EPiC Series in Computing

Volume 54, 2018, Pages 145–173

ARCH18. 5th International Workshop on Applied Verification of Continuous and Hybrid Systems

Implementation of Taylor Models in CORA 2018

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

Tool Presentation: 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 our MATLAB tool CORA 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 [45], the error of an approximate solution of ordinary differential equations is obtained from function bounds. In [12, 14, 35, 43], 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 [2, 23, 24] to abstract arbitrary differential equations to polynomial differential equations for reachability analysis. The Jacobian of linearized systems is bounded in [20] to compute reachable sets. Other approaches for reachability analysis use interval constraint propagation [27, 44]. A combination of Taylor models with SAT solving is presented in [19]. Function values are also bounded for verified runtime validation [36, Sec. 3.2]. Further tools requiring bounds of functions can be found in [4, 13, 47].

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 [26], generalized interval arithmetic [22], affine arithmetic [17], and Taylor models [9, 18]. Since generalized interval arithmetic and affine arithmetic are conceptually identical [17, 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 rather many cases in which interval arithmetic provides tighter results [41]. 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 [3, 5]. 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 [41, 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 [25] to remove bias from choosing the examples considered. We also compare our results with the established tools Flow* [12] for Taylor models and INTLAB [46] for interval arithmetic. Please note that Flow* has been primarily designed for reachability analysis and not for bounding function values. Another work that has compared interval arithmetic, affine arithmetic, and Taylor models is [38], 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 [32], but this work has evaluated the performance on rather small input uncertainties, while 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* [9, 31, 32, 35]. An earlier development of Taylor models, which was not introduced under that name can be found in [18]. Taylor models are implemented in the tools Flow* [11, 12], COSY INFINITY [34], and Ariadne [7, 8]; the implementation of Taylor models in Ariadne is additionally validated in Coq [15]. Other tools also use concepts from Taylor models, such as VNODE-LP [37].

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 (see Sec. 3 in [40])). *Let us first introduce the multi-index set*

$$\mathcal{L}^q = \left\{ (l_1, l_2, \ldots, l_n) \middle| l_i \in \mathbb{N}, \sum_{i=1}^{n} l_i \leq q \right\}.$$

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

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

Definition 2 (Taylor model (see Def. 1 in [32])). *Let $f : \mathbb{R}^n \to \mathbb{R}^m$ be a function that is $(q+1)$ times continuously differentiable in an open set 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 on 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]. An illustration of a fourth-order Taylor model is shown in Fig. 1 for $r = \cos(x)$ and the range $[x] = [-\pi/3, \pi/2]$.

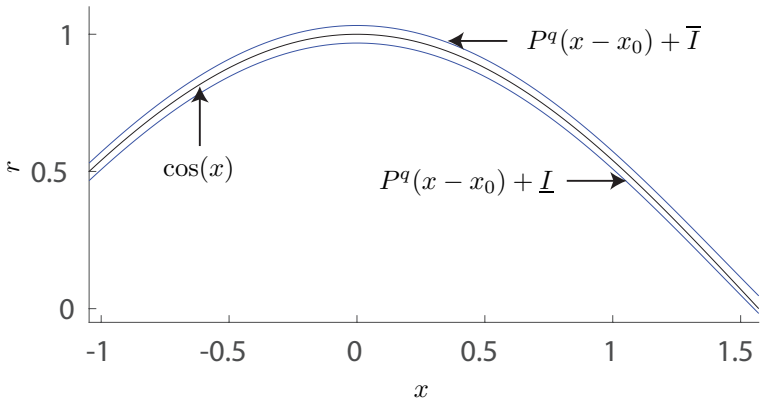


Figure 1: Fourth-order Taylor model for $\cos(x)$ and $[x] = [-\pi/3, \pi/2]$.

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 [5] for information on its implementation in CORA. Binary operations typically required for scalar intervals are addition, subtraction, multiplication, and division:

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

Similarly, bounds on standard functions, such as $\sin([x])$ can be obtained [5]. 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 [26, Sec. 2.2.3]. 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 [41, Sec. 12]. Let us first introduce the polynomials P_1 and P_2 (both of order q), the remaining part P_r of the polynomial $P_1\cdot P_2$ including terms of orders 0 to q, and the cut-off part P_c of the polynomial $P_1\cdot P_2$ including terms of orders $q+1$ to $2q$. We also require the operator

$$B(P^q(x-x_0)) = [\min_{x\in[x]} P^q(x-x_0), \max_{x\in[x]} P^q(x-x_0)] \tag{5}$$

returning an over-approximation of the exact bounds. 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 [32, Sec. 2]:

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

$$\alpha T_1 - \beta T_2 = (\alpha P_1 - \beta P_2,\ \alpha[I_1] - \beta[I_2]), \tag{7}$$

$$T_1\cdot T_2 = (P_r,\ B(P_c) + B(P_1)\cdot I_2 + B(P_2)\cdot I_1 + I_1\cdot I_2), \tag{8}$$

$$\frac{T_1}{T_2} = T_1\cdot\left(\frac{1}{T_2}\right), \tag{9}$$

$$\frac{1}{T} = \frac{1}{c_f}\left[1 - \frac{\tilde{T}}{c_f} + \frac{\tilde{T}^2}{c_f^2}\ldots + (-1)^q\frac{\tilde{T}^q}{c_f^q}\right] + (-1)^{q+1}\frac{B(\tilde{T})^{q+1}}{c_f^{q+2}}\frac{1}{\left(1+[0,1]\cdot\frac{B(\tilde{T})}{c_f}\right)^{q+2}}. \tag{10}$$

Please note that $1/T$ as well as $[1]/[y]$ is only defined if the division by zero is excluded. We have improved the computation of $1/T$ as presented in Appendix B. It is now obvious that the previous example $[x]-[x]=0$ 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 compute $B(\cdot)$, which is addressed subsequently.

2.2 Conversion of an Interval to a Taylor Model

Converting an interval to a Taylor model is trivial [39, Sec. 2.4]. A Taylor model T of an interval $[a]$ is

$$T = P(x) + I = 0.5(\overline{a} + \underline{a}) + 0.5(\overline{a} - \underline{a})\, x + [0, 0], \text{ where } x \in [-1, 1]. \tag{11}$$

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 [1,28]. 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. The bounds of a Taylor model T with a polynomial P and interval $[I]$ can be over-approximated as

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

using (5). 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:

$$B\big(P(x,y)\big) = \left\{ P(x,y) = \sum_{i=0}^{n} \sum_{j=0}^{n} c_{i,j} x^i y^j \,\middle|\, x \in [\underline{x}, \overline{x}], y \in [\underline{y}, \overline{y}] \right\} \subseteq \sum_{i=0}^{n} \sum_{j=0}^{n} c_{i,j} [\underline{x}, \overline{x}]^i [\underline{y}, \overline{y}]^j. \tag{12}$$

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.

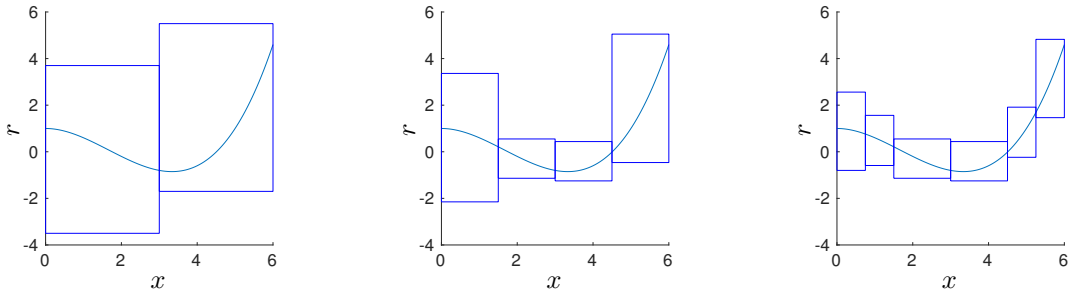
2.3.2 Branch and Bound

For most polynomials, interval arithmetic provides too conservative results. Finding guaranteed bounds is the goal of global optimization techniques [21]. 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 [6, 30] 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. We refine the computation by splitting 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 each split in their respective coordinate with the largest radius (line 18-29). If the lower and upper bounds of the overall range $[r_\cup] \leftarrow \bigcup_i [r^{(i)}]$ (line 30) do not change more than $2\epsilon \cdot \text{rad}([r_\cup])$ (line 31), where $\text{rad}([x]) := 0.5(\overline{x} - \underline{x})$, the final range $[r_\cup]$ is returned. If we set $\epsilon \to \infty$, the branch and bound technique becomes identical to 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 computationally 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 bounds 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*.

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 (12)
  $[r_\cup] \leftarrow [r^{(k)}]$ ▷ 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 (12)
    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 (12)
      $\beta \leftarrow \text{argmax}_i(\text{rad}([x]_i^{(\underline{\alpha})}))$ ▷ return coordinate with the largest radius
      $k = k + 1$
      $[x^{(\underline{\alpha},fh)}] \leftarrow [x^{(\underline{\alpha})}]$, $[x_\beta^{(\underline{\alpha},fh)}] \leftarrow [\underline{x}_\beta^{(\underline{\alpha})}, \overline{x}_\beta^{(\underline{\alpha})} - \texttt{rad}([x_\beta^{(\underline{\alpha})}])]$ ▷ 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 (12)
    $[r_\cup] \leftarrow \bigcup_i [r^{(i)}]$ ▷ update union of all intervals
  until $\overline{r}_{\cup,prev} - \overline{r}_\cup <= 2\epsilon \cdot \text{rad}([r_\cup])$ & $\underline{r}_{\cup,prev} - \underline{r}_\cup <= 2\epsilon \cdot \text{rad}([r_\cup])$
  return $[r_\cup]$

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 combining the Linear Dominated Bounder (LDB) and the Quadratic Fast Bounder (QFB) presented in [33]. Similar to branch and bound, this algorithm is based on the iterative splitting of the input interval into smaller sub-intervals 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. Our algorithm 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. 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.

