Certified, Efficient and Sharp Univariate Taylor Models in COQ

Érik Martin-Dorel, Micaela Mayero, Ioana Pasca, Laurence Rideau, Laurent Théry

Certified, Efficient and Sharp Univariate Taylor Models in COQ

Érik Martin-Dorel, Laurence Rideau, and Laurent Théry
Inria Sophia Antipolis
2004 route des Lucioles, BP 93
06902 Sophia Antipolis Cedex, France
Email: erik.martin-dorel@ens-lyon.org
Email: laurence.rideau@inria.fr
Email: laurent.thery@inria.fr

Micaela Mayero
Université Paris 13
LIPN (UMR CNRS 7030), Institut Galilée
99 avenue Jean-Baptiste Clément
93430 Villetaneuse, France
Email: micaela.mayero@lipn.univ-paris13.fr

Ioana Paşca
Université Montpellier 2
IUT de Nîmes
8 rue Jules Raimu
30907 Nîmes Cedex 2, France
Email: ioana.pasca@iut-nimes.fr

***Abstract*—We present a library for univariate Taylor models that has been developed with the COQ proof assistant. Each algorithm of this library is executable and has been formally proved correct. Using this library, one can then effectively compute rigorous and sharp approximations of univariate functions composed of usual functions such as $1/x$, $\sqrt{x}$, e^x, $\sin x$ among others. In this paper, we present the key parts of the formalisation as well as of the proofs of correctness, and we evaluate the quality of our certified library on a set of examples.**

I. Introduction and Motivations

Polynomial approximations are a practical way to represent real-valued functions. In fact, on processors, where the only primitive arithmetic operations are $+$, $-$, and $\times$, they are the only effective way to compute real-valued functions. The quality of the approximation naturally comes into play: being able to guarantee some bounds on the error that occurs when using the approximation instead of the real function is mandatory for the reliability of numerical software. Yet bounds are not always available and when so they are often very difficult to be proved formally.

The work presented here addresses this issue of reliability. We provide a systematic way to formally prove the error bounds of some specific polynomial approximations. This is done in the COQ system but the same approach could be implemented in any other proof assistant. Our starting point is the notion of *rigorous polynomial approximations* (RPAs), which consists of pairs $(P, \mathbf{\Delta})$ where P is a polynomial in a given basis and $\mathbf{\Delta}$ an interval error bound. Several symbolic-numeric techniques that rely on such a data type have been presented in [1]. They heavily rely on interval arithmetic. For instance, most of these algorithms manipulate polynomials with *tight interval coefficients*. Also, rounding errors that may occur when computing the polynomial approximation can easily be handled in this setting. In the following, we will especially focus on RPAs in the Taylor polynomial basis, which we will refer to as *Taylor models* (TMs), borrowing the term coined by Berz and Makino [2], [3].

We formalise univariate Taylor models in the COQ proof assistant, aiming at:

- *Genericity:* the formalisation should be modular (to easily switch implementation of data structures) and extensible (to easily add new functions by specialising some generic algorithms);
- *Efficiency:* computing the approximations (even if performed in a trusted environment with restricted computing power) should be reasonably fast;
- *Correctness:* no underestimate is possible, the computed error-bounds should be formally proved correct;
- *Sharpness:* the algorithms should lead (most of the time) to sharp bounds.

A preliminary version of this work was already presented in [4]. It was composed of a first implementation of the algorithms in order to evaluate the feasibility and accuracy of what could be obtained while computing within COQ. The main achievement of what is presented here is that, now, a machine-checked correctness proof is attached to each of these algorithms. Also, some major improvements in the algorithms have been made in order to get tighter error bounds for basic functions as well as for the division of Taylor models.

Our implementation of Taylor models is composed of a set of models for basic functions (exp, sin, $\sqrt{\cdot}$, and others) and another set of algorithms that are used to combine these models (for addition, multiplication, composition, or division of two functions). We start, in Section II, by describing the general framework of Taylor models as well as the formal tools that are needed for their implementation and proof. In Section III, we present in detail the issues related to Taylor models for basic functions. We then present Taylor models for composite functions in Section IV. In Section V, we provide some benchmarks, including a comparison with respect to [4]. Finally, Section VI relates our work with other approaches.

This work was supported by the TaMaDi project of the French *Agence Nationale de la Recherche*.

II. Generic Taylor Models and their Formalisation

Given a point x_0 and an interval $\boldsymbol{I}$ around this point, a Taylor model usually consists of a pair $(P, \boldsymbol{\Delta})$ where P is a polynomial in the Taylor basis around x_0 and $\boldsymbol{\Delta}$ is an interval. In the following, we use the convention that intervals as well as polynomials with interval coefficients will always be printed in bold. A Taylor model approximates a whole set of functions : the functions that are at a distance of less than $\boldsymbol{\Delta}$ over $\boldsymbol{I}$. More formally, $(P, \boldsymbol{\Delta})$ approximates f if and only if $\forall x \in \boldsymbol{I},\ f(x) - P(x) \in \boldsymbol{\Delta}$. Starting from a given function f, a natural way to derive a Taylor model $(P, \boldsymbol{\Delta})$ is to use the Taylor–Lagrange formula

