over-approximated as

$$B(T) = B(P) + [I].$$

Several approaches to obtain $B(P)$ exist, out of which *interval arithmetic*, *branch and bound*, and the *LDB/QFB algorithm* are currently implemented in CORA.

2.3.1 Interval Arithmetic

Obviously, one can obtain the bounds of a polynomial by interval arithmetic, which is demonstrated for a polynomial $P(x, y)$ with two arguments:

$$B\big(P(x,y)\big) = \left\{ P(x,y) = \sum_{i=0}^{n}\sum_{j=0}^{n} c_{i,j} x^i y^j \middle| x \in [\underline{x}, \overline{x}], y \in [\underline{y}, \overline{y}] \right\} \subseteq \sum_{i=0}^{n}\sum_{j=0}^{n} c_{i,j} [\underline{x}, \overline{x}]^i [\underline{y}, \overline{y}]^j. \quad (11)$$

This approach works particularly well for monotone polynomials. To obtain better results, we compute the bounds from interval arithmetic using the normalized form as demonstrated by the next example.

Example 1 (Evaluation of original and normalized Taylor model)**.** Let us consider the function $r = 0.1x^3 - 0.5x^2 + 1$ and the range $[x] = [0, 6]$. The normalized function for $[\tilde{x}] = [-1, 1]$ is $\tilde{r} = 2.7\tilde{x}^3 + 3.6\tilde{x}^2 - 0.9\tilde{x} - 0.8$. Evaluating the original function and the normalized function with interval arithmetic results in

$$[r] = 0.1[0, 6]^3 - 0.5[0, 6]^2 + 1 = [-17.0, 22.6],$$
$$[\tilde{r}] = 2.7[-1, 1]^3 + 3.6[-1, 1]^2 - 0.9[-1, 1] - 0.8 = [-4.4, 6.4],$$

showing that the normalized form clearly outperforms the original form in this example.

2.3.2 Branch and Bound

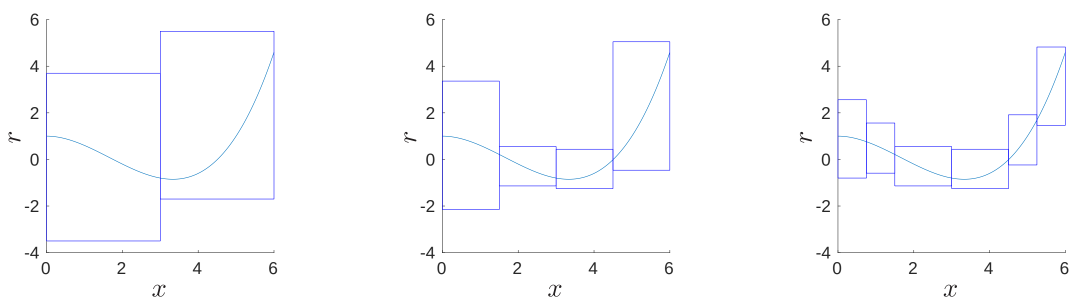
For most polynomials, interval arithmetic provides too conservative results. Finding guaranteed bounds is the goal of global optimization techniques [18]. Techniques which can efficiently obtain a global minimum or maximum, such as convex optimization or geometric programming, cannot be applied to general polynomials. Therefore, we have implemented a branch and bound algorithm [7, 27] in CORA.

Our implementation of a branch and bound algorithm is shown in Alg. 1, which evaluates each coordinate of the output $r = f(x)$ individually. In line 2-5 we first compute the bound using interval arithmetic. We use the short form $P([x]) = \{P(x)|x \in [x]\}$; in Alg. 1 it is assumed that $P : \mathbb{R}^n \to \mathbb{R}$ due to evaluating each coordinate individually. In branch and bound approaches, one refines the inputs; in our case we split the input intervals in sub-intervals $[x^{(k)}]$ resulting in corresponding sub-intervals of outputs $[r^{(k)}]$. The index $\overline{\alpha}$ of the input intervals $[x^{(\overline{\alpha})}]$ resulting in the output intervals $[r^{(\overline{\alpha})}]$ with the largest upper bound, and the index $\underline{\alpha}$ resulting in the smallest lower bound are determined in line 8 and line 9, respectively. If $\overline{\alpha} = \underline{\alpha}$ (which is typically only the case when $[x]$ has not yet been split), we split the coordinate of $[x^{(k)}]$ with the largest radius (line 11-16). Otherwise, the intervals $[x^{(\overline{\alpha})}]$ and $[x^{(\underline{\alpha})}]$ are also split in the coordinate with the largest radius (line 18-29). If the lower and upper bound of the overall range $[r_\cup] \leftarrow \bigcup_i [r^{(i)}]$ (line 30) do not change more than $2\epsilon \cdot \mathrm{rad}([r_\cup])$ (line 31), the final range

$[r_\cup]$ is returned. If we set $\epsilon \to \infty$, the branch and bound technique becomes identical to just using interval arithmetic.

In the lines 15, 22 and 28, we re-expand the original Taylor model P (using the operator `reexp`$(\cdot)$) at the midpoints of the newly generated sub-intervals before the bounds are calculated. This re-expansion leads to tighter bounds, but can also be computational expensive, especially for Taylor models with high polynomial order and a large number of variables. Due to this trade-off between computation time and precision, CORA offers the two algorithm variants *standard* (without re-expansion) and *advanced* (with re-expansion).

Example 2 (Branch and bound on a non-monotonic function)**.** We consider the non-monotonic function $r = 0.1x^3 - 0.5x^2 + 1$. When applying Alg. 1, the upper and lower bound are reduced in each iteration as shown in Fig. 2.



(a) 1^{st} iteration (b) 2^{nd} iteration (c) 3^{rd} iteration

Figure 2: Improvements of bounding a polynomial using branch and bound for the first three iterations of Alg. 1. The improvements of $[r]$ over the iterations are $[-3.50, 5.50]$, $[-2.15, 5.05]$, $[-1.25, 1.91]$ (values are rounded). We have used the setting *standard*.

2.3.3 Linear Dominated Bounder and Quadratic Fast Bounder

In addition to the branch and bound algorithm described in the previous subsection, we have also implemented an algorithm for bounding values that is based on the Linear Dominated Bounder (LDB) and the Quadratic Fast Bounder (QFB) presented in [32]. Similar to branch and bound, this algorithm is based on the iterative splitting of the input interval into smaller sub-intervals. The algorithm then iteratively eliminates all sub-intervals that are guaranteed to not contain the minimum and maximum of the polynomial, until only one sub-interval is left for the minimum, and one for the maximum. For this, a linear and a quadratic approximation of the Taylor model is used to determine the sub-intervals that are most likely to contain the minimum and the maximum, which significantly speeds up the search. With this overall concept it is possible to determine the real minimum and maximum of a polynomial up to a user-defined precision ϵ. Please note that due to the interval remainder part of the Taylor model, the difference between the real minimum and maximum of the function $r = f(x)$ and the calculated bounds can still be much larger than ϵ.

Algorithm 1 Branch and bound

procedure BRANCHANDBOUND($P, [x], \epsilon$)
  $k = 1$
  $[x]^{(k)} \leftarrow [x]$
  $[r^{(k)}] \leftarrow P([x^{(k)}])$ ▷ obtain bounds as in (11)
  $[r_\cup] \leftarrow [r^{(k)}]$ ▷ initialize union of all intervals
  repeat
    $[r_{\cup,prev}] \leftarrow [r_\cup]$ ▷ update previous union of all intervals
    $\overline{\alpha} \leftarrow \text{argmax}_k(\overline{r}^{(k)})$ ▷ index of interval containing the upper bound
    $\underline{\alpha} \leftarrow \text{argmax}_k(\underline{r}^{(k)})$ ▷ index of interval containing the lower bound
    if $\overline{\alpha} == \underline{\alpha}$ **then**
      $\beta \leftarrow \text{argmax}_i(\text{rad}([x]_i^{(\overline{\alpha})}))$ ▷ return coordinate with the largest radius
      $k = k + 1$
      $[x^{(\overline{\alpha},fh)}] \leftarrow [x^{(\overline{\alpha})}]$, $[x_\beta^{(\overline{\alpha},fh)}] \leftarrow [\underline{x}_\beta^{(\overline{\alpha})}, \overline{x}_\beta^{(\overline{\alpha})} - \texttt{rad}([x_\beta^{(\overline{\alpha})}])]$ ▷ first half of $[x^{(\overline{\alpha})}]$
      $[x^{(\overline{\alpha})}] \leftarrow [x^{(\overline{\alpha},fh)}]$, $[x^{(k)}] \leftarrow [x^{(\overline{\alpha})}] \setminus [x^{(\overline{\alpha},fh)}]$ ▷ split of $[x^{(\overline{\alpha})}]$
      $P_1 \leftarrow \texttt{reexp}(P, \texttt{mid}([x^{(\overline{\alpha})}])$, $P_2 \leftarrow \texttt{reexp}(P, \texttt{mid}([x^{(k)}])$ ▷ adapt expansion point
      $[r^{(\overline{\alpha})}] \leftarrow P_1([x^{(\overline{\alpha})}])$, $[r^{(k)}] \leftarrow P_2([x^{(k)}])$ ▷ update bounds as in (11)
    else
      $\beta \leftarrow \text{argmax}_i(\text{rad}([x]_i^{(\overline{\alpha})}))$ ▷ return coordinate with the largest radius
      $k = k + 1$
      $[x^{(\overline{\alpha},fh)}] \leftarrow [x^{(\overline{\alpha})}]$, $[x_\beta^{(\overline{\alpha},fh)}] \leftarrow [\underline{x}_\beta^{(\overline{\alpha})}, \overline{x}_\beta^{(\overline{\alpha})} - \texttt{rad}([x_\beta^{(\overline{\alpha})}])]$ ▷ first half of $[x^{(\overline{\alpha})}]$
      $[x^{(\overline{\alpha})}] \leftarrow [x^{(\overline{\alpha},fh)}]$, $[x^{(k)}] \leftarrow [x^{(\overline{\alpha})}] \setminus [x^{(\overline{\alpha},fh)}]$ ▷ split of $[x^{(\overline{\alpha})}]$
      $P_1 \leftarrow \texttt{reexp}(P, \texttt{mid}([x^{(\overline{\alpha})}])$, $P_2 \leftarrow \texttt{reexp}(P, \texttt{mid}([x^{(k)}])$ ▷ adapt expansion point
      $[r^{(\overline{\alpha})}] \leftarrow P_1([x^{(\overline{\alpha})}])$, $[r^{(k)}] \leftarrow P_2([x^{(k)}])$ ▷ update bounds as in (11)
      $\beta \leftarrow \text{argmax}_i(\text{rad}([x]_i^{(\underline{\alpha})}))$ ▷ return coordinate with the largest radius
      $k = k + 1$
      $[x^{(\underline{\alpha},fh)}] \leftarrow [x^{(\underline{\alpha})}]$, $[x_\beta^{(\underline{\alpha},fh)}] \leftarrow [\underline{x}_\beta^{(\underline{\alpha})}, \underline{x}_\beta^{(\underline{\alpha})} - \texttt{rad}([x_\beta^{(\underline{\alpha})}])]$ ▷ first half of $[x^{(\underline{\alpha})}]$
      $[x^{(\underline{\alpha})}] \leftarrow [x^{(\underline{\alpha},fh)}]$, $[x^{(k)}] \leftarrow [x^{(\underline{\alpha})}] \setminus [x^{(\underline{\alpha},fh)}]$ ▷ split of $[x^{(\underline{\alpha})}]$
      $P_1 \leftarrow \texttt{reexp}(P, \texttt{mid}([x^{(\underline{\alpha})}])$, $P_2 \leftarrow \texttt{reexp}(P, \texttt{mid}([x^{(k)}])$ ▷ adapt expansion point
      $[r^{(\underline{\alpha})}] \leftarrow P_1([x^{(\underline{\alpha})}])$, $[r^{(k)}] \leftarrow P_2([x^{(k)}])$ ▷ update bounds as in (11)
    $[r_\cup] \leftarrow \bigcup_i [r^{(i)}]$ ▷ update union of all intervals
  until $\overline{r}_{\cup,prev} - \overline{r}_\cup <= 2\epsilon \cdot \text{rad}([r_\cup])$ & $\underline{r}_{\cup,prev} - \underline{r}_\cup <= 2\epsilon \cdot \text{rad}([r_\cup])$ ▷ rad($\cdot$) as in (10)
  return $[r_\cup]$