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 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 [49, Sec. V.B], which is based on the Sherman-Morrison (SM) formula (see [42, Sec. 2.7.1])

$$(A+B)^{-1} = A^{-1} - \frac{1}{1+\mathrm{tr}(A^{-1}B)}(A^{-1}BA^{-1}), \tag{13}$$

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 $T = (P^q(x - x_0), I) = c_f + (\tilde{P}^q(x - x_0), I) = c_f + \tilde{T}$, where $\tilde{P}^q(x - x_0) = P^q(x - x_0) - c_f$ and $\tilde{T} = T - c_f$. When $\tilde{T}$ has unitary rank, we can compute

$$(c_f + \tilde{T})^{-1} = c_f^{-1} - \frac{1}{1+\mathrm{tr}(c_f^{-1}\tilde{T})}(c_f^{-1}\tilde{T}c_f^{-1}) \tag{14}$$

using only methods from Sec. 2.1. However, in most cases, $\tilde{T}$ has full rank so that [49, Sec. V.B] splits the matrix $\tilde{T}$ into a 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 (14) for $\tilde{T}_1$ and c_f. Next, $(c_f + \tilde{T}_1 + \tilde{T}_2)^{-1}$ is calculated by choosing $A = c_f + \tilde{T}_1$ and $B = \tilde{T}_2$ in (13); 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., [17]):

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

The possible values of $\hat{x}$ lie within a zonotope [29]. 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 [2].

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 [48, 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 from 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 require naming 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 (11), 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., $x_1^2 x_2^3 x_3$. Let us define d as the number of monomial terms and n as the number of variables. 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. Let us consider the following polynomial $p(x)$ that consists of $d = 4$ monomial terms and has $n = 3$ variables as a demonstrating example:

$$p(x) = 1.3 + 3.4x_1^2x_2 - 2.3x_1^3x_4^2 + 4.5x_1^3x_2^2x_4^3. \tag{15}$$

This polynomial is represented in CORA as:

$$P(x) = \{c, E, id\}, \text{ with } c = \begin{bmatrix} 1.3 \\ 3.4 \\ -2.3 \\ 4.5 \end{bmatrix}, E = \begin{bmatrix} 0 & 2 & 3 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 2 & 3 \end{bmatrix}, id = \begin{bmatrix} "x_1" \\ "x_2" \\ "x_4" \end{bmatrix}. \tag{16}$$

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,

[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

which can be easily realized by commutation of matrix rows and insertion of new all-zero rows. We demonstrate this concept using two polynomials $P_1(x)$ and $P_2(x)$:

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

The transformation of these two polynomials to a common representation yields

$$\begin{aligned} P_1(x) &= \{c_1, E_1, id_1\}, \quad \text{with } c_1 = \begin{bmatrix} 2.3 \\ 3.1 \\ -2.7 \end{bmatrix}, \; E_1 = \begin{bmatrix} 0 & 1 & 4 \\ 0 & 0 & 0 \\ 1 & 2 & 2 \end{bmatrix}, \; id_1 = \begin{bmatrix} "x_2" \\ "x_3" \\ "x_5" \end{bmatrix}, \\ P_2(x) &= \{c_2, E_2, id_2\}, \quad \text{with } c_2 = \begin{bmatrix} -1.1 \\ 5.2 \end{bmatrix}, \; E_2 = \begin{bmatrix} 0 & 0 \\ 0 & 3 \\ 0 & 0 \end{bmatrix}, \; id_2 = \begin{bmatrix} "x_2" \\ "x_3" \\ "x_5" \end{bmatrix}. \end{aligned} \tag{18}$$

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 [16, Chap. 2]. 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 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 E_{new} according to a scalar hash value $h(e_{new}^{(i)})$, where $e_{new}^{(i)}$ is the i-th matrix column of E_{new} 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 $h(e_{new}^{(i)}) = [1, 1, \ldots, 1]e_{new}^{(i)}$ for our Taylor model implementation, which is identical to the sum of entries in the column $e_{new}^{(i)}$.

The multiplication of two polynomials can be implemented similar to 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 described above.

4.2 List of 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(·) function as defined in (31)
`asin`	arcsin(·) function as defined in (30)
`atan`	arctan(·) function as defined in (32)
`cos`	cos(·) function as defined in (25)
`cosh`	cosh(·)function as defined in (28)
`det`	determinant of a Taylor model matrix
`exp`	exponential function as defined in (21)
`interval`	various implementations of the bound operator $B(\cdot)$ as presented in Sec. 2.3
`log`	natural logarithm function as defined in (22)
`minus`	overloaded '-' operator, see (7)
`mpower`	overloaded '^' operator (power)
`mrdivide`	overloaded '/' operator (division), see (9)
`mtimes`	overloaded '*' operator (multiplication), see (8) for scalars and Sec. 2.4 for matrices
`plus`	overloaded '+' operator (addition), see (6) for scalars and Sec. 2.4 for matrices
`power`	overloaded '.^' 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(·) function as defined in (24)
`sinh`	sinh(·) function as defined in (27)
`sqrt`	$\sqrt{(\cdot)}$ function as defined in (23)
`tan`	tan(·) function as defined in (26)
`tanh`	tanh(·) function as defined in (29)
`times`	overloaded '.*' operator for elementwise multiplication of matrices
`trace`	trace of a Taylor model matrix
`uminus`	overloaded '-' operator for a single operand
`uplus`	overloaded '+' operator for a single operand

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.