Theorem 1: If f is a real-valued function that is $n+1$ times differentiable on an interval $\boldsymbol{I}$ and x_0 is a point of $\boldsymbol{I}$ then we can consider the n^{th}-order Taylor expansion of f around x_0. For all x in $\boldsymbol{I}$, there exists a ξ between x_0 and x such that

$$f(x) = \underbrace{\left(\sum_{i=0}^{n} \frac{f^{(i)}(x_0)}{i!}\,(x-x_0)^i\right)}_{P(x)} + \underbrace{\frac{f^{(n+1)}(\xi)}{(n+1)!}\,(x-x_0)^{n+1}}_{\Delta(x_0,x,\xi)}.$$

If $f^{(n+1)}(\xi)$ can be bounded over the interval $\boldsymbol{I}$, it is then easy to compute an interval $\boldsymbol{\Delta}$ so that $\Delta(x_0, x, \xi) \in \boldsymbol{\Delta}$ for $x, \xi \in \boldsymbol{I}$ and build the approximation $(P, \boldsymbol{\Delta})$. Doing so, $\boldsymbol{\Delta}$ accumulates the "method error". Note that, in practice, it is not always possible to compute an approximation with exact numbers. The coefficients of the polynomial need to be rounded to rational numbers, or even more restrictively to floating-point numbers. This is not a problem for Taylor models since rounding errors can always be incorporated into the $\boldsymbol{\Delta}$. The approximation is then less accurate but still valid.

As explained in the introduction, a key ingredient to get effective Taylor models is to use interval arithmetic. In this setting, the polynomial of a Taylor model has tight interval coefficients. As usual, we represent polynomials as lists of coefficients, but we do not enforce that the leading coefficient is non-zero. In the following, we will then talk about the "order" of a Taylor model rather than its "degree" and reason in terms of the size of the corresponding list of coefficients. An n^{th}-order Taylor model $(\boldsymbol{P}, \boldsymbol{\Delta})$ will satisfy $\mathrm{size}(\boldsymbol{P}) = n+1$. Moreover, the "expansion point" will not be a point x_0 but rather a small interval $\boldsymbol{x_0}$. Taylor models can thus be built around an irrational expansion point. The definition of validity is then rephrased as

Definition 1: $(\boldsymbol{P}, \boldsymbol{\Delta})$ is a valid TM for $f : \mathbb{R} \to \mathbb{R}$ over $\boldsymbol{I}$ around $\boldsymbol{x_0}$ if we have $\boldsymbol{x_0} \subset \boldsymbol{I}$, $0 \in \boldsymbol{\Delta}$, and for all $\xi_0 \in \boldsymbol{x_0}$, there exists a polynomial Q over $\mathbb{R}$ such that

$$\begin{cases} \mathrm{size}\, Q = \mathrm{size}\, \boldsymbol{P}, \\ \forall k < \mathrm{size}\, \boldsymbol{P}, \quad Q_k \in \boldsymbol{P}_k, \\ \forall x \in \boldsymbol{I}, \quad f(x) - \sum_{i<\mathrm{size}\, Q} Q_i \cdot (x-\xi_0)^i \in \boldsymbol{\Delta}. \end{cases}$$

Formalising this definition in COQ is straightforward. The type $\mathbb{R}$ in COQ is defined *via* a classical axiomatisation of an Archimedean ordered complete field [5] in the Reals standard library. It provides all the basic theorems usually involved in real analysis, e.g., about differentials or integrals, but no computation is available.

Table I. Semantics of the inductive type for intervals

COQ term	mathematical meaning
`Ibnd (Xreal` x`) (Xreal` y`)` with $x > y$	$\varnothing$
`Ibnd (Xreal` x`) (Xreal` y`)` with $x \leqslant y$	$[x, y]$
`Ibnd (Xreal` x`) Xnan`	$[x, +\infty[$
`Ibnd Xnan (Xreal` y`)`	$]-\infty, y]$
`Ibnd Xnan Xnan`	$\mathbb{R}$
`Inan`	$\mathbb{R} \cup \{\mathrm{NaN}\}$

For the intervals, we use the Coq.Interval library [6]. It provides abstract data types for an interval arithmetic that handles undefined values (NaNs):

```
R̄ := Xnan | Xreal (r : R).
interval := Inan | Ibnd (l u : R̄).
```

The semantics of these "NaN" values is summarised in Table I. Note that, in this context, we have `Xnan` $\in$ `Inan` but `Xnan` $\notin$ `(Ibnd Xnan Xnan)`. In the formal development, Definition 1 is formalised as a predicate `i_validTM`, with two slight differences: f has type $\overline{\mathbb{R}} \to \overline{\mathbb{R}}$ instead of $\mathbb{R} \to \mathbb{R}$, and the coefficients of Q are in $\overline{\mathbb{R}}$ instead of $\mathbb{R}$.

For the polynomials, we use the library provided by the SSReflect extension [7]. This extension provides its own tactic language and libraries, and has been designed to make it easier to formalise mathematics. In this work, two libraries are of special interest. A first one defines standard algebraic structures and the theorems associated to them. A second one provides a very powerful theory `bigop` for iterated "big" operations like summations. For polynomial approximations over $\overline{\mathbb{R}}$, the theory `bigop` was quite handy to use, except that we had to circumvent the fact that in $\overline{\mathbb{R}}$, the neutral element of addition, `(Xreal 0)`, is not an absorbing element for the multiplication. This issue is directly related to the presence of undefined values such as `Xnan` and `Inan` in the formalism of Coq.Interval. Some careful case analyses are then frequently required when developing our correctness proofs.