2.4 Matrix Operations

This section addresses how CORA handles vectors and matrices represented by Taylor models, where a vector is defined as a matrix with a single row or column (a one dimensional tensor). A matrix Taylor model is a matrix in which each element is represented by a Taylor model. The operations of transpose, addition, subtraction, multiplication, and division by a scalar, are defined in a standard way. The aforementioned operations can be broken down to addition and multiplication of scalar Taylor models, which are realized as presented in Sec. 2.1. All unary operations are realized element-wise as

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

so that we again resort to the implementation described in Sec. 2.1.

One exception is the computation of the inverse of a matrix Taylor model. Although in principle, one could obtain the inverse of a matrix symbolically, e.g.

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, \quad A^{-1} = \frac{1}{a_{11}a_{22} - a_{12}a_{21}} \begin{bmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{bmatrix},$$

this approach is infeasible for larger matrices. For this reason we use the approach in [45], which is based on the Sherman-Morrison (SM) formula (see [38, Sec. 2.7.1])

$$(A + B)^{-1} = A^{-1} + \frac{1}{1 + \mathrm{tr}(A^{-1}B)}(A^{-1}BA^{-1}), \tag{12}$$

where $\mathrm{tr}(\cdot)$ returns the trace of a matrix. The formula holds when B has unitary rank. By splitting a Taylor model in a constant part and a remaining part, we obtain $T = (P^q(x - x_0), I) = (c_f, [\mathbf{0}]) + (\tilde{P}^q(x - x_0), I) = c_f + \tilde{T}$, where $\tilde{P}^q(x - x_0) = P^q(x - x_0) - c_f$. When $\tilde{T} = T - c_f$ has unitary rank, we can compute

$$(c_f + \tilde{T})^{-1} = c_f^{-1} + \frac{1}{1 + \mathrm{tr}(c_f^{-1}\tilde{T})}(c_f^{-1}\tilde{T}c_f^{-1}) \tag{13}$$

using only methods from Sec. 2.1. However, in most cases, $\tilde{T}$ has full rank so that [45] splits the matrix $\tilde{T}$ into the sum of rank-one matrices $\tilde{T} = \sum_{i=1}^{n} \tilde{T}_i$, where $\tilde{T}_i$ is a matrix whose i^{th} column coincides with $\tilde{T}$ and all other entries are zero. One first applies (13) for $\tilde{T}_1$ and c_f. Next, $(c_f + \tilde{T}_1 + \tilde{T}_2)^{-1}$ is calculated choosing $A = c_f + \tilde{T}_1$ and $B = \tilde{T}_2$ in (12); the inverse of A is available from the previous step. This is repeated until the final matrix $\tilde{T}_n$ has been considered.

3 Affine Arithmetic

Affine arithmetic uses affine forms, i.e. first-order polynomials consisting of a vector $x \in \mathbb{R}^n$ and noise symbols $\epsilon_i \in [-1, 1]$ (see e.g. [15]):

$$\hat{x} = x_0 + \epsilon_1 x_1 + \epsilon_2 x_2 + \ldots + \epsilon_p x_p.$$

The possible values of $\hat{x}$ lie within a zonotope [26]. In contrast to Taylor models, it is possible that $p > n$ so that affine forms are not a special case of Taylor models. This is the same reason why polynomial zonotopes are different from Taylor models, since polynomial zonotopes are a generalization of zonotopes [3].

However, in this work, we only consider intervals as inputs and outputs so that Taylor models of first order can implement an affine arithmetic. Obviously, other methods for realizing an affine arithmetic exist, e.g. Chebyshev (minimax) approximations or the min-range approximation [44, Sec. 3].

4 Implementation in CORA

Taylor models are implemented in CORA by the class `taylm`. To make use of cancellation effects, we have to provide names for variables in order to recognize identical variables; this is different to implementations of interval arithmetic, where each variable is treated individually as demonstrated in (4). We have realized three primal ways to generate a matrix containing Taylor models.

Method 1: Composition from scalar Taylor models. The first possibility is to generate scalar Taylor models from intervals as shown subsequently.

```
a1 = interval(-1, 2); % generate a scalar interval [-1,2]
b1 = taylm(a1, 6);  % generate a scalar Taylor model of order 6
a2 = interval(2, 3);  % generate a scalar interval [2,3]
b2 = taylm(a2, 6);  % generate a scalar Taylor model of order 6
c = [b1; b2]           % generate a row of Taylor models
```

When a scalar Taylor model is generated from a scalar interval, the name of the variable is deduced from the name of the interval. If one wishes to overwrite the name of a variable `a2` to `c`, one can use the command `taylm(a2, 6, {'c'})`.

Method 2: Converting an Interval Matrix. One can also first generate an interval matrix, i.e., a matrix containing intervals and then convert the interval matrix into a Taylor model. The subsequent example generates the same Taylor model as in the previous example.

```
a = interval([-1;2], [2;3]);  % generate an interval vector [[-1,2]; [2,3]]
c = taylm(a, 6, {'a1';'a2'})  % generate a Taylor model (order 6) with variable names a1 and a2
```

Note that the cell for naming variables `{'a1';'a2'}` has to have the same dimensions as the interval matrix `a`. If no names are provided, default names are automatically generated.

Method 3: Symbolic expressions. We also provide the possibility to create a Taylor model from a symbolic expression.

```
syms a1 a2; % instantiate symbolic variables
s = [2 + 1.5*a1; 2.75 + 0.25*a2]; % create symbolic function
c = taylm(s, interval([-2;-3],[0;1]), 6)  % generate Taylor model
```

This method does not requiring to name variables since variable names are taken from the variable names of the symbolic expression. The interval of possible values has to be specified after the symbolic expression s; here: $[[-2,0]\,[-3,1]]^T$.

All examples generate a row vector c as shown in Sec. 2.2. Since all variables are normalized to the range $[-1,1]$ according to (10), we obtain

$$c = \begin{bmatrix} 0.5 + 1.5 \cdot \tilde{a}_1 + [0,0] \\ 2.5 + 0.5 \cdot \tilde{a}_2 + [0,0] \end{bmatrix} .$$

The following workspace output of MATLAB demonstrates how the dependency problem is considered by keeping track of all encountered variables:

```
>> c(1) + c(1)
ans =
        1.0 + 3.0*a1 + [0.00000,0.00000]

>> c(1) + c(2)
ans =
        3.0 + 1.5*a1 + 0.5*a2 + [0.00000,0.00000]
```

4.1 Implementation of Polynomials

Competitive runtimes of Taylor model implementations can only be realized by efficient addition and multiplication of polynomials, since these two base operations are repeatedly used as shown in Appendix A. To this end, we use a concept similar to the MONOMIAL Toolbox[1]. A monomial is defined as a product of powers of variables with nonnegative integer exponents, e.g., $x_1^2 x_2^3 x_3$. We encode polynomials as tuples with three entries $P(x) = \{c, E, id\}$, where $c \in \mathbb{R}^d$ is a vector storing the coefficients for each monomial term of the polynomial, $E \in \mathbb{R}^{n \times d}$ is a matrix that stores the exponents of the monomials, and $id \in S^n$ is a list of strings S that stores the identifiers for all variables that are part of the polynomial. In the above definitions, d represents the number of monomial terms in the polynomial, and n is the number of variables. Let us consider the following polynomial $p(x)$ that consists of $d = 4$ monomial terms and has $n = 3$ variables as a demonstrating example:

$$p(x) = 1.3 + 3.4x_1^2 x_2 - 2.3x_1^3 x_4^2 + 4.5x_1^3 x_2^2 x_4^3 \tag{14}$$

This polynomial is represented in CORA as:

$$P(x) = \{c, E, id\}\,, \text{ with } \ c = \begin{pmatrix} 1.3 \\ 3.4 \\ -2.3 \\ 4.5 \end{pmatrix}, \ E = \begin{pmatrix} 0 & 2 & 3 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 2 & 3 \end{pmatrix}, \ id = \begin{pmatrix} "x_1" \\ "x_2" \\ "x_4" \end{pmatrix}. \tag{15}$$