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 `taylm`
`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]`

- **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 Sec. 2.3.2 and for the algorithm based on the Linear Dominated Bounder and the Quadratic Fast Bounder from Sec. 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 using the rules

$$\begin{aligned} \text{max\_order}_{new} &= \max(\text{max\_order}_1, \text{max\_order}_2), \\ \text{tolerance}_{new} &= \min(\text{tolerance}_1, \text{tolerance}_2), \\ \text{eps}_{new} &= \min(\text{eps}_1, \text{eps}_2), \end{aligned} \tag{19}$$

where the subscript *new* refers to the resulting object, and 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 Sec. 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
epsilon = 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,epsilon,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) the performance of Taylor models implemented in CORA with other tools and b) 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 [5, 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 (24), 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 (24) we obtain*

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

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 where affine arithmetic is better than a Taylor model)**.** *Typically, Taylor models provide better results than affine arithmetic, but for the example* $r = \sqrt{x^3 - x}$ *with* $[x] = [2, 3]$ *affine arithmetic is better. However, interval arithmetic provides the best result.*
Interval arithmetic: After inserting the intervals, interval arithmetic returns (see [5, 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 (23), we obtain*

$$\begin{aligned}
[r] =& \sqrt{(2.5 + \epsilon 0.5)^3 - (2.5 + \epsilon 0.5)} \\
=& \sqrt{8.875\epsilon + 13.125 + [0,2]} \\
=& 1.2249\epsilon + 3.6228 + [-1.68727, 0.78250] \\
=& [0.71071, 5.63021].
\end{aligned}$$

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* (23) *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 [5, 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* (6), (7), *and* (8)*:*

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

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*

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

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* [12] and our implementation of affine arithmetic in Sec. 3 with INTLAB [46]. 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 [25]. For our results, the relative precision of an interval x is obtained as

$$[\underline{\delta}, \overline{\delta}] = \left[-\frac{\underline{x} - \underline{x}_{ref}}{\overline{x}_{ref} - \underline{x}_{ref}}, \frac{\overline{x} - \overline{x}_{ref}}{\overline{x}_{ref} - \underline{x}_{ref}} \right] \cdot 100\%, \tag{20}$$

where x_{ref} is a reference value. This measure of the precision is more informative than the scalar overestimation factor used in [32, Eq. 5.1], because it evaluates the precision of the lower and the upper range bounds separately. Another possibility would be to focus on the remainder of the Taylor models; however, by comparing the obtained function bounds, we also evaluate how well the polynomial part can be bounded. 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 [33]. If the computation of the bounds took longer than 10 seconds we terminated it. 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.) as described in Sec. 2.3.1, branch and bound (BnB) as well as advanced branch and bound (BnB adv.) as described in Sec. 2.3.2 and Alg. 1 (with $\epsilon = 0.001$), and our algorithm based on LDB and QFB (LDB/QFB) as described in Sec. 2.3.3 (with $\epsilon = 0.001$).

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, 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; in return, however, they 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.

Table 3: Comparison of Taylor model techniques (part 1). The precision is provided in percent compared to the exact result as shown in (20). The input ranges are presented in Tab. 8 and the examples are taken out of [25].