The Coq.Interval library also provides ways to instantiate intervals and in particular to get intervals we can compute with. One such instantiation uses two floating-point numbers to represent the bounds of the interval. The floating-point numbers are also defined within Coq.Interval either by pairs of integers (a mantissa and an exponent) or by an undefined value :

```
float := Fnan | Float (m e : Z).
```

The operations on these numbers are defined in Coq.Interval (such as $+$, $-$, $\times$, $\div$, $\sqrt{\cdot}$, comparison, etc.) but they do not assume any bound on the exponent. Yet they take a *precision* argument that is used for rounding the mantissa of the result. Using such an instantiation, it is then possible to specialise the previous definition that uses intervals to polynomials with simple floating-point coefficients, as explained below.

From any pair $(\boldsymbol{P}, \boldsymbol{\Delta})$ that is a valid TM according to Definition 1, we can easily produce a pair $(P, \boldsymbol{\Delta'})$ where P is polynomial with floating-point coefficients satisfying:

Definition 2: $(P, \boldsymbol{\Delta'})$ is a valid TM for $f : \overline{\mathbb{R}} \to \overline{\mathbb{R}}$ over $\boldsymbol{I}$ around $\boldsymbol{x_0}$ if we have $\boldsymbol{x_0} \subset \boldsymbol{I}$, $0 \in \boldsymbol{\Delta'}$, and

$$\forall \xi_0 \in \boldsymbol{x_0}, \quad \forall x \in \boldsymbol{I}, \quad f(x) - \sum\nolimits_{i < \mathrm{size}\, P} P_i \cdot (x - \xi_0)^i \in \boldsymbol{\Delta'}.$$

It suffices to take the midpoint of each interval coefficient of $\boldsymbol{P}$ and add the corresponding errors to $\boldsymbol{\Delta}$. In the formal development, Definition 2 is formalised as a predicate `f_validTM`, and the function `i2f_tm` transforms $(\boldsymbol{P}, \boldsymbol{\Delta})$ into $(P, \boldsymbol{\Delta'})$.

Generic implementation of polynomials, coefficients and intervals: COQ provides three mechanisms for modularisation: *type classes*, *structures*, and *modules*. Modules are less generic than the other two (which are first-class citizens) but they have a better computational behaviour: module applications are performed statically, so the code that is executed is often more compact. More details can be found in [4] and in [8]. As our generic implementation only requires simple parametricity, we have been using modules only. First, abstract interfaces called `Module Type`s are defined. Then concrete "instances" of these abstract interfaces are created in the form of `Module`s that implement all the fields of the `Module Type`. The definition of `Module`s can be parameterised by other `Module`s. These parameterised modules are crucial to factorise code in our structures.

We describe abstract interfaces for polynomials and for their coefficients in the form of `Module Type`s. The interface for coefficients contains the common base of all kinds of "computable real numbers" we may want to use. Usually coefficients of a polynomial are taken in a ring. We cannot do this here. For example, addition of two intervals with floating-point bounds is not associative. Therefore, our abstract interface for coefficients only contains the required *operations* (addition, multiplication, etc.). The usual *properties* (associativity, distributivity, etc.) are specified in a specialised abstract interface for exact arithmetic. The case of abstract polynomials is similar. They are also declared as a `Module Type` but this time parameterised by the coefficients. The interface only contains the operations on polynomials (addition, evaluation, iterator, etc.) along with the properties that are satisfied by all common instantiations of polynomials: these properties are essentially the size of the various operations on polynomials, an induction scheme based on the zero-size polynomial and the function $(c, p) \mapsto c + X \cdot p$, and the behaviour of a "fold-right" iterator with respect to these two constructors. Finally for intervals, we directly use the `IntervalOps` abstract interface provided by the Coq.Interval library.

We are now able to give the definition of our RPA.

```
Module RigPolyApprox
  (C: BaseOps)(Pol: PolyOps C)(I: IntervalOps).
Record rpa := RPA { approx: Pol.T; error: I.type }.
```

The module is parameterised by `C` (the coefficients), by `Pol` (the polynomials with coefficients in `C`), and by `I` (the intervals). An `rpa` structure consists of a polynomial `approx` and an interval `error`. Taylor models will be defined as an instance of the generic `rpa` structure.

III. Taylor Models for Basic Functions

A. Mathematical setup

In order to compute a Taylor model associated with an expression $f(x)$, we can rely on Theorem 1 and take an interval enclosure of $\Delta(x_0, x, \xi)$ with respect to $x_0 \in \boldsymbol{x_0}$, $x \in \boldsymbol{I}$ and $\xi \in \boldsymbol{I}$ to get a value for $\boldsymbol{\Delta}$. Yet this strategy would yield very pessimistic error bounds for composite functions such as $x \mapsto e^{1/\cos x}$ [9]. Hence, it is better to follow a two-level strategy:

- For each basic function ($\sqrt{x}$, e^x, $\sin x$, etc.) involved in the leafs of the expression tree of $f(x)$, compute a Taylor model using, for instance, a naive enclosure of the Taylor–Lagrange remainder;
- Combine the various Taylor models computed for the atoms of expression $f(x)$ using the algebraic rules that correspond to operations $+$, $\times$, as well as $\circ$ (composition).