Before two polynomials can be added or multiplied, we have to ensure that the identification vectors of both polynomials are identical. The first step is therefore to determine a common identification vector id_{new} from the vectors id_1 and id_2 of the two polynomials. After this, the exponent matrices E_1 and E_2 have to be transformed to this new common representation, which can be easily realized by commutation of matrix rows and insertion of new all-zero rows. We demonstrate this concept with the example of two polynomials $P_1(x)$ and $P_2(x)$:

$$\begin{aligned} &P_1(x) = \{c_1, E_1, id_1\}, \quad \text{with } c_1 = \begin{pmatrix} 2.3 \\ 3.1 \\ -2.7 \end{pmatrix}, \ E_1 = \begin{pmatrix} 0 & 1 & 4 \\ 1 & 2 & 2 \end{pmatrix}, \ id_1 = \begin{pmatrix} "x_2" \\ "x_5" \end{pmatrix}, \\ &P_2(x) = \{c_2, E_2, id_2\}, \quad \text{with } c_2 = \begin{pmatrix} -1.1 \\ 5.2 \end{pmatrix}, \ E_2 = \begin{pmatrix} 0 & 3 \end{pmatrix}, \ id_2 = \begin{pmatrix} "x_3" \end{pmatrix}. \end{aligned} \tag{16}$$

[1]MONOMIAL is a MATLAB toolbox for multivariate polynomials. The toolbox is available at https://people.sc.fsu.edu/~jburkardt/m_src/monomial/monomial.html

The transformation of these two polynomials to a common representation yields the following result:

$$\begin{aligned} P_1(x) &= \{c_1, E_1, id_1\}, \quad \text{with } c_1 = \begin{pmatrix} 2.3 \\ 3.1 \\ -2.7 \end{pmatrix}, \ E_1 = \begin{pmatrix} 0 & 1 & 4 \\ 0 & 0 & 0 \\ 1 & 2 & 2 \end{pmatrix}, \ id_1 = \begin{pmatrix} "x_2" \\ "x_3" \\ "x_5" \end{pmatrix}, \\ P_2(x) &= \{c_2, E_2, id_2\}, \quad \text{with } c_2 = \begin{pmatrix} -1.1 \\ 5.2 \end{pmatrix}, \ E_2 = \begin{pmatrix} 0 & 0 \\ 0 & 3 \\ 0 & 0 \end{pmatrix}, \ id_2 = \begin{pmatrix} "x_2" \\ "x_3" \\ "x_5" \end{pmatrix}. \end{aligned} \tag{17}$$

After the polynomials have been transformed to a common representation, addition is realized with a simple concatenation $||$ of the coefficient vectors $c_{new} = c_1||c_2$ and the exponent matrices $E_{new} = E_1||E_2$. The representation of the resulting polynomial $P_{new}(x)$ possibly contains redundant information, because there could be multiple monomials with identical exponents. In order to obtain a compact representation, these monomials have to be combined to a single monomial by adding their respective coefficients. In order to find all monomials that have identical exponents, we first sort the columns of the exponent according to the graded lexicographic order [14]. After that, we only have to compare neighboring columns in the sorted exponent matrix to determine all monomials with identical exponents. Note that the sorting of the exponent matrix can in general be costly, since the number of variables n in the Taylor model can be large. To speed up the computations, we pre-sort the matrix according to a scalar hash value $h(e_i)$, where e_i is the i-th matrix column and $h(\cdot)$ is a hash function. By using this pre-sorting, only columns with identical hash values have to be further sorted according to the entries in the matrix rows. Note that this concept implements a Bucket Sort algorithm [10], where each bucket contains elements with identical hash values. In order to get a low number of hash function collisions for matrix columns that are not identical, we choose the hash function $h(e_i) = (1\ 1 \dots 1) \cdot e_i$ for our Taylor model implementation, which is identical to the sum of entries in the column e_i.

The multiplication of two polynomials can be implemented similar like the addition, where all $d_{new} = d_1 \cdot d_2$ monomial terms of the resulting polynomial have to be generated by multiplication of the respective coefficients in combination with an addition of the corresponding exponent matrix columns. Afterwards, all monomials with identical exponents have to be combined in order to obtain a compact representation, as it was already described above.

4.2 List of the Functions of the Class `taylm`

In this subsection we list 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. Tab. 1 lists the implemented mathematical functions and Tab. 2 shows auxiliary functions, which are required to e.g. specify Taylor models, arrange them in matrices, access values, perform set operations, and display results, among others. Examples of using some of the listed functions in MATLAB are provided in Appendix C.

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

name	description
`acos`	$\arccos(\cdot)$ function as defined in (31)
`asin`	$\arcsin(\cdot)$ function as defined in (30)
`atan`	$\arctan(\cdot)$ function as defined in (32)
`cos`	$\cos(\cdot)$ function as defined in (24)
`cosh`	$\cosh(\cdot)$function as defined in (28)
`exp`	exponential function as defined in (20)
`interval`	various implementations of the bound operator $B(\cdot)$ as presented in Sec. 2.3
`log`	natural logarithm function as defined in (21)
`minus`	overloaded - operator, see (6)
`mpower`	overloaded ˆ operator (power)
`mrdivide`	overloaded / operator (division), see (8)
`mtimes`	overloaded * operator (multiplication), see (7) for scalars and (Sec. 2.4) for matrices
`plus`	overloaded + operator (addition), see (5) for scalars and (Sec. 2.4) for matrices
`power`	overloaded .ˆ operator (elementwise power)
`rdivide`	overloads the ./ operator: provides elementwise division of two matrices
`reexpand`	re-expand the taylor model at a new expansion point
`sin`	$\sin(\cdot)$ function as defined in (23)
`sinh`	$\sinh(\cdot)$ function as defined in (27)
`sqrt`	$\sqrt{(\cdot)}$ function as defined in (22)
`tan`	$\tan(\cdot)$ function as defined in (26)
`tanh`	$\tanh(\cdot)$ function as defined in (29)
`times`	overloaded .* operator for elementwise multiplication of matrices
`trace`	trace of a taylm matrix
`uminus`	overloaded - operator for a single operand
`uplus`	overloaded + operator for a single operand

Table 2: Auxiliary methods of the class `taylm`.

name	description
`display`	displays the values of a `taylm` object in the MATLAB workspace
`getCoef`	returns the array of polynomial coefficients of a `taylm` object
`getRem`	returns the interval part of a `taylm` object
`getSyms`	returns the polynomial part of a `taylm` object as a symbolic expression
`optBnb`	implementation of the branch and bound algorithm as presented in Sec. 2.3.2
`optBnbAdv`	implementation of the advanced branch and bound algorithm as presented in Sec. 2.3.2
`optLinQuad`	implementation of the algorithm based on LDB and QFB as presented in Sec. 2.3.3
`horzcat`	overloads the operator for horizontal concatenation, e.g. `a = [b, c, d]`
`set`	set the additional class parameters (see Sec. 4.3)
`setName`	set the names of the variables in the Taylor model
`subsasgn`	overloads the operator that assigns elements of a taylm 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 a taylm matrix `I`, e.g. `value = I(1,2)`, where the element of the first row and second column is read
`taylm`	constructor of the `taylm` class
`vertcat`	overloads the operator for vertical concatenation, e.g. `a = [b; c; d]`

4.3 Additional parameters for the Class `taylm`

CORA's Taylor model implementation contains some additional parameters which can be modified by the user:

- **max_order:** Maximum polynomial degree of the monomials in the polynomial part of the Taylor model. Monomials with a degree larger than **max_order** are bounded with the bounding operator $B(\cdot)$ and added to the interval remainder. Further, $q =$ **max_order** is used for the implementation of the formulas listed in Appendix A.

- **tolerance:** Minimum absolute value of the monomial coefficients in the polynomial part of the Taylor model. Monomials with a coefficient whose absolute value is smaller than **tolerance** are bounded with the bounding operator $B(\cdot)$ and added to the interval remainder.

- **eps:** Termination tolerance ϵ for the branch and bound algorithm from Section 2.3.2 and for the algorithm based on the Linear Dominated Bounder and the Quadratic Fast Bounder from Section 2.3.3.

These parameters are stored as properties of the class `taylm`. In the functions `plus`, `minus` and `times`, two Taylor models are combined to one resulting Taylor model object. Here, we implemented the following rules to determine the values of the above parameters for the resulting object:

$$\begin{aligned} &\text{max\_order}_{new} = max\{\text{max\_order}_1, \text{max\_order}_2\}, \\ &\text{tolerance}_{new} = min\{\text{tolerance}_1, \text{tolerance}_2\}, \\ &\text{eps}_{new} = min\{\text{eps}_1, \text{eps}_2\}, \end{aligned} \tag{18}$$

where index *new* refers to the resulting object, and the indices 1 and 2 to the initial objects.

4.4 Class `zoo`

As later shown in Sec. 5, it is often better to use several simple range bounding methods in parallel and intersect the results. For instance, it is advantageous for most range bounding problems to combine interval arithmetic with Taylor models of order 3 rather than only applying one accurate method, such as using Taylor models of order 6. On average this strategy is more accurate and faster to compute.

To facilitate mixing different techniques, we have created the class `zoo` in which one can specify the methods to be combined. All methods implemented for Taylor models in Tab. 1 and Tab. 2 are also available for the class `zoo`.

For the initialization of the `zoo` object, the different techniques that should be used for range bounding can be specified as a list of strings. Currently, the following techniques are available:

- **interval:** Interval arithmetic.

- **affine(int):** Affine arithmetic; the bounds $B(\cdot)$ are calculated with interval arithmetic.

- **affine(bnb):** Affine arithmetic; the bounds $B(\cdot)$ are calculated with the branch and bound algorithm (see Sec. 2.3.2).

- **affine(bnbAdv):** Affine arithmetic; the bounds $B(\cdot)$ are calculated with the advanced branch and bound algorithm (see Sec. 2.3.2).

- **taylm(int):** Taylor models; the bounds $B(\cdot)$ are calculated with interval arithmetic.

- **taylm(bnb):** Taylor models; the bounds $B(\cdot)$ are calculated with the branch and bound algorithm (see Sec. 2.3.2).
- **taylm(bnbAdv):** Taylor models; the bounds $B(\cdot)$ are calculated with the advanced branch and bound algorithm (see Sec. 2.3.2).
- **taylm(linQuad):** Taylor models; the bounds $B(\cdot)$ are calculated with the algorithm that is based on LDB and QFB (see Sec. 2.3.3).