Bench-mark	**Max. order**	**CORA**					**Flow***
		Interval subs.	**BnB**	**BnB (adv.)**	**LDB/ QFB**	**Zoo (Int. subs.)**	
sin	2	[140, 148]	[140, 148]	[86.7, 148]	[82.0, 148]	[0, 0]	[215, 148]
	5	[166, 146]	[41.5, 8.23]	[6.29, 3.55]	[5.92, 3.54]	[0, 0]	[241, 174]
	10	[162, 146]	[35.1, 3.87]	[0.40, 0.03]	[0.01, 0.01]	[0, 0]	[241, 174]
bspline0	2	[27.4, 0]	[27.4, 0]	[1.33, 0]	[1.00, 0]	[0, 0]	[54.8, 0]
	5	[27.4, 0]	[27.4, 0]	[0, 0]	[0, 0]	[0, 0]	[54.8, 0]
	10	[27.4, 0]	[27.4, 0]	[0, 0]	[0, 0]	[0, 0]	[54.8, 0]
bspline1	2	[0, 30.9]	[0, 30.9]	[0, 3.61]	[0, 2.86]	[0, 0]	[0, 61.7]
	5	[0, 30.9]	[0, 30.9]	[0, 0]	[0, 0]	[0, 0]	[0, 61.7]
	10	[0, 30.9]	[0, 30.9]	[0, 0]	[0, 0]	[0, 0]	[0, 61.7]
bspline2	2	[33.8, 0]	[33.8, 0]	[6.77, 0]	[5.08, 0]	[3.92, 0]	[67.6, 0]
	5	[33.8, 0]	[33.8, 0]	[0, 0]	[0, 0]	[3.92, 0]	[67.6, 0]
	10	[33.8, 0]	[33.8, 0]	[0, 0]	[0, 0]	[3.92, 0]	[67.6, 0]
bspline3	2	[34.9, 0]	[34.9, 0]	[8.40, 0]	[6.41, 0]	[0, 0]	[69.7, 0]
	5	[34.9, 0]	[34.9, 0]	[0, 0]	[0, 0]	[0, 0]	[69.7, 0]
	10	[34.9, 0]	[34.9, 0]	[0, 0]	[0, 0]	[0, 0]	[69.7, 0]
doppler	2	[0.07, 1.58]	[0.07, 1.58]	[0.06, 1.41]	[0.07, 1.58]	[0.07, 0.50]	[0.05, 1.58]
	5	[0.07, 1.58]	[0.07, 1.58]	[0.06, 1.41]	[0.07, 1.58]	[0.07, 0.50]	[0.05, 1.58]
	10	[0.07, 1.58]	[0.07, 1.58]	[0.06, 1.41]	[0.07, 1.58]	[0.07, 0.50]	[0.05, 1.58]
himmilbeau	2	[105, 90.2]	[105, 90.2]	[56.0, 89.5]	-	[105, 12.7]	[167, 90.9]
	5	[105, 90.2]	[29.7, 11.5]	[0.03, 0]	[0, 0]	[105, 12.7]	[167, 90.9]
	10	[105, 90.2]	[29.7, 11.5]	[0.03, 0]	[0, 0]	[105, 12.7]	[167, 90.9]

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

Table 4: Comparison of Taylor model techniques (part 2). The precision is provided in percent compared to the exact result as shown in (20). The input ranges are presented in Tab. 8 and the examples are taken out of [25].

Benchmark	**Max. order**	**CORA**					**Flow***
		Interval subs.	**BnB**	**BnB (adv.)**	**LDB/ QFB**	**Zoo (Int. subs.)**	
kepler0	2	[8.22, 15.9]	[8.22, 15.9]	[1.23, 1.79]	[0, 0]	[8.22, 15.9]	[8.22, 19.2]
	5	[8.22, 15.9]	[8.22, 15.9]	[1.23, 1.79]	[0, 0]	[8.22, 15.9]	[8.22, 19.2]
	10	[8.22, 15.9]	[8.22, 15.9]	[1.23, 1.79]	[0, 0]	[8.22, 15.9]	[8.22, 19.2]
kepler1	2	[11.8, 37.7]	[11.8, 37.7]	[7.36, 4.70]	-	[11.8, 37.7]	[12.6, 51.4]
	5	[11.8, 37.7]	[2.14, 11.9]	[0, 0.02]	[0, 0]	[11.8, 37.7]	[12.6, 51.4]
	10	[11.8, 37.7]	[2.14, 11.9]	[0, 0.02]	[0, 0]	[11.8, 37.7]	[12.6, 51.4]
kepler2	2	[31.7, 42.1]	[31.7, 42.1]	[4.86, 6.16]	-	[31.7, 42.1]	[32.0, 53.8]
	5	[31.7, 42.1]	[15.7, 28.0]	[0, 0.04]	[0, 0]	[31.7, 42.1]	[32.0, 53.8]
	10	[31.7, 42.1]	[15.7, 28.0]	[0, 0.04]	[0, 0]	[31.7, 42.1]	[32.0, 53.8]
rigidBody1	2	[0, 14.7]	[0, 14.7]	[0, 0.01]	[0, 0]	[0, 10.9]	[0, 14.7]
	5	[0, 14.7]	[0, 14.7]	[0, 0.01]	[0, 0]	[0, 10.9]	[0, 14.7]
	10	[0, 14.7]	[0, 14.7]	[0, 0.01]	[0, 0]	[0, 10.9]	[0, 14.7]
rigidBody2	2	[13.5, 3.57]	[13.5, 3.57]	[1.95, 2.12]	-	[13.5, 3.57]	[21.9, 3.57]
	5	[12.6, 2.64]	[10.5, 1.32]	[0.05, 0]	[0, 0]	[12.5, 2.64]	[21.0, 2.64]
	10	[12.5, 2.64]	[10.50, 1.32]	[0.05, 0]	[0, 0]	[12.5, 2.64]	[21.0, 2.64]
turbine1	2	[135, 148]	[123, 140]	[122, 61.3]	-	[135, 2.67]	[1645, 1703]
	5	[20.4, 62.1]	[20.4, 62.1]	[18.4, 7.85]	-	[20.4, 2.67]	[6113, 14542]
	10	[2.29, 49.5]	[2.25, 49.5]	[2.20, 2.98]	-	[2.29, 2.67]	$[1.4, 1.4] \cdot 10^7$
turbine2	2	[124, 152]	[116, 147]	[58.7, 128]	-	[2.72, 152]	[1200, 1203]
	5	[60.0, 66.6]	[28.0, 32.2]	[5.77, 17.3]	-	[2.72, 66.6]	[3296, 4739]
	10	[50.5, 53.4]	[22.3, 18.9]	[0.78, 2.14]	-	[2.72, 53.4]	[51543, 51546]

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.