For example, in order to compute a Taylor model for the function $x \mapsto e^{1/\cos x}$ we can notice that $e^{1/\cos x} = (\exp \circ \mathrm{inv} \circ \cos)(x)$ and first compute a Taylor model for $\cos x$, then deduce a Taylor model for the overall function by using twice the algebraic rule for composition with the inverse function and the exponential. We will give more details on these rules in Section IV.

B. Formal setup

In this section we give some insight on our formalisation of Taylor models for basic functions. The focus is on the design of the generic algorithm and its correctness proof.

A first building block is composed by the formalisation of Theorem 1. This is the topic of the upcoming Section III-B1. Section III-B2 is devoted to the formalisation of an efficient computation of Taylor polynomials. Section III-B3 focuses on the generic computation of sharp error bounds for these polynomials. Finally, Section III-B4 deals with the specialisation of our generic framework to concrete functions.

1) Formal proof of the Taylor–Lagrange theorem: As the Taylor–Lagrange theorem (Theorem 1) was not available in the Reals standard library of COQ, our first task has been to prove this result. In the Reals library, we can talk about the derivative of a function only when having a proof that the function is actually differentiable. This is often annoying when doing proofs involving derivatives. Here, in order to talk about the $(n+1)^{th}$-order derivative of a given function f over an interval $[a, b]$, we consider a total function $D : \mathbb{N} \to \mathbb{R} \to \mathbb{R}$ that satisfies the following properties: (i) $D_0 = f$; (ii) $\forall k \leqslant n,\ D_k$ is differentiable over $]a, b[$ and its derivative is D_{k+1}; (iii) $\forall k \leqslant n,\ D_k$ is continuous over $[a, b]$. Next, for some given x_0, $x \in [a, b]$, we define an auxiliary function g, differentiable over $]a, b[$:

$$g(y) := D_0(x) - \left(\sum_{i=0}^{n} \frac{D_i(y)}{i!}(x - y)^i\right) - c \cdot (x - y)^{n+1},$$

where the constant c is chosen so that $g(x_0) = 0$ holds. As we also have $g(x) = 0$, we can apply Rolle's theorem to the function g. This yields a ξ between x_0 and x such that $g'(\xi) = 0$, and some elementary calculation leads to $c = \frac{D_{n+1}(\xi)}{(n+1)!}$. Combined with $g(x_0) = 0$, this proves Theorem 1.

In our formalisation we use real numbers extended with a NaN value. This means that a specific version of the Taylor–Lagrange theorem is needed for the type $\overline{\mathbb{R}}$ where the variables x_0 and x have type $\overline{\mathbb{R}}$, but the witness ξ is in in $\mathbb{R}$. The advantage of working in $\overline{\mathbb{R}}$ is that derivatives can be handled more easily : if a function is not differentiable at a point, its derivative simply takes the value `Xnan` at this point. The drawback, of course, is that extra case analyses are often needed in proofs in order to coverall potential `Xnan` values.

2) Efficient computation of Taylor polynomials for the class of D-finite functions: In order to implement Taylor models for basic functions in a generic way, we consider a large class of functions whose successive derivatives can be computed in a uniform and efficient way. We thus focus on the so-called *D-finite* functions. They correspond to solutions of homogeneous *linear ordinary differential equations* (LODEs) with polynomial coefficients, that is equations of the form

$$a_r(x)y^{(r)}(x) + \cdots + a_1(x)y'(x) + a_0(x)y(x) = 0,$$

where the a_k are univariate polynomials over a given field. Most common functions are D-finite, while a simple counter-example is tan. A nice feature of this class of functions is that the Taylor series coefficients of any D-finite function satisfy a linear recurrence relation.

We thus provide the functions `trec1`, `trec2` and `trecN` that allow one to compute polynomials whose coefficients are given by a first order, second order, or N^{th}-order recurrence along with the appropriate number of initial conditions. Having specific functions for recurrences of small order makes it possible to have an optimised implementation for these frequent cases. For instance, `trec1` is a function defined using tail-recursion, which has type

```
trec1 : (T → ℕ → T) → T → ℕ → Pol.T
```

and returns the polynomial $\texttt{trec1}(G, c_0, n) = \sum_{k=0}^{n} c_k X^k$ where $c_k = G(c_{k-1}, k)$ for all $k \geqslant 1$.

3) Zumkeller's technique and sharp error bounds: Let f be a basic function, and assume that we have a function $\boldsymbol{T}$: `T` $\to \mathbb{N} \to$ `Pol.T` (possibly based on the D-finite recurrences presented in the previous section) to compute the n^{th}-order Taylor polynomial of f around a given point. Then we can compute a Taylor model of f by using Algorithm 1 below.