In addition to the bounding techniques, the additional parameters from Section 4.3 as well as the names of the variables can be specified as optional input arguments. The bounds of the `zoo` object can be computed with the `interval` operator, which intersects the intervals obtained by all specified techniques.

```
int = interval(1, 2); % generate a scalar interval [1,2]
methods = {'interval','taylm(bnb)','affine(int)'}; % range bounding techniques
max_order = 10; % max. polynomial order of the Taylor models
eps = 0.001; % termination tolerance for the branch and bound algorithm
tolerance = 1e-8; % minimum absolute value for monomial coefficients

z = zoo(int,methods,'x',max_order,eps,tolerance); % generate the zoo-object
s_z = sin(z); % calculate the sine of the zoo-object
r = interval(s_z) % calculate the bounds
```

The above code produces the following output:

```
r =
        [0.84147,1.00000]
```

5 Numerical Experiments

In this section, we compare two aspects: a) we compare the performance of Taylor models implemented in CORA with other tools and b) compare the accuracy of Taylor model computations with interval arithmetic, affine arithmetic, and a combination of techniques. We first start with a few motivating examples, which highlight advantages and disadvantages of the investigated techniques.

5.1 Motivating Examples

We first motivate the study of complementary techniques by presenting examples for which different techniques provide the best result.

Example 3 (Example in favor of interval arithmetic)**.** Let us obtain the bound $[r] = \sin([x])$, $[x] = [0, \frac{\pi}{2}]$.
Interval arithmetic: Since $\sin(x)$ is monotone on $[0, \frac{\pi}{2}]$, interval arithmetic returns (see [6, eq. (12)])

$$[r] = [\sin(0), \sin(\frac{\pi}{2})] = [0, 1],$$

which is the exact result.
Affine arithmetic: The affine form of $[x]$ is $[x] = \frac{\pi}{4} + \epsilon\frac{\pi}{4}$. To use (23), we first require $x_0 =$

$0.5(\overline{x}+\underline{x}) = \frac{\pi}{4}$, $c_f = x_0 = \frac{\pi}{4}$, and $\tilde{T} = x - c_f$. Using (23) we obtain

$$\begin{aligned}
[r] =& \sin(c_f) + \cos(c_f)\tilde{T} - \frac{1}{2}B(\tilde{T})^2 \sin(c_f + [0,1]B(\tilde{T}))\\
=& \sin(\pi/4) + \cos(\pi/4)(x - \pi/4)\\
&- \frac{1}{2}B(x - \sin(\pi/4))^2 \sin(\pi/4 + [0,1]B(x - \pi/4))\\
=& \sin(\pi/4) - \cos(\pi/4)\sin(\pi/4) + \cos(\pi/4)x\\
&- \frac{1}{2}([0,\pi/2] - \pi/4)^2 \sin(\pi/4 + [0,1]([0,\pi/2] - \pi/4))\\
=& 0.1517 + 0.7071x - [0.00000, \frac{\pi^2}{32}]\\
=& [-0.1567, 1.2624].
\end{aligned}$$

Taylor model: We use a Taylor model of third order. Due to the length of the computation, we do not present it and only provide the output interval using interval arithmetic (see Sec. 2.3.1): $[r] = [-0.1234, 1.3354]$.

Example 4 (Example in favor of affine arithmetic)**.** The same example $r = \sqrt{x^3 - x}$ with $[x] = [2,3]$ is best bounded via affine arithmetic.
Interval arithmetic: After inserting the intervals, interval arithmetic returns (see [6, eq. (5)])

$$[r] = \sqrt{[2,3]^3 - [2,3]} = \sqrt{[5,25]} = [\sqrt{5}, 5] = [2.2361, 5.0].$$

Affine arithmetic: The affine form of $[x]$ is $[x] = 2.5 + \epsilon 0.5$. Using (22), we obtain

$$\begin{aligned}
[r] =& \sqrt{(2.5 + \epsilon 0.5)^3 - (2.5 + \epsilon 0.5)}\\
=& \sqrt{8.875\epsilon + 13.125 + [0,2]}\\
=& 1.2249\epsilon + 3.6228 + [-1.68727, 0.78250]\\
=& [-1.46404, 6.76670].
\end{aligned}$$

Taylor model: We use a Taylor model of third order; the Taylor model of $[x]$ is $T = 2.5 + 0.5x + [0,0]$. When using (22) and interval arithmetic to bound the polynomial (see Sec. 2.3.1), the result is

$$\begin{aligned}
[r] =& \sqrt{T^3 - T}\\
=& \sqrt{0.125x^3 + 1.875x^2 + 8.875x + 13.125}\\
=& -0.00023256x^3 + 0.051714x^2 + 1.2249x + 3.6228 + [-3.86179, 1.86704]\\
=& [-1.46404, 6.76670].
\end{aligned}$$

Example 5 (Example in favor of Taylor models)**.** We use the same example as in example 2: $r = 0.1x^3 - 0.5x^2 + 1$ and $[x] = [0,6]$.
Interval arithmetic: After inserting the intervals, interval arithmetic returns (see [6, eq. (9)])

$$[r] = 0.1[0,6]^3 - 0.5[0,6]^2 + 1 = 0.1[0,216] - 0.5[0,36] + 1 = [-17.0, 22.6].$$

Affine arithmetic: The affine form of $[x]$ is $[x] = 3 + \epsilon 3$. Since power is realized by multiple times applying multiplication, we only require (5), (6), and (7):

$$\begin{aligned}[r] =&0.1(3 + \epsilon 3)^3 - 0.5(3 + \epsilon 3)^2 + 1\\ =&-0.9\epsilon - 0.8 + [-4.5, 10.8]\\ =&[-6.2, 10.9].\end{aligned}$$

Taylor model: We use a Taylor model of third order. The Taylor model of $[x]$ is $T = 3+3x+[0, 0]$. When using interval arithmetic to bound the polynomial (see Sec. 2.3.1), we obtain

$$\begin{aligned}[r] =&0.1\,T^3 - 0.5\,T^2 + 1\\ =&0.1(27x^3 + 81x^2 + 81x + 27) - 0.5(9x^2 + 18x + 9) + 1\\ =&2.7x^3 + 3.6 * x^2 - 0.9x - 0.8\\ =&[-4.4, 6.4].\end{aligned}$$

Next, we compare our results with Flow* and INTLAB.

5.2 Comparison with Flow* and INTLAB

In this subsection, we compare our implementation of Taylor models with Flow* [11] and our implementation of affine arithmetic in Sec. 3 with INTLAB [42]. Flow* and INTLAB both consider rounding errors of floating-point numbers. Since CORA has been designed primarily for prototyping new methods for reachability analysis and for other set-based techniques, CORA is currently not considering floating-point rounding errors.

All computations are performed on an Intel Core i7-7820HQ processor running at 2.9GHz. In order to reduce any possible bias, we have chosen the benchmarks from another work on relative error bounds for floating-point arithmetic [22]. For our results, the relative precision of an interval x is obtained as

$$[\underline{\delta}, \overline{\delta}] = \Big[-\frac{\underline{x} - \underline{x}_{ref}}{\overline{x}_{ref} - \underline{x}_{ref}}, \frac{\overline{x} - \overline{x}_{ref}}{\overline{x}_{ref} - \underline{x}_{ref}} \Big] \cdot 100\%, \tag{19}$$

where x_{ref} is a reference value. This measure of the precision is more informative than the scalar overestimation factor used in [37], because it evaluates the precision of the lower and the upper range bound separately. We used the real bounds of the function for the reference interval x_{ref}. This ensures that $x_{ref} \subseteq x$, so that the relative precision $\delta \geq 0$. Large values for δ therefore correspond to a large over-approximation, and a value of $\delta = 0$ means that the calculated bound is identical to the real bound. We determined the real bounds for all benchmark models with a precision of 10^{-6} using the Verified Global Optimization approach from [32]. If the computation of the bounds took longer than 10 seconds we terminated the calculation. These cases are marked with the symbol "$-$" in Tab. 3 and Tab. 4.

For our Taylor model and affine arithmetic implementation, we compare the results for bounding the polynomial parts with interval arithmetic (Interval subs., see Sec. 2.3.1), using branch and bound (BnB) as well as advanced branch and bound (BnB adv.) with $\epsilon = 0.001$ (see Sec. 2.3.2

and Alg. 1), and using the algorithm that is based on LDB and QFB with $\epsilon = 0.001$ (LDB/QFB, see Sec. 2.3.3).

The results from the comparison of our Taylor model implementation with Flow* in speed and accuracy are summarized in Tab. 3, Tab. 4 and Tab. 5. We also show the results obtained by using a `zoo` object with the range bounding techniques `interval` and `taylm(int)` (see Sec. 4.4). For each benchmark model, we compare the results computed with different maximal polynomial orders. Since Flow* is written in C++, the implementation is significantly faster for all functions compared to CORA. For most benchmark models the precision of Flow* is slightly worse than the precision of Taylor models with interval substitution in CORA. An exception to this are the last two benchmark models, for which the use of Flow* results in large over-approximations. Since these two benchmark models are the only ones that contain fractions, this is most probably caused by a large over-approximation from the Lagrange remainder of the inverse-operation, as it is described in Appendix B. Flow* is not primarily designed to use Taylor models alone, but for computing reachable sets using Taylor models. We are grateful to Xin Chen for providing us with a standalone Taylor model library comprising basic mathematical operations.

For the different bounding techniques implemented in CORA the numerical results are as expected: The more sophisticated algorithms LDB/QFB and branch and bound in general have a higher precision than interval substitution, but in return lead to significantly larger execution times.

In Tab. 6 and Tab. 7 we list the results from our comparison with INTLAB. In addition to the results for affine arithmetic, the results calculated with CORA's interval arithmetic are displayed, too. Since INTLAB considers rounding errors, it is slower than CORA's interval arithmetic implementation. The numerical results show further that the precision of INTLAB's affine arithmetic is better than the one for affine arithmetic in CORA. A possible reason for this is that affine arithmetic in CORA is implemented with Taylor models that have a polynomial order of one, which seems to not fully exploit all advantages of the affine arithmetic approach.