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

Benchmark	Max. order	CORA					Flow*
		Int. subs.	BnB	BnB (adv.)	LDB/QFB	Zoo	
sin	2	5.121	9.160	20.190	662.538	5.444	0.760
	5	10.711	29.834	45.990	761.600	11.338	0.602
	10	20.686	54.855	83.278	1183.286	21.189	2.038
bspline0	2	3.398	6.184	24.351	763.741	5.123	0.315
	5	3.215	5.848	28.387	46.442	4.931	0.253
	10	3.232	5.858	28.377	46.453	4.955	0.229
bspline1	2	5.268	8.977	28.578	1374.154	6.748	0.393
	5	5.123	8.825	34.226	60.206	6.474	0.343
	10	5.098	8.645	34.128	60.103	6.912	0.341
bspline2	2	6.408	9.809	29.103	1372.837	7.808	0.410
	5	5.918	10.221	34.901	60.763	7.656	0.368
	10	5.952	9.435	34.487	61.052	7.668	0.381
bspline3	2	3.394	5.865	22.618	757.405	3.953	0.297
	5	2.922	5.546	26.177	56.131	3.737	0.250
	10	2.891	5.944	26.403	56.403	4.116	0.248
doppler	2	11.861	33.916	168.653	50.641	14.419	2.254
	5	19.447	63.644	269.977	77.486	22.072	15.275
	10	29.076	90.646	300.920	120.311	31.661	16.039
himmilbeau	2	10.092	20.671	69.174	$> 10^4$	13.276	0.368
	5	9.694	44.447	138.585	200.525	13.034	0.304
	10	10.440	44.718	139.237	199.924	13.653	0.300
kepler0	2	10.375	24.926	79.291	204.928	13.930	0.259
	5	9.968	24.936	79.315	205.600	13.613	0.248
	10	10.486	24.964	78.712	206.022	13.862	0.246
kepler1	2	16.325	37.301	200.240	$> 10^4$	22.034	0.488
	5	16.142	90.636	707.967	286.366	21.724	0.419
	10	16.162	90.168	707.910	287.493	21.673	0.419
kepler2	2	27.233	75.755	2162.720	$> 10^4$	35.211	1.242
	5	26.249	230.252	2842.870	2788.519	34.213	0.947
	10	26.267	230.206	2840.406	2777.606	33.811	0.927
rigidBody1	2	3.803	7.505	47.418	17.503	5.824	0.128
	5	3.815	7.571	47.543	17.514	5.800	0.096
	10	3.825	7.582	46.938	17.538	5.898	0.093
rigidBody2	2	10.660	23.180	175.691	$> 10^4$	13.787	0.403
	5	9.841	28.158	273.927	152.315	13.071	0.257
	10	9.827	29.152	274.003	152.527	13.004	0.254
turbine1	2	38.963	94.590	445.262	$> 10^4$	44.399	1.254
	5	71.851	122.223	1390.662	$> 10^4$	74.605	2.279
	10	123.518	210.891	3905.585	$> 10^4$	127.326	4.822
turbine2	2	24.550	53.024	487.946	$> 10^4$	27.044	0.541
	5	42.189	168.089	2929.171	$> 10^4$	48.433	1.498
	10	75.779	257.485	6138.943	$> 10^4$	75.459	3.855

Table 6: Comparison of affine arithmetic. The precision is provided in percent compared to the exact result as shown in (20). The input ranges are presented in Tab. 8 and the examples are taken out of [25].

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 is shown in Tab. 6.

Bench-mark	**CORA**				
	Int. subs.	**BnB**	**BnB (adv.)**	**Interval**	**IntLab**
sin	3.293	6.540	11.332	0.192	10.649
bspline0	3.921	7.084	14.532	0.111	6.245
bspline1	6.305	11.060	21.072	0.136	8.966
bspline2	7.322	11.543	22.552	0.132	10.235
bspline3	3.544	6.476	13.892	0.095	4.071
doppler	9.728	24.025	35.102	0.197	10.624
himmilbeau	12.429	23.759	43.541	0.146	13.347
kepler0	12.546	28.480	39.405	0.216	12.799
kepler1	19.186	42.304	63.121	0.237	21.082
kepler2	30.710	79.738	129.766	0.348	33.086
rigidBody1	4.703	8.768	13.309	0.106	5.735
rigidBody2	11.942	24.764	41.808	0.161	14.412
turbine1	30.123	48.471	76.614	0.300	19.165
turbine2	19.423	34.023	53.376	0.219	12.887

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

References

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

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

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

[4] 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.