Algorithm 1 (Zumkeller's technique):

Input: $\boldsymbol{F}$: interval evaluator for function f
Input: $\boldsymbol{T}(\boldsymbol{y_0}, n)$: n^{th}-order Taylor polynomial around $\boldsymbol{y_0}$
Input: $\boldsymbol{x_0} \subset \boldsymbol{I}$ and $n \in \mathbb{N}$
Output: $(\boldsymbol{P}, \boldsymbol{\Delta})$

1: $\boldsymbol{P} \leftarrow \boldsymbol{T}(\boldsymbol{x_0}, n)$
2: $\boldsymbol{\Gamma} \leftarrow \left[X^{n+1}\right] \boldsymbol{T}(\boldsymbol{I}, n+1)$
3: **if** $(\sup \boldsymbol{\Gamma} \leqslant 0$ **or** $\inf \boldsymbol{\Gamma} \geqslant 0)$ **and** $\boldsymbol{I}$ is bounded **then**
4: $\quad \boldsymbol{a} \leftarrow [\inf \boldsymbol{I}, \inf \boldsymbol{I}]$
5: $\quad \boldsymbol{b} \leftarrow [\sup \boldsymbol{I}, \sup \boldsymbol{I}]$
6: $\quad \boldsymbol{\Delta_a} \leftarrow \boldsymbol{F}(\boldsymbol{a}) - \boldsymbol{P}(\boldsymbol{a} - \boldsymbol{x_0})$
7: $\quad \boldsymbol{\Delta_b} \leftarrow \boldsymbol{F}(\boldsymbol{b}) - \boldsymbol{P}(\boldsymbol{b} - \boldsymbol{x_0})$
8: $\quad \boldsymbol{\Delta_{x_0}} \leftarrow \boldsymbol{F}(\boldsymbol{x_0}) - \boldsymbol{P}(\boldsymbol{x_0} - \boldsymbol{x_0})$
9: $\quad \boldsymbol{\Delta} \leftarrow \boldsymbol{\Delta_a} \vee \boldsymbol{\Delta_b} \vee \boldsymbol{\Delta_{x_0}}$
10: **else**
11: $\quad \boldsymbol{\Delta} \leftarrow \boldsymbol{\Gamma} \times (\boldsymbol{I} - \boldsymbol{x_0})^{n+1}$
12: **end if**

The notation $[X^n]Q$ denotes the coefficient of Q of degree n and the notation $\boldsymbol{\Delta_a} \vee \boldsymbol{\Delta_b}$ denotes the smallest interval that contains the two intervals $\boldsymbol{\Delta_a}$ and $\boldsymbol{\Delta_b}$.

The computation of $\boldsymbol{P}$ is straightforward but that of $\boldsymbol{\Delta}$ deserves some detailed explanation. To start with, the **else** branch computes a naive enclosure of the Taylor–Lagrange remainder. In particular, due to the very similar shape of this enclosure with respect to a Taylor coefficient of degree $(n+1)$ "around $\boldsymbol{I}$", we compute both this enclosure and the Taylor coefficients of $\boldsymbol{P}$ in a uniform way (cf. Line 1 and Line 2).

Now let us focus on the **then** branch of Algorithm 1. This part of the algorithm is an optimisation with respect to the implementation that was presented in [4]. The expressions $\boldsymbol{\Delta_{x_0}}$, $\boldsymbol{\Delta_a}$ and $\boldsymbol{\Delta_b}$ correspond to *the evaluation of the Taylor–Lagrange remainder* on the small intervals $\boldsymbol{x_0}$, $\boldsymbol{a}$ and $\boldsymbol{b}$ (composed of the endpoints of $\boldsymbol{I}$). As a result, beyond the slight rounding errors that may occur in these evaluations, the value of $\boldsymbol{\Delta}$ that is obtained in this branch is the sharpest possible bound we could achieve. It then remains to ensure that this value is not underestimated.

We give below an overview of the main steps involved in its formal proof, which relies on the Proposition 2.2.1 in Mioara Joldeş' thesis [1], itself based on Lemma 5.12 in Roland Zumkeller's thesis [10]. Let us denote the Taylor–Lagrange remainder of f by

$$R_n(f, \xi_0)(x) := f(x) - \sum_{i=0}^{n} \frac{f^{(i)}(\xi_0)}{i!} \cdot (x - \xi_0)^i.$$

We have successively proved the following steps in COQ.

- If the condition holds on Line 3, then $f^{(k)}(x)$ is never `Xnan` for $0 \leqslant k \leqslant n+1$ and $x \in \boldsymbol{I}$, and $f^{(n+1)}$ has a constant sign over $[\inf \boldsymbol{I}, \sup \boldsymbol{I}]$.
- We have $\forall \xi_0 \in \boldsymbol{I}$, $\forall x \in \boldsymbol{I}$, $R_n(f, \xi_0)'(x) = R_{n-1}(f', \xi_0)(x)$, and, by Theorem 1, there exists ξ' between ξ_0 and x such that $R_{n-1}(f', \xi_0)(x) = \frac{(f')^{(n)}(\xi')}{n!}(x - \xi_0)^n$ (the case where $n = 0$ is handled separately in the formalisation).
- Then we study the sign of the expression $R_n(f, \xi_0)'(x)$ to conclude that $R_n(f, \xi_0)$ is monotonous over $[\inf \boldsymbol{I}, \xi_0]$ as well as over $[\xi_0, \sup \boldsymbol{I}]$.
- Since $\inf \boldsymbol{I} \in \boldsymbol{a}$, the enclosure properties of $\boldsymbol{F}$ and $\boldsymbol{P}$ proves that $R_n(f, \xi_0)(\inf \boldsymbol{I})$ belongs to $\boldsymbol{F}(\boldsymbol{a}) - \boldsymbol{P}(\boldsymbol{a} - \boldsymbol{x_0}) = \boldsymbol{\Delta_a}$, which is a sub-interval of $\boldsymbol{\Delta}$. Similarly, we prove that we have $R_n(f, \xi_0)(\sup \boldsymbol{I}) \in \boldsymbol{\Delta}$ and $R_n(f, \xi_0)(\xi_0) \in \boldsymbol{\Delta}$.
- Finally, for any $x \in \boldsymbol{I}$, we have $x \in [\inf \boldsymbol{I}, \xi_0]$ or $x \in [\xi_0, \sup \boldsymbol{I}]$, so combining the monotonic-