Contrary to the Taylor model implementation, using the branch and bound algorithm for affine arithmetic does not improve the precision. The LDB/QFB algorithm cannot be applied for affine arithmetic, since it is based on a separation of the Taylor model monomials into linear and higher-order monomials. Since affine arithmetic objects are linear by definition, they do not contain any higher-order monomials, which makes the application of the algorithm impossible.

6 Conclusions

This paper has four main contributions. First, we present how we have implemented Taylor models and affine arithmetic in CORA. To our best knowledge, our implementation of Taylor models is the only one available for MATLAB.

Second, we have improved the existing technique for computing divisions with Taylor models by computing a tighter Lagrange remainder.

Third, we compare our implementation of Taylor models with Flow* and our implementation of affine arithmetic with INTLAB. While Flow* is faster due to its implementation in C++, the implementation in CORA is easier to use due to the prototyping capabilities of MATLAB. Our implementation of affine arithmetic in CORA is faster compared to INTLAB, which is also

Table 3: Comparison of Taylor model techniques (part 1). The precision is provided in percent compared to the exact result as shown in (19). The ranges are presented in Tab. 8 and the examples are taken out of [22].

Bench-mark	**Max. order**	**CORA**					**Flow***
		Interval subs.	**BnB**	**BnB (adv.)**	**LDB/QFB**	**Zoo (Int. subs.)**	
sin	2	[139.95, 148.09]	[139.95, 148.09]	[86.67, 148.09]	[82.00, 148.09]	[0, 0]	[215.26, 148.09]
	5	[165.71, 146.30]	[41.51, 8.23]	[6.29, 3.55]	[5.92, 3.54]	[0, 0]	[241.02, 173.98]
	10	[161.56, 145.78]	[35.12, 3.87]	[0.40, 0.03]	[0.01, 0.01]	[0, 0]	[240.96, 173.78]
bspline0	2	[27.40, 0]	[27.40, 0]	[1.33, 0]	[1.00, 0]	[0, 0]	[54.80, 0]
	5	[27.40, 0]	[27.40, 0]	[0, 0]	[0, 0]	[0, 0]	[54.80, 0]
	10	[27.40, 0]	[27.40, 0]	[0, 0]	[0, 0]	[0, 0]	[54.80, 0]
bspline1	2	[0, 30.87]	[0, 30.87]	[0, 3.61]	[0, 2.86]	[0, 0]	[0, 61.74]
	5	[0, 30.87]	[0, 30.87]	[0, 0]	[0, 0]	[0, 0]	[0, 61.74]
	10	[0, 30.87]	[0, 30.87]	[0, 0]	[0, 0]	[0, 0]	[0, 61.74]
bspline2	2	[33.78, 0]	[33.78, 0]	[6.77, 0]	[5.08, 0]	[3.92, 0]	[67.55, 0]
	5	[33.78, 0]	[33.78, 0]	[0, 0]	[0, 0]	[3.92, 0]	[67.55, 0]
	10	[33.78, 0]	[33.78, 0]	[0, 0]	[0, 0]	[3.92, 0]	[67.55, 0]
bspline3	2	[34.85, 0]	[34.85, 0]	[8.40, 0]	[6.41, 0]	[0, 0]	[69.71, 0]
	5	[34.85, 0]	[34.85, 0]	[0, 0]	[0, 0]	[0, 0]	[69.71, 0]
	10	[34.85, 0]	[34.85, 0]	[0, 0]	[0, 0]	[0, 0]	[69.71, 0]
doppler	2	[0.02, 1.56]	[0.02, 1.56]	[0.01, 1.38]	[0.02, 1.56]	[0.02, 0.50]	[0.05, 1.58]
	5	[0.02, 1.55]	[0.02, 1.55]	[0, 1.38]	[0.02, 1.55]	[0.02, 0.50]	[0.05, 1.58]
	10	[0.02, 1.55]	[0.02, 1.55]	[0, 1.38]	[0.02, 1.55]	[0.02, 0.50]	[0.05, 1.58]
himmilbeau	2	[104.80, 90.24]	[104.80, 90.24]	[55.99, 89.54]	-	[104.80, 12.71]	[167.17, 90.94]
	5	[104.80, 90.24]	[29.69, 11.48]	[0.03, 0]	[0, 0]	[104.80, 12.71]	[167.17, 90.94]
	10	[104.80, 90.24]	[29.69, 11.48]	[0.03, 0]	[0, 0]	[104.80, 12.71]	[167.17, 90.94]

Table 4: Comparison of Taylor model techniques (part 1). The precision is provided in percent compared to the exact result as shown in (19). The ranges are presented in Tab. 8 and the examples are taken out of [22].

Bench-mark	**Max. order**	**CORA**					
		Interval subs.	**BnB**	**BnB (adv.)**	**LDB/QFB**	**Zoo (Int. subs.)**	**Flow***
kepler0	2	[8.22, 15.89]	[8.22, 15.89]	[1.23, 1.79]	[0, 0]	[8.22, 15.89]	[8.22, 19.22]
	5	[8.22, 15.89]	[8.22, 15.89]	[1.23, 1.79]	[0, 0]	[8.22, 15.89]	[8.22, 19.22]
	10	[8.22, 15.89]	[8.22, 15.89]	[1.23, 1.79]	[0, 0]	[8.22, 15.89]	[8.22, 19.22]
kepler1	2	[11.82, 37.66]	[11.82, 37.66]	[7.36, 4.70]	-	[11.82, 37.66]	[12.57, 51.39]
	5	[11.82, 37.66]	[2.14, 11.89]	[0, 0.02]	[0, 0]	[11.82, 37.66]	[12.57, 51.39]
	10	[11.82, 37.66]	[2.14, 11.89]	[0, 0.02]	[0, 0]	[11.82, 37.66]	[12.57, 51.39]
kepler2	2	[31.67, 42.14]	[31.67, 42.14]	[4.86, 6.16]	-	[31.67, 42.14]	[31.95, 53.78]
	5	[31.67, 42.14]	[15.74, 28.01]	[0, 0.04]	[0, 0]	[31.67, 42.14]	[31.95, 53.78]
	10	[31.67, 42.14]	[15.74, 28.01]	[0, 0.04]	[0, 0]	[31.67, 42.14]	[31.95, 53.78]
rigidBody1	2	[0, 14.71]	[0, 14.71]	[0, 0.01]	[0, 0]	[0, 10.85]	[0, 14.71]
	5	[0, 14.71]	[0, 14.71]	[0, 0.01]	[0, 0]	[0, 10.85]	[0, 14.71]
	10	[0, 14.71]	[0, 14.71]	[0, 0.01]	[0, 0]	[0, 10.85]	[0, 14.71]
rigidBody2	2	[13.47, 3.57]	[13.47, 3.57]	[1.95, 2.12]	-	[13.47, 3.57]	[21.92, 3.57]
	5	[12.54, 2.64]	[10.50, 1.32]	[0.05, 0]	[0, 0]	[12.54, 2.64]	[20.99, 2.64]
	10	[12.54, 2.64]	[10.50, 1.32]	[0.05, 0]	[0, 0]	[12.54, 2.64]	[20.99, 2.64]
turbine1	2	[135.33, 148.41]	[123.19, 139.93]	[121.69, 61.26]	-	[135.33, 2.67]	[1645.32, 1702.68]
	5	[20.38, 62.14]	[20.38, 62.14]	[18.41, 7.85]	-	[20.38, 2.67]	[6113.43, 14542.23]
	10	[2.29, 49.51]	[2.25, 49.49]	[2.20, 2.98]	-	[2.29, 2.67]	[14398213.48, 14398270.84]
turbine2	2	[124.00, 151.92]	[116.18, 146.75]	[58.67, 127.50]	-	[2.72, 151.92]	[1199.66, 1202.64]
	5	[59.99, 66.64]	[28.05, 32.20]	[5.77, 17.32]	-	[2.72, 66.64]	[3295.72, 4738.72]
	10	[50.50, 53.35]	[22.33, 18.88]	[0.78, 2.14]	-	[2.72, 53.35]	[51542.58, 51545.56]

Table 5: Execution time of Taylor models in [ms]. The precision of the examples is presented in Tab. 3 and Tab. 4.

Bench-mark	**Max. order**	**CORA**					
		Int. subs.	**BnB**	**BnB (adv.)**	**LDB/QFB**	**Zoo**	**Flow***
sin	2	7.610	16.925	40.390	690.022	6.614	0.175
	5	12.776	42.321	50.854	806.454	12.542	0.063
	10	23.159	75.946	90.183	1288.858	28.689	0.236
bspline0	2	4.694	8.256	26.380	830.514	8.620	0.095
	5	5.137	8.491	36.153	56.564	6.092	0.027
	10	4.139	7.746	31.300	47.427	5.239	0.026
bspline1	2	6.405	12.009	31.047	1536.890	7.402	0.145
	5	6.082	11.014	36.325	63.269	7.809	0.054
	10	6.777	12.616	39.637	63.654	6.924	0.053
bspline2	2	7.232	11.932	30.868	1388.466	8.730	0.117
	5	6.959	11.916	38.307	63.569	8.245	0.058
	10	7.188	11.535	37.657	61.369	8.012	0.058
bspline3	2	4.000	7.826	23.898	771.525	4.272	0.075
	5	3.708	7.221	27.936	56.928	3.962	0.031
	10	3.956	7.271	27.891	59.755	4.557	0.031
doppler	2	15.331	48.696	176.618	72.490	14.998	0.613
	5	21.461	90.139	278.200	80.106	22.941	3.424
	10	31.933	127.429	308.640	123.788	33.960	6.895
himmilbeau	2	11.856	26.974	73.296	$> 10^4$	14.998	0.290
	5	11.133	65.974	147.003	209.858	14.375	0.086
	10	11.235	66.423	146.714	209.694	14.222	0.082
kepler0	2	11.692	33.202	83.172	220.816	16.270	0.131
	5	11.282	33.020	84.174	218.009	15.158	0.121
	10	11.432	33.273	83.983	217.224	15.109	0.092
kepler1	2	18.062	48.610	213.107	$> 10^4$	24.472	0.479
	5	17.640	132.902	744.081	301.351	23.831	0.312
	10	17.450	131.120	745.826	298.799	25.074	0.307
kepler2	2	30.870	99.677	2315.325	$> 10^4$	41.566	0.739
	5	30.531	341.963	2967.705	2979.728	38.245	0.435
	10	28.018	350.555	3023.280	2917.997	39.378	0.432
rigidBody1	2	5.321	11.406	56.329	21.239	6.445	0.074
	5	5.775	12.030	56.961	20.943	6.786	0.042
	10	5.180	12.124	65.125	21.046	6.353	0.041
rigidBody2	2	16.050	35.270	207.042	$> 10^4$	16.661	0.262
	5	12.952	51.701	315.271	177.691	14.433	0.136
	10	12.098	43.862	328.342	183.508	14.233	0.135
turbine1	2	64.790	165.714	541.358	$> 10^4$	113.913	0.567
	5	102.399	194.195	1452.165	$> 10^4$	102.124	1.978
	10	162.201	316.186	3773.953	$> 10^4$	174.362	3.593
turbine2	2	31.478	77.325	529.880	$> 10^4$	35.558	0.549
	5	53.520	274.855	2691.977	$> 10^4$	61.416	1.595
	10	94.606	379.205	5378.260	$> 10^4$	94.999	3.005

Table 6: Comparison of affine arithmetic. The precision is provided in percent compared to the exact result as shown in (19). The ranges are presented in Tab. 8 and the examples are taken out of [22].