[5] 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.

[6] 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.

[7] A. Balluchi, A. Casagrande, P. Collins, A. Ferrari, T. Villa, and A. L. Sangiovanni-Vincentelli. Ariadne: a framework for reachability analysis of hybrid automata. In *Proc. of the 17th International Symposium on Mathematical Theory of Networks and Systems*, pages 1259–1267, 2006.

[8] L. Benvenuti, D. Bresolin, P. Collins, A. Ferrari, L. Geretti, and T. Villa. Assume-guarantee verification of nonlinear hybrid systems with ARIADNE. *International Journal of Robust and Nonlinear Control*, 24:699–724, 2014.

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

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

[11] X. Chen. *Reachability Analysis of Non-Linear Hybrid Systems Using Taylor Models*. PhD thesis, RWTH Aachen University, 2015.

[12] 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.

[13] 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.

[14] 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.

[15] P. Collins, M. Niqui, and N. Revol. A Taylor function calculus for hybrid system analysis: Validation in Coq. In *Third International Workshop on Numerical Software Verification*, 2010.

[16] D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties, and Algorithms*. Springer, 2007.

[17] L. H. de Figueiredo and J. Stolfi. Affine arithmetic: Concepts and applications. *Numerical Algorithms*, 37(1-4):147–158, 2004.

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

[19] 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.

[20] 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.

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

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

[23] 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.

[24] 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.

[25] A. Izycheva and E. 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.

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

[27] 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.

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

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

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

[31] K. Makino and M. Berz. *Computational Differentiation: Techniques, Applications, and Tools*, chapter Remainder Differential Algebras and Their Applications, pages 63–75. SIAM, 1996.

[32] 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.

[33] K. Makino and M. Berz. Verified global optimization with Taylor model based range bounders. *Transactions on Computers*, 4(11):1611–1618, 2005.

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

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

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

[37] 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.

[38] 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.

[39] 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.

[40] R. D. Neidinger. Directions for computing truncated multivariate Taylor series. *Mathematics of Computation*, 74(249):321–340, 2004.

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

[42] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 2007.

[43] 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.

[44] 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.

[45] 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.

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

[47] 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.

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

[49] 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 $\forall x \in [x]$ (see Def. 2), 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 $B(\cdot)$ returns over-approximative bounds for Taylor models.

Monotonic functions:

- Exponential from [32, Eq (2.2)–(2.4)]:

$$\begin{aligned}\exp(T) = &\exp(c_f) \cdot \left[1 + \tilde{T} + \frac{1}{2!}\tilde{T}^2 + ... + \frac{1}{q!}\tilde{T}^q\right] \\ &+ \frac{\exp(c_f)}{(q+1)!}\Big(B(\tilde{T}^{q+1}) \cdot \exp([0,1] \cdot B(\tilde{T})\Big).\end{aligned} \tag{21}$$

- Logarithm from [32, p. 386] is defined under the condition$B(T) \subset (0, \infty)$:

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

- Square root from [32, p. 386] is defined under the condition$B(T) \subset (0, \infty)$:

$$\begin{aligned}\sqrt{T} = &\sqrt{c_f}\Big[1 + \frac{1}{2}\frac{\tilde{T}}{c_f} - \frac{1}{2!2^2}\frac{\tilde{T}^2}{c_f^2} + ... + (-1)^{q-1}\frac{(2q-3)!!}{q!2^q}\frac{\tilde{T}^q}{c_f^q}\Big] \\ &+ (-1)^q\sqrt{c_f}\frac{(2q-1)!!}{(q+1)!2^{q+1}}\frac{B(\tilde{T}^{q+1})}{c_f^{q+1}}\frac{1}{(1 + [0,1] \cdot \frac{B(\tilde{T})}{c_f})^{q+1/2}}.\end{aligned} \tag{23}$$

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

Trigonometric functions:

- Sine from [32, p. 387]:

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

  where

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

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

- Cosine from [32, p. 387]:

$$\begin{aligned}\cos(T) = &\cos(c_f) - \sin(c_f) \cdot \tilde{T} - \frac{1}{2!}\cos(c_f) \cdot \tilde{T}^2 + \frac{1}{3!}\sin(c_f) \cdot \tilde{T}^3 + ... \\ &+ \frac{1}{(q+1)!}B(\tilde{T}^{q+1}) \cdot J,\end{aligned} \tag{25}$$

where

$$J = \begin{cases} -J_0 & \text{if} \mod (q,4) = 0 \text{ or } 1, \\ J_0 & \text{else}, \end{cases}$$
$$J_0 = \begin{cases} \sin(c_f + [0,1] \cdot B(\tilde{T})) & \text{if } q \text{ is even}, \\ \cos(c_f + [0,1] \cdot B(\tilde{T})) & \text{else}. \end{cases}$$

- Tangent:

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

Hyperbolic functions:

- Hyperbolic sine from [32, 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 [32, 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:

$$\tanh(T) = \frac{\sinh(T)}{\cosh(T)}. \tag{29}$$

Inverse trigonometric functions:

- Arcsine from [32, p. 388] under the condition of $B(T) \subset (-1, 1)$:

$$\begin{aligned} \arcsin(T) = &\arcsin(c_f) + G + \frac{1}{3!}G^3 + \frac{3^2}{5!}G^5 + \frac{3^2 5^2}{7!}G^7 + ... \\ &+ \frac{1}{(q+1)!} B(G)^{q+1} \arcsin^{(q+1)}([0,1] \cdot B(G)), \end{aligned} \tag{30}$$

where

$$G = T \cdot \sqrt{1 - c_f^2} - c_f \cdot \sqrt{1 - T^2}$$

and $\arcsin^{(1)}(0) = 1$, $\arcsin^{(2)}(0) = 0$, $\arcsin^{(3)}(0) = 1$, and $\arcsin^{(q)}(0) = (q-2)^2 \arcsin^{(q-2)}(0)$.

- Arccosine from [32, p. 389]:

$$\arccos(T) = \frac{\pi}{2} - \arcsin(T). \tag{31}$$

- Arctangent from [32, p. 389]:

$$\begin{aligned} \arctan(T) = &\arctan(c_f) + G - \frac{1}{3}G^3 + \frac{1}{5}G^5 - \frac{1}{7}G^7 + ... \\ &+ \frac{1}{(q+1)} B(G)^{q+1} \cos^{q+1}\Big(\arctan([0,1] \cdot B(G))\Big) \\ &\cdot \sin\Big((q+1)(\arctan([0,1] \cdot B(G)) + \frac{\pi}{2})\Big), \end{aligned} \tag{32}$$

where

$$G = \frac{T - c_f}{1 + c_f \cdot T}.$$

B Tight Bounds for the Taylor Model Inverse Operation

As shown in (10), 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

$$L_q(B(\tilde{T}), c_f) = (-1)^{q+1} \frac{B(\tilde{T})^{q+1}}{c_f^{q+2}} \frac{1}{\left(1 + [0,1] \cdot \frac{B(\tilde{T})}{c_f}\right)^{q+2}}, \; c_f = P(0), \; \tilde{T} = T - c_f, \tag{33}$$

thereby represents an over-approximation of the exact integral remainder

$$R_q(x, c_f) = \int_{c_f}^{x} f^{(q+1)}(t) \frac{(x-t)^q}{q!} dt, \quad x \in B(T), \tag{34}$$

where $f^{(i)}$ represents the i-th derivative of the function $f(\cdot)$. Evaluation of the Lagrange remainder (33) for a Taylor model can lead to very large over-approximations, as the following example demonstrates:

$$\begin{aligned} &T = 0.45 + 0.35x + [0,0], \; x \in [-1,1], \\ &B(T) = [0.1, 0.8], \; c_f = 0.45, \; B(\tilde{T}) = [-0.35, 0.35], \\ &\frac{1}{B(T)} = [1.25, 10], \\ &L_{q=6}([-0.35, 0.35], 0.45) = [-64339.29688, 64339.29688]. \end{aligned} \tag{35}$$

One solution is to analytically solve the integral in (34) for the function $f(x) = \frac{1}{x}$, which results in an equation for the exact remainder:

$$\begin{aligned} R_q(x, c_f) &= \underbrace{(q+1) \cdot (-1)^{(q+1)}}_{:=a} \int_{c_f}^{x} \frac{(x-t)^q}{t^{(q+2)}} dt, \\ &= -\frac{a}{q+1} \frac{(x-t)^q}{t^{(q+1)}} \Big|_{c_f}^{x} - \frac{a \cdot q}{q+1} \int_{c_f}^{x} \frac{(x-t)^{(q-1)}}{t^{(q+1)}} dt, \\ &= \frac{a}{q+1} \frac{(x-c_f)^q}{c_f^{(q+1)}} - \frac{q}{q+1} \cdot R_{q-1}(x, c_f) \ , \end{aligned} \tag{36}$$

with

$$R_0(x, c_f) = a \cdot \int_{c_f}^{x} \frac{1}{t^2} dt = -\frac{a}{t} \Big|_{c_f}^{x} = \frac{a}{c_f} - \frac{a}{x}. \tag{37}$$

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 (33). 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 (33) is faster than the evaluation of the exact remainder (36), 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:

$$rem = \begin{cases} L_q(B(\tilde{T}), c_f) & \text{if } L_q(B(\tilde{T}), c_f) \subset \frac{1}{B(T)} \\ R_q(B(T), c_f) & \text{otherwise} \end{cases}, \tag{38}$$

where $c_f = P(0)$ and $\tilde{T} = T - c_f$.

C Examples

This section presents the results of several examples evaluated in CORA:

```
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);  % Taylor model with order 6 and name a1
b2 = taylm(a2, 6);  % Taylor model with order 6 and name a2
b3 = taylm(a3, 6);  % Taylor model with order 6 and name a3
b4 = taylm(a4, 6);  % Taylor model with order 6 and name 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
```

The resulting workspace output is:

```
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]
```