ity of $R_n(f, \xi_0)$ and the convexity of $\mathbf{\Delta}$ gives $R_n(f, \xi_0)(x) \in \mathbf{\Delta}$.

4) Specialising generic proofs to handle usual functions: In order to add a function to our framework, the user has to provide:

- the recurrence relation between the Taylor coefficients of the function in order to be able to compute the coefficients,
- the interval evaluator for the function in order to be able to provide an initial value,
- the definition of the abstract function and properties of the function and of its derivatives to be able to prove correct the computed Taylor model.

We detail here the example of the exponential function. Its Taylor coefficients $(c_n)_{n\in\mathbb{N}}$ satisfy $c_n = \frac{c_{n-1}}{n}$. The corresponding COQ code is thus

```
Definition exp_rec (c : T)(n : ℕ) := tdiv c (tnat n).
```

where `tdiv` represents division and `tnat` an injection of integers into the type of coefficients. Note that all these functions are defined for an abstract type of numbers `T`. The user will then be able to instantiate the recurrence with real numbers, or floating-point numbers, or intervals, according to his or her needs.

The generic Taylor polynomial for the exponential is given by the `trec1` function for first-order recurrences, which was presented in Section III-B2:

```
Definition T_exp y0 n := trec1 exp_rec (texp y0) n.
```

Notice that one of the arguments passed to `trec1` is `(texp y0)`, the value of the exponential at `y0`. This is where the evaluator for the function is used to "initialise" the computation.

In order to provide an n^{th}-order Taylor model for exp over interval $\boldsymbol{I}$, an n^{th}-order Taylor polynomial around $\boldsymbol{x_0}$ is combined with an enclosure of the Taylor–Lagrange remainder as computed by Algorithm 1 (named `Ztech` in the code). The Taylor model that is obtained is an instance of the `rpa` structure described in Section II:

```
Definition TM_exp x0 I n : rpa :=
  let P := (T_exp x0 n) in
  RPA P (Ztech T_exp P texp x0 I n).
```

Note that we have omitted a precision argument for readability in all the above COQ definitions. This argument allows the user to set the desired precision of each computation.

As regards to proofs, the generic theorems of correctness apply to Taylor models with interval coefficients. The correctness is proved with respect to functions from the Reals standard library. The properties required for the correctness concern:

- the compatibility of the interval function with the real function,
- the appropriate behaviour on the NaN value,
- the compatibility between the function and the recurrence used to compute the Taylor coefficients.

In order to ease the verification of this latter property in future versions of the library, we expect to benefit from ongoing formalisation efforts in the Coquelicot[1] project that is coordinated by the Toccata team of Inria. The long-term goal of formalizing the DDMF,[2] undertook by the SpecFun team of Inria, may also lead to a more generic approach for defining the functions that we want to approximate.

Once all these properties are provided, we can prove the following correctness lemma of our algorithm by simply applying the generic theorem.

```
Lemma TM_exp_correct :
  forall x0 I n, x0 ⊂ I -> x0 ≠ ∅ ->
  i_validTM x0 I (TM_exp x0 I n) Xexp.
```

This result states that the computed Taylor model `TM_exp` is a valid model (in the sense of Definition 1) of `Xexp`, which is the exponential function defined in the standard library of COQ, lifted from $\mathbb{R}$ to $\overline{\mathbb{R}}$.

Available functions and current restrictions: The current version of our CoqApprox library provides fully formally proved Taylor model algorithms for the following functions: constants, identity, inv, sqrt, inv ∘ sqrt, exp, sin, and cos.

All of these functions are D-finite functions, so they fit in our framework perfectly. The implementation as well as the correctness proof of each of these functions are just instantiations of the generic algorithms and proofs. For these functions, all the necessary ingredients were already available in the Coq.Interval library and the Reals library. However, other functions require more work in order to have an associated Taylor model.

For example, the logarithm function is available in the Reals library, but we are not able to compute a Taylor model as the interval evaluator is not yet implemented in Coq.Interval.

Another special case is the tangent function. It is not a D-finite function. Its ordinary differential equation is not linear. So we cannot describe the Taylor coefficients of the tangent using a linear recurrence relation. However, we may describe them with a non-linear recurrence relation. Since our generic framework and in particular Algorithm 1 is not restricted to Taylor polynomials given by linear recurrence relations, we could implement tangent in this way. The other option is to deal with $\tan(x)$ as $\frac{\sin x}{\cos x}$, which can be handled through the Taylor models algorithms for composite functions, but yields a much slower algorithm.