Bench-mark	**CORA**				
	Interval subs.	**BnB**	**BnB (adv.)**	**Interval**	**IntLab**
sin	[170.89, 106.22]	[170.89, 106.22]	[170.89, 106.22]	[0, 0]	[0, 0]
bspline0	[21.76, 0]	[21.76, 0]	[30.02, 0]	[0, 0]	[0, 0]
bspline1	[0, 23.83]	[0, 23.83]	[-0.00, 33.31]	[0, 0]	[0, 0]
bspline2	[25.12, 0]	[25.12, 0]	[35.73, 0]	[3.92, 3.92]	[3.92, 0]
bspline3	[24.69, 0]	[24.69, 0]	[35.94, 0]	[0, 0]	[0, 0]
doppler	[0.44, 1.95]	[0.44, 1.95]	[0.44, 1.95]	[1.31, 0.50]	[0.45, 0.50]
himmilbeau	[136.64, 92.14]	[136.64, 92.14]	[159.66, 92.40]	[110.10, 12.71]	[110.10, 12.71]
kepler0	[8.22, 15.89]	[8.22, 15.89]	[8.22, 16.72]	[21.12, 32.14]	[8.22, 19.22]
kepler1	[16.82, 42.66]	[16.82, 42.66]	[16.82, 46.10]	[34.96, 77.05]	[20.74, 59.55]
kepler2	[67.20, 77.68]	[67.20, 77.68]	[67.20, 80.33]	[139.40, 148.98]	[69.01, 90.83]
rigidBody1	[0, 14.71]	[0, 14.71]	[0, 14.71]	[0.72, 10.85]	[0, 10.85]
rigidBody2	[18.32, 8.53]	[18.32, 8.53]	[20.46, 8.53]	[15.65, 9.86]	[15.65, 8.53]
turbine1	[206.21, 184.71]	[206.21, 184.71]	[208.26, 191.47]	[240.73, 2.67]	[206.98, 2.67]
turbine2	[148.01, 202.92]	[148.01, 202.92]	[153.10, 205.02]	[2.72, 238.35]	[2.72, 203.87]

Table 7: Execution time of affine arithmetic in [ms]. The precision of the examples is presented in Tab. 6.

Bench-mark	**CORA**				
	Int. subs.	**BnB**	**BnB (adv.)**	**Interval**	**IntLab**
sin	5.275	10.831	15.802	0.410	11.849
bspline0	5.598	12.559	20.037	0.406	9.128
bspline1	7.336	12.767	22.489	0.255	9.375
bspline2	8.404	13.910	23.523	0.317	10.650
bspline3	5.163	9.640	16.495	0.524	4.989
doppler	12.075	31.575	36.919	0.350	11.089
himmilbeau	13.857	29.189	45.662	0.328	14.203
kepler0	14.392	36.029	41.629	0.377	13.657
kepler1	23.585	54.178	66.857	0.436	22.142
kepler2	33.777	102.636	138.833	0.803	36.142
rigidBody1	6.538	12.633	16.050	0.391	6.352
rigidBody2	15.096	34.581	48.820	0.497	17.200
turbine1	250.520	167.818	170.426	1.351	195.326
turbine2	28.528	47.981	60.709	0.882	38.426

Table 8: Input uncertainty for all benchmark models downloadable from https://github.com/malyzajko/daisy/blob/master/testcases/. The placeholders for the uncertainties are defined as follows: $a = [-4.5, -0.3]$, $b = [0.4, 0.9]$, $c = [3.8, 7.8]$, $d = [8, 10]$, $e = [-10, 8]$, $f = [1, 2]$.

Benchmark	**Uncertainty**	**File path**
sin	$sin(x)$, $x \in a$	
bspline0	$u \in a$	/mixed-precision/Bsplines0.scala
bspline1	$u \in a$	/rosa/Bsplines.scala
bspline2	$u \in a$	/rosa/Bsplines.scala
bspline3	$u \in a$	/rosa/Bsplines.scala
doppler	$u \in a$, $v \in b$, $T \in c$	/rosa/Doppler.scala
himmilbeau	$x_1 \in a$, $x_2 \in b$	/real2float/Himmilbeau.scala
kepler0	$x_1 \in a$, $x_2 \in b$, $x_3 \in c$, $x_4 \in d$, $x_5 \in e$, $x_6 \in f$	/real2float/Kepler.scala
kepler1	$x_1 \in a$, $x_2 \in b$, $x_3 \in c$, $x_4 \in d$	/real2float/Kepler.scala
kepler2	$x_1 \in a$, $x_2 \in b$, $x_3 \in c$, $x_4 \in d$, $x_5 \in e$, $x_6 \in f$	/real2float/Kepler.scala
rigidBody1	$x_1 \in a$, $x_2 \in b$, $x_3 \in c$	/control/RigidBody.scala
rigidBody2	$x_1 \in a$, $x_2 \in b$, $x_3 \in c$	/control/RigidBody.scala
turbine1	$v \in a$, $w \in b$, $r \in c$	/rosa/Turbine.scala
turbine2	$v \in a$, $w \in b$, $r \in c$	/rosa/Turbine.scala

implemented using MATLAB. However, our implementation does not yet consider floating-point errors.

Fourth, we provide an evaluation of range bounding techniques. We are not aware of any other paper that has evaluated the combined use of several range bounding techniques. Our finding is that it is typically better to use several simple range bounding methods in parallel and intersect the results, rather than using a single, precise technique.

Acknowledgments

The authors gratefully acknowledge financial support by the European Commission project UnCoVerCPS under grant number 643921 and the German Research Foundation (DFG) under grant AL 1185/5-1. We are also grateful to Xin Chen for providing us with a standalone Taylor model library comprising basic mathematical operations.

References

[1] 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.

[2] G. Alefeld. The centered form and the mean value form – a necessary condition that they yield the range. *Computing*, 33(2):165–169, 1984.

[3] M. Althoff. Reachability analysis of nonlinear systems using conservative polynomialization and non-convex sets. In *Hybrid Systems: Computation and Control*, pages 173–182, 2013.

[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] M. Althoff, S. Bak, D. Cattaruzza, X. Chen, G. Frehse, R. Ray, and S. Schupp. ARCH-COMP17 category report: Continuous and hybrid systems with linear continuous dynamics. In *Proc. of the 4th International Workshop on Applied Verification for Continuous and Hybrid Systems*, pages 143–159, 2017.

[6] M. Althoff and D. Grebenyuk. Implementation of interval arithmetic in CORA 2016. In *Proc. of the 3rd International Workshop on Applied Verification for Continuous and Hybrid Systems*, pages 91–105, 2016.

[7] V. Balakrishnan, S. Boyd, and S. Balemi. Branch and bound algorithm for computing the minimum stability degree of parameter-dependent linear systems. *International Journal of Robust and Nonlinear Control*, 1(4):295–317, 1991.

[8] M. Berz and G. Hoffstätter. Computation and application of Taylor polynomials with interval remainder bounds. *Reliable Computing*, 4:83–97, 1998.

[9] Martin Berz. From Taylor Series to Taylor Models. 405, 1997.

[10] Apostolos Burnetas, Daniel Solow, and Rishi Agarwal. An analysis and implementation of an efficient in-place bucket sort. *Acta Informatica*, 34(9):687–700, 1997.

[11] 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.

[12] X. Chen, M. Althoff, and F. Immler. ARCH-COMP17 category report: Continuous systems with nonlinear dynamics. In *Proc. of the 4th International Workshop on Applied Verification for Continuous and Hybrid Systems*, pages 160–169, 2017.

[13] X. Chen, S. Sankaranarayanan, and E. Ábrahám. Taylor model flowpipe construction for non-linear hybrid systems. In *Proc. of the 33rd IEEE Real-Time Systems Symposium*, 2012.

[14] David Cox, John Little, and Donal OShea. Groebner bases. In *Ideals, varieties, algorithms: An introduction to computational algebraic geometry and commutative algebra, UTM*, chapter 2. Springer-Verlag, New York, 2007.

[15] Luiz Henrique de Figueiredo and Jorge Stolfi. Affine arithmetic: Concepts and applications. *Numerical Algorithms*, 37(1-4):147–158, 2004.

[16] J.-P. Eckmann, A. Malaspinas, and S. Oliffson Kamphorst. *A Software Tool for Analysis in Function Spaces*, pages 147–167. Springer, 1991.

[17] C. Fan, J. Kapinski, X. Jin, and S. Mitra. Locally optimal reach set over-approximation for nonlinear systems. In *Proc. of the International Conference on Embedded Software*, pages 1–10, 2016.

[18] E. Hansen, W. Walster, and G. W. Walster. *Global Optimization Using Interval Analysis*. CRC Press, 2003.

[19] E. R. Hansen. A generalized interval arithmetic. In Karl Nickel, editor, *Interval Mathematics*, pages 7–18. Springer, 1975.

[20] F. Immler. Formally verified computation of enclosures of solutions of ordinary differential equations. In *Proc. of the 6th NASA Formal Methods Symposium*, pages 113–127, 2014.