IV. Taylor Models for Composite Functions

In order to provide Taylor models for the addition, multiplication, composition and division of two functions we do not use a Taylor expansion (as for the base functions) but we introduce an arithmetic on Taylor models by

[1]URL: http://coquelicot.saclay.inria.fr/

[2]The Dynamic Dictionary of Mathematical Functions, available at the URL http://ddmf.msr-inria.inria.fr/

providing a specific algorithm for each of these operations. The main reason for this choice is to ensure tighter bounds on the error estimated for the Taylor model. The algorithms and proofs we use for these operations on Taylor models closely follow those described in [1].

A. Mathematical setup

Consider two Taylor models $(\boldsymbol{P}_1, \boldsymbol{\Delta}_1)$ and $(\boldsymbol{P}_2, \boldsymbol{\Delta}_2)$ of n^{th}-order, approximating f_1 and f_2 over the interval $\boldsymbol{I}$. We define the following operations on these Taylor models and obtain in each case an n^{th}-order Taylor model.

Addition: $(\boldsymbol{P}_1, \boldsymbol{\Delta}_1) \oplus (\boldsymbol{P}_2, \boldsymbol{\Delta}_2) := (\boldsymbol{P}_1 + \boldsymbol{P}_2, \boldsymbol{\Delta}_1 + \boldsymbol{\Delta}_2)$;

Multiplication: $(\boldsymbol{P}_1, \boldsymbol{\Delta}_1) \odot (\boldsymbol{P}_2, \boldsymbol{\Delta}_2) := (\boldsymbol{P}, \boldsymbol{\Delta})$ where $\boldsymbol{P} := (\boldsymbol{P}_1 \cdot \boldsymbol{P}_2)_{\leqslant n}$ and $\boldsymbol{\Delta} := \boldsymbol{\Delta}_1 \cdot \boldsymbol{\Delta}_2 + \mathrm{eval}(\boldsymbol{P}_1) \cdot \boldsymbol{\Delta}_2 + \boldsymbol{\Delta}_1 \cdot \mathrm{eval}(\boldsymbol{P}_2) + \mathrm{eval}\big((\boldsymbol{P}_1 \cdot \boldsymbol{P}_2)_{>n}\big)$.

In the above definition $(\boldsymbol{Q})_{\leqslant n}$ denotes the polynomial (with interval coefficients) containing all the monomials of $\boldsymbol{Q}$ up to degree n, and $(\boldsymbol{Q})_{>n}$ those of higher degree; $\mathrm{eval}(\boldsymbol{Q})$ is the evaluation of polynomial $\boldsymbol{Q}$ over interval $\boldsymbol{I}$.

As regards the *composition* of two functions $f_1 \circ f_2$, the Taylor model algorithm is essentially the evaluation of the polynomial corresponding to f_1 in the Taylor model of f_2. This is accomplished by relying on the addition and multiplication of Taylor models. The error is composed of the error that results from this evaluation and the error of the Taylor model of f_1.

We also define the inverse of a Taylor model and the division of two Taylor models. We compute a Taylor model for $\frac{1}{f(x)}$ by using the Taylor model algorithm for the composition $\mathrm{inv} \circ f$. We compute a Taylor model for $\frac{f(x)}{g(x)}$ as the multiplication of a Taylor model for $f(x)$ by a Taylor model for $\frac{1}{g(x)}$.

All the correctness theorems for addition, multiplication and composition of Taylor models (TMs) have the same form. For example, the correctness for the addition is stated as follows: if $(\boldsymbol{P}_1, \boldsymbol{\Delta}_1)$ and $(\boldsymbol{P}_2, \boldsymbol{\Delta}_2)$ are two valid TMs for functions f_1 and f_2 over $\boldsymbol{I}$, then the sum defined as above is a valid TM for $f_1 + f_2$ over $\boldsymbol{I}$.

B. Formal setup

The arithmetic operations on Taylor models are implemented using the algorithms summarised in the previous section. As for the basic functions, we have generic algorithms for these operations, that are then instantiated with the type of coefficients and the type of polynomials desired. Here is the COQ code for the definition of addition:

```
Definition TM_add (Mf Mg : rpa) : rpa :=
  RPA (Pol.tadd (approx Mf) (approx Mg))
      (I.add (error Mf) (error Mg)).
```

The formal proofs of correctness for addition, multiplication and composition closely follow the pen-and-paper proofs that can be found in [1, Chap. 2]. The main adjustments that we have to make concern the use of NaN values that can occur in our setting.

For example, some steps in the proof of correctness for the multiplication of Taylor models involve distributivity results about big operators. But for these steps we cannot directly reason over extended reals $(\overline{\mathbb{R}}, +, \cdot)$, because the neutral element of addition, namely `(Xreal 0)`, is not an absorbing element for multiplication, as pointed out in Section II. In this case, we can adjust the proof by passing through big operators over $\mathbb{R}$. However, in some situations we have to exclude the undefined values from the theorem. For example, the correctness of the composition of Taylor models can only be proved for polynomials whose size is greater than zero, i.e., they must have at least one coefficient. In practice, however, this is not a problem. We never generate such polynomials with no coefficients in our formalisation.