[21] 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.

[22] Anastasiia Izycheva and Eva Darulova. On sound relative error bounds for floating-point arithmetic. In *Proc. of the 17th Conference on Formal Methods in Computer-Aided Design*, pages 15–22, 2017.

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

[24] 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.

[25] R. Krawczyk and A. Neumaier. Interval slopes for rational functions and associated centered forms. *SIAM Journal on Numerical Analysis*, 22(3):604–616, 1985.

[26] W. Kühn. *Mathematical Visualization*, chapter Zonotope Dynamics in Numerical Quality Control, pages 125–134. Springer, 1998.

[27] E. L. Lawler and D. E. Wood. Branch-and-bound methods: A survey. *Operations Research*, 14(4):699–719, 1966.

[28] K. Makino and M. Berz. Taylor models and other validated functional inclusion methods. *International Journal of Pure and Applied Mathematics*, 4(4):379–456, 2003.

[29] K. Makino and M. Berz. COSY INFINITY version 9. *Nuclear Instruments and Methods in Physics Research A*, 558(1):346–350, 2006.

[30] K. Makino and M. Berz. Rigorous integration of flows and ODEs using Taylor models. In *Proc. of Symbolic-Numeric Computation*, pages 79–84, 2009.

[31] Kyoko Makino and Martin Berz. Remainder Differential Algebras and Their Applications. 1996.

[32] Kyoko Makino and Martin Berz. Verified global optimization with taylor model-based range bounders. *Transactions on Computers*, 11(1):1611–1618, 2005.

[33] Stefan Mitsch and André Platzer. ModelPlex: Verified runtime validation of verified cyber-physical system models. *Formal Methods in System Design*, 49(1):33–74, 2016.

[34] N. S. Nedialkov. VNODE-LP—a validated solver for initial value problems in ordinary differential equations. Technical Report CAS-06-06-NN, McMaster University, 2006.

[35] N. S. Nedialkov, V. Kreinovich, and S. A. Starks. Interval arithmetic, affine arithmetic, Taylor series methods: Why, what next? *Numerical Algorithms*, 37:325–336, 2004.

[36] M. Neher, K. R. Jackson, and N. S. Nedialkov. On Taylor model based integration of ODEs. *SIAM Journal on Numerical Analysis*, 45(1):236–262, 2007.

[37] A. Neumaier. Taylor forms—use and limits. *Reliable Computing*, 9:43–79, 2003.

[38] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 2007.

[39] N. Ramdani and N. S. Nedialkov. Computing reachable sets for uncertain nonlinear hybrid systems using interval constraint-propagation techniques. *Nonlinear Analysis: Hybrid Systems*, 5(2):149–162, 2010.

[40] S. Ratschan and Z. She. Safety verification of hybrid systems by constraint propagation based abstraction refinement. *ACM Transactions in Embedded Computing Systems*, 6(1):1–23, 2007.

[41] A. Rauh, E. Auer, and E. P. Hofer. Valencia-ivp: A comparison with other initial value problem solvers. In *CD-Proc. of the 12th GAMM-IMACS International Symposium on Scientific Computing, Computer Arithmetic, and Validated Numerics*. IEEE Computer Society, 2007.

[42] S. M. Rump. *Developments in Reliable Computing*, chapter INTLAB - INTerval LABoratory, pages 77–104. Kluwer Academic Publishers, 1999.

[43] 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.

[44] J. Stolfi and L. H. de Figueiredo. Self-validated numerical methods and applications. Monograph for 21st Brazilian Mathematics Colloquium, IMPA, 1997.

[45] R. Trinchero, P. Manfredi, T. Ding, and I. S. Stievano. Combined parametric and worst case circuit analysis via taylor models. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 63(7):1067–1078, 2016.

A List of Smooth Functions

Let $T = P(x - x_0) + [I]$ be a Taylor model, $\forall x \in \mathcal{D}$, where $P(x - x_0)$ is a polynomial part up to the q-th order, I is an interval, $c_f = P(0)$, $\tilde{P}(x - x_0) = P(x - x_0) - c_f$, $\tilde{T} = \tilde{P}(x - x_0) + [I]$, and $B(\cdot)$ a boundary estimator for Taylor models.

Formulas for monotonic functions:

- Exponential from [28, Eq (2.2)–(2.4)]:

$$\begin{aligned} \exp(T) = \exp(c_f) \cdot \Big[1 + \tilde{T} + \frac{1}{2!}\tilde{T}^2 + ... + \frac{1}{q!}\tilde{T}^q\Big] \\ + \frac{\exp(c_f)}{(q+1)!}\Big(B(\tilde{T}^{q+1}) \cdot \exp([0,1] \cdot B(\tilde{T})\Big). \end{aligned} \tag{20}$$

- Logarithm from [28, p. 386] is defined under the condition$B(T) \subset (0, \infty)$:

$$\begin{aligned} \log(T) = \log c_f + \frac{\tilde{T}}{c_f} - \frac{1}{2}\frac{\tilde{T}^2}{c_f^2} + ... + (-1)^{q+1}\frac{1}{q}\frac{\tilde{T}^q}{c_f^q} \\ + (-1)^{q+2}\frac{1}{q+1}\frac{B(\tilde{T}^{q+1})}{c_f^{q+1}}\frac{1}{(1 + [0,1] \cdot \frac{B(\tilde{T})}{c_f})^{q+1}}. \end{aligned} \tag{21}$$

- Square root from [28, p. 386] is defined under the condition$B(T) \subset (0, \infty)$:

$$\begin{aligned} \sqrt{T} = \sqrt{c_f}\Big[1 + \frac{1}{2}\frac{\tilde{T}}{c_f} - \frac{1}{2!2^2}\frac{\tilde{T}^2}{c_f^2} + ... + (-1)^{q-1}\frac{(2q-3)!!}{q!2^q}\frac{\tilde{T}^q}{c_f^q}\Big] \\ + (-1)^q\sqrt{c_f}\frac{(2q-1)!!}{(q+1)!2^{q+1}}\frac{B(\tilde{T}^{q+1})}{c_f^{q+1}}\frac{1}{(1 + [0,1] \cdot \frac{B(\tilde{T})}{c_f})^{q+1/2}}. \end{aligned} \tag{22}$$

- Inverse $\frac{1}{T}$: see Sec. 2.1

Formulas for trigonometric functions:

- Sine from [28, p. 387]:

$$\begin{aligned} \sin(T) = \sin(c_f) + \cos(c_f) \cdot \tilde{T} - \frac{1}{2!}\sin(c_f) \cdot \tilde{T}^2 - \frac{1}{3!}\cos(c_f) \cdot \tilde{T}^3 + ... \\ + \frac{1}{(q+1)!}B(\tilde{T}^{q+1}) \cdot J \end{aligned} \tag{23}$$

  where

$$J = \begin{cases} -J_0 & \text{if} \quad \bmod (q,4) = 1 \text{ or } 2, \\ J_0 & \text{else,} \end{cases}$$

$$J_0 = \begin{cases} \cos(c_f + [0,1] \cdot B(\tilde{T})) & \text{if } q \text{ is even,} \\ \sin(c_f + [0,1] \cdot B(\tilde{T})) & \text{else.} \end{cases}$$

- Cosine from [28, p. 387]:

$$\begin{aligned}\cos(T) = \cos(c_f) - \sin(c_f)\cdot\tilde{T} - \frac{1}{2!}\cos(c_f)\cdot\tilde{T}^2 + \frac{1}{3!}\sin(c_f)\cdot\tilde{T}^3 + ...\\ + \frac{1}{(q+1)!}B(\tilde{T}^{q+1})\cdot J\end{aligned} \tag{24}$$

  where

$$J = \begin{cases} -J_0 & \text{if} \quad \bmod (q,4) = 0 \text{ or } 1, \\ \ \ J_0 & \text{else}, \end{cases}$$

$$J_0 = \begin{cases} \sin(c_f + [0,1]\cdot B(\tilde{T})) & \text{if } q \text{ is even}, \\ \cos(c_f + [0,1]\cdot B(\tilde{T})) & \text{else}. \end{cases}$$

- Tangent:

$$\tan(T) = \tan(c_f + \tilde{T}) = (\tan(c_f) + \tan(\tilde{T}))\cdot\Big(\frac{1}{1-\tan(c_f)\cdot\tan(\tilde{T})}\Big), \tag{25}$$

  which is computationally equivalent to

$$\tan(T) = \sin(T)\cdot\Big(\frac{1}{\cos(T)}\Big). \tag{26}$$

  Sine it is easier to implement (26) while the formula (25) has no computational benefits, therefore the formula (26) is used in CORA.

Formulas for hyperbolic functions:

- Hyperbolic sine from [28, p. 388]:

$$\begin{aligned}\sinh(T) = \sinh(c_f) + \cosh(c_f)\cdot\tilde{T} + \frac{1}{2!}\sinh(c_f)\cdot\tilde{T}^2 + \frac{1}{3!}\cosh(c_f)\cdot\tilde{T}^3 + ...\\ + \frac{1}{(q+1)!}B(\tilde{T}^{q+1})\cdot J_0\ ,\end{aligned} \tag{27}$$

  where

$$J_0 = \begin{cases} \cosh(c_f + [0,1]\cdot B(\tilde{T})) & \text{if } q \text{ is even}, \\ \sinh(c_f + [0,1]\cdot B(\tilde{T})) & \text{else}. \end{cases}$$

- Hyperbolic cosine from [28, p. 388]:

$$\begin{aligned}\cosh(T) = \cosh(c_f) + \sinh(c_f)\cdot\tilde{T} + \frac{1}{2!}\cosh(c_f)\cdot\tilde{T}^2 + \frac{1}{3!}\sinh(c_f)\cdot\tilde{T}^3 + ...\\ + \frac{1}{(q+1)!}B(\tilde{T}^{q+1})\cdot J_0\ ,\end{aligned} \tag{28}$$

  where

$$J_0 = \begin{cases} \sinh(c_f + [0,1]\cdot B(\tilde{T})) & \text{if } q \text{ is even}, \\ \cosh(c_f + [0,1]\cdot B(\tilde{T})) & \text{else}. \end{cases}$$

- Hyperbolic tangent:
$$\tanh(T) = \frac{\sinh(T)}{\cosh(T)}. \tag{29}$$

Formulas for inverse trigonometric functions:

- Arcsine from [28, p. 388] under the condition of$B(T) \subset (-1, 1)$:
$$\arcsin(T) = \arcsin(c_f) + \arcsin(T \cdot \sqrt{1 - c_f^2} - c_f \cdot \sqrt{1 - T^2}) \tag{30}$$
Utilizing that
$$G \equiv T \cdot \sqrt{1 - c_f^2} - c_f \cdot \sqrt{1 - T^2}$$
we have
$$\arcsin(G) = G + \frac{1}{3!}G^3 + \frac{3^2}{5!}G^5 + \frac{3^2 5^2}{7!}G^7 + ... + \frac{1}{(q+1)!}B(G)^{q+1} \arcsin^{(q+1)}(\theta \cdot B(G))$$
where $\arcsin^{(1)}(0) = 1$, $\arcsin^{(2)}(0) = 0$, $\arcsin^{(3)}(0) = 1$, and $\arcsin^{(q)}(0) = (q - 2)^2 \arcsin^{(q-2)}(0)$.
- Arccosine from [28, p. 389]:
$$\arccos(T) = \frac{\pi}{2} - \arcsin(T). \tag{31}$$
- Arctangent from [28, p. 389]:
$$\arctan(T) = \arctan(c_f) + \arctan\left(\frac{T - c_f}{1 + c_f \cdot T}\right) \tag{32}$$
Utilizing that
$$G \equiv \frac{T - c_f}{1 + c_f \cdot T}$$
we have
$$\begin{aligned} \arctan(G) = G - \frac{1}{3}G^3 + \frac{1}{5}G^5 - \frac{1}{7}G^7 + ... \\ + \frac{1}{(q+1)}B(G)^{q+1} \cos^{q+1}\Big(\arctan(\theta \cdot B(G))\Big) \\ \cdot \sin\Big((q+1)(\arctan(\theta \cdot B(G)) + \frac{\pi}{2})\Big). \end{aligned} \tag{33}$$

B Tight Bounds for the Taylor Model Inverse Operation

As shown in Section 2.1, the equation for the $\frac{1}{T}$ operation for a Taylor model $T = P(x - x_0) + [I]$ can be derived from the Taylor series of the function $f(x) = \frac{1}{x}$. The Lagrange remainder term

$$L_q(B(\tilde{T}), c_f) = (-1)^{q+1} \frac{B(\tilde{T})^{q+1}}{c_f^{q+2}} \frac{1}{\left(1 + [0, 1] \cdot \frac{B(\tilde{T})}{c_f}\right)^{q+2}} \ , \ c_f = P(0), \ \tilde{T} = T - c_f \tag{34}$$

thereby represents an over-approximation of the exact integral remainder

$$R_q(x, c_f) = \int_{c_f}^{x} f^{(q+1)}(t) \frac{(x-t)^q}{q!} dt, \quad x \in B(T), \tag{35}$$

where $f^{(i)}$ represents the i-th derivative of the function $f(\cdot)$. Evaluation of the Lagrange remainder (34) for a Taylor model can lead to very large over-approximations, as the following example demonstrates:

$$\begin{aligned}
&T = 0.45 + 0.35x + [0, 0],\ x \in [-1, 1],\\
&B(T) = [0.1, 0.8],\ c_f = 0.45,\ B(\tilde{T}) = [-0.35, 0.35],\\
&\frac{1}{B(T)} = [1.25, 10],\\
&L_{q=6}([-0.35, 0.35], 0.45) = [-64339.29688, 64339.29688].
\end{aligned} \tag{36}$$

One solution to this problem is to analytically solve the integral in (35) for the function $f(x) = \frac{1}{x}$, which results in an equation for the exact remainder:

$$\begin{aligned}
R_q(x, c_f) &= \underbrace{(q+1) \cdot (-1)^{(q+1)}}_{:=a} \int_{c_f}^{x} \frac{(x-t)^q}{t^{(q+2)}} dt,\\
&= -\frac{a}{q+1} \frac{(x-t)^q}{t^{(q+1)}} \bigg|_{c_f}^{x} - \frac{a \cdot q}{q+1} \int_{c_f}^{x} \frac{(x-t)^{(q-1)}}{t^{(q+1)}} dt,\\
&= \frac{a}{q+1} \frac{(x-c_f)^q}{c_f^{(q+1)}} - \frac{q}{q+1} \cdot R_{q-1}(x, c_f)\ ,
\end{aligned} \tag{37}$$

with

$$R_0(x, c_f) = a \cdot \int_{c_f}^{x} \frac{1}{t^2} dt = -\frac{a}{t}\bigg|_{c_f}^{x} = \frac{a}{c_f} - \frac{a}{x}. \tag{38}$$

The exact remainder therefore represents a polynomial, and its bounds on the range of the Taylor model $R_q\left(B(T), c_f\right)$ can be calculated with standard interval arithmetic. Due to the limitations of interval arithmetic, this can also lead to possibly large over-approximations, which are however usually smaller than the over-approximations resulting from evaluation of the Lagrange remainder (34). Tighter bounds for the exact remainder can be obtained if the input interval $B(T)$ is split into several sub-intervals. The over-approximation error can be made arbitrarily small if the sub-intervals are chosen small enough. Since the problem is one dimensional, this splitting-based technique does not suffer from the curse of dimensionality.

Because the evaluation of the Lagrange remainder (34) is faster than the evaluation of the exact remainder (37), and because the Lagrange remainder does not always result in large over-approximations, we implemented the following heuristic in CORA for the calculation of the remainder rem:

$$rem = \begin{cases} L_q(\tilde{r}, c_f) & L_q(\tilde{r}, c_f) \subset b_{int} \\ R_q(r, c_f) & \text{else} \end{cases}, \tag{39}$$

where $c_f = P(0)$, $\tilde{T} = T - c_f$, $r = B(\tilde{T})$, $r = B(T)$ and $b_{int} = \frac{1}{r}$.

C Examples

Several examples of Mathematical functions evaluated in the MATLAB:

```
a1 = interval(-1, 2); % generate a scalar interval [-1,2]
a2 = interval(2, 3);  % generate a scalar interval [2,3]
a3 = interval(-6, -4);  % generate a scalar interval [-6,4]
a4 = interval(4, 6);  % generate a scalar interval [4,6]

b1 = taylm(a1, 6);  % generate a scalar Taylor model with the maximal order of 6 and names a1
b2 = taylm(a2, 6);  % generate a scalar Taylor model with the maximal order of 6 and names a2
b3 = taylm(a3, 6);  % generate a scalar Taylor model with the maximal order of 6 and names a3
b4 = taylm(a4, 6);  % generate a scalar Taylor model with the maximal order of 6 and names a4

B1 = [b1; b2] % generate a row of Taylor models
B2 = [b3; b4] % generate a row of Taylor models

B1 + B2 % addition
B1' * B2 % matrix multiplication
B1 .* B2 % pointwise multiplication
B1 / 2 % division by scalar
B1 ./ B2 % pointwise division
B1.^3 % power function
sin(B1) % sine function
sin(B1(1,1)) + B1(2,1).^2 - B1' * B2 % scalar + matrix combination of functions
```

This produces the workspace output:

```
B1 =
          0.5 + 1.5*a1 + [0.00000,0.00000]
          2.5 + 0.5*a2 + [0.00000,0.00000]


B2 =
          -5.0 + a3 + [0.00000,0.00000]
          5.0 + a4 + [0.00000,0.00000]


B1 + B2 =
          -4.5 + 1.5*a1 + a3 + [0.00000,0.00000]
          7.5 + 0.5*a2 + a4 + [0.00000,0.00000]


B1' * B2 =
          10.0 - 7.5*a1 + 2.5*a2 + 0.5*a3 + 2.5*a4 + 1.5*a1*a3 + 0.5*a2*a4 + [0.00000,0.00000]


B1 .* B2 =
          -2.5 - 7.5*a1 + 0.5*a3 + 1.5*a1*a3 + [0.00000,0.00000]
          12.5 + 2.5*a2 + 2.5*a4 + 0.5*a2*a4 + [0.00000,0.00000]


B1 / 2 =
          0.25 + 0.75*a1 + [0.00000,0.00000]
          1.25 + 0.25*a2 + [0.00000,0.00000]


B1 ./ B2 =
```

```
          -0.1 - 0.3*a1 - 0.02*a3 - 0.06*a1*a3 - 0.004*a3^2 - 0.012*a1*a3^2
          - 0.0008*a3^3 - 0.0024*a1*a3^3 - 0.00016*a3^4 - 0.00048*a1*a3^4
          - 0.000032*a3^5 - 0.000096*a1*a3^5 - 6.4e-6*a3^6 + [-0.00005,0.00005]
          0.5 + 0.1*a2 - 0.1*a4 - 0.02*a2*a4 + 0.02*a4^2 + 0.004*a2*a4^2
          - 0.004*a4^3 - 0.0008*a2*a4^3 + 0.0008*a4^4 + 0.00016*a2*a4^4
          - 0.00016*a4^5 - 0.000032*a2*a4^5 + 0.000032*a4^6 + [-0.00005,0.00005]

B1.^3 =
          0.125 + 1.125*a1 + 3.375*a1^2 + 3.375*a1^3 + [0.00000,0.00000]
          15.625 + 9.375*a2 + 1.875*a2^2 + 0.125*a2^3 + [0.00000,0.00000]

sin(B1) =
          0.47943 + 1.3164*a1 - 0.53935*a1^2 - 0.49364*a1^3 + 0.10113*a1^4
          + 0.055535*a1^5 - 0.0075847*a1^6 + [-0.00339,0.00339]
          0.59847 - 0.40057*a2 - 0.074809*a2^2 + 0.01669*a2^3 + 0.0015585*a2^4
          - 0.00020863*a2^5 - 0.000012988*a2^6 + [-0.00000,0.00000]

sin(B1(1,1)) + B1(2,1).^2 - B1' * B2 =
          -3.2706 + 8.8164*a1 - 0.5*a3 - 2.5*a4 - 0.53935*a1^2 + 0.25*a2^2
          - 1.5*a1*a3 - 0.5*a2*a4 - 0.49364*a1^3 + 0.10113*a1^4
          + 0.055535*a1^5 - 0.0075847*a1^6 + [-0.00339,0.00339]
```