Finally as regards the correctness proof for division, it is immediately given by applying the generic theorems asserting the correctness of Taylor models for multiplication, composition, and the basic function $\mathrm{inv} : x \mapsto \frac{1}{x}$.

V. Running the Library

Our library is freely available at the following URL: http://tamadi.gforge.inria.fr/CoqApprox/

The following table compares the status of the current library with respect to what was presented at NFM 2012 [4].

	# Lines of code		# Proved
	Specs	Proofs	lemmas
CoqApprox v1	794	570	47
CoqApprox v2	2707	4564	452

These figures illustrate the amount of work that was needed in order to get a fully-proved library from our initial prototype.

The current version of our library is compatible with both the native-coq branch[3] of COQ [11] and the current version of COQ (version 8.4pl2). For evaluating the performances of our library, we rely on the native-coq version, which features native machine integers and a compilation to fast and native OCaml code.

A state-of-the-art implementation of univariate Taylor Models written in C is available in the Sollya tool[4] [12]. Roughly speaking, Sollya represents polynomials as arrays of interval coefficients with multiple-precision floating-point bounds and relies on several C libraries such as GMP and MPFR.

Table II gives the timing and the quality of the approximation obtained for a selection of functions. A laptop based on an Intel Core i7 processor clocked at 2.60GHz running the 2013-03-15 version of native-coq with OCaml 3.12.1, and Sollya 4.0 has been used for these tests.

In particular, the computations that are performed with Sollya rely on the `taylorform()` function. As regards the computations performed with COQ, we use the instantiation of our hierarchy with polynomials as lists (with

[3]URL: https://github.com/maximedenes/native-coq
[4]URL: http://sollya.gforge.inria.fr/

linear access time) gathering interval coefficients with floating-point bounds, themselves built upon the machine-efficient big integers that are provided by the `BigZ` library of COQ.

Table II. BENCHMARKS FOR OUR LIBRARY ON TAYLOR MODELS

	Execution time			Approximation error		
	COQ	Sollya	ratio	naive COQ	COQ	Sollya
$f(x)=e^x$ $\boldsymbol{I}=[2,4]$ order=80 prec=500	0.174s	**0.092s**	1.9	$1.52{\cdot}2^{-396}$	$1.14{\cdot}2^{-397}$	$1.14{\cdot}2^{-397}$
$f(x)=\sin x$ $\boldsymbol{I}=[-1,1]$ order=80 prec=500	0.146s	**0.092s**	1.6	$1.79{\cdot}2^{-402}$	$1.79{\cdot}2^{-402}$	$1.79{\cdot}2^{-402}$
$f(x)=\frac{1}{x}$ $\boldsymbol{I}=[1,3]$ order=100 prec=125	**0.022s**	0.165s	0.13	$1{\cdot}2^{0}$	$1{\cdot}2^{-101}$	$1{\cdot}2^{-101}$
$f(x)=\sqrt{x}$ $\boldsymbol{I}=[1,3]$ order=100 prec=125	**0.037s**	0.169s	0.22	$1.98{\cdot}2^{-12}$	$1.60{\cdot}2^{-112}$	$1.60{\cdot}2^{-112}$
$f(x)=\frac{1}{\sqrt{x}}$ (as a basic function) $\boldsymbol{I}=[1,3]$ order=100 prec=125	**0.029s**	0.424s	0.068	$1.80{\cdot}2^{-5}$	$1.27{\cdot}2^{-105}$	$1.27{\cdot}2^{-105}$
$f(x)=e^x\sin x$ $\boldsymbol{I}=[-\frac{3}{2},\frac{3}{2}]$ order=50 prec=500	0.497s	**0.048s**	10	$1.94{\cdot}2^{-166}$	$1.94{\cdot}2^{-166}$	$1.94{\cdot}2^{-166}$
$f(x)=e^x\sin x$ $\boldsymbol{I}=[-\frac{3}{2},\frac{3}{2}]$ order=100 prec=500	1.010s	**0.306s**	3.3	$1.63{\cdot}2^{-423}$	$1.63{\cdot}2^{-423}$	$1.63{\cdot}2^{-423}$
$f(x)=e^{1/\cos x}$ $\boldsymbol{I}=[0,1]$ order=50 prec=100	6.378s	**0.095s**	67	$1.46{\cdot}2^{-23}$	$1.45{\cdot}2^{-41}$	$1.45{\cdot}2^{-41}$
$f(x)=e^{1/\cos x}$ $\boldsymbol{I}=[0,1]$ order=100 prec=100	52.92s	**0.653**	81	$1.97{\cdot}2^{-49}$	$1.99{\cdot}2^{-89}$	$\mathbf{1.98{\cdot}2^{-89}}$
$f(x)=\frac{\sin x}{\cos x}$ $\boldsymbol{I}=[-1,1]$ order=50 prec=100	1.228s	**0.083s**	15	$1.06{\cdot}2^{14}$	$1.66{\cdot}2^{-32}$	$\mathbf{1.10{\cdot}2^{-52}}$
$f(x)=\frac{\sin x}{\cos x}$ $\boldsymbol{I}=[-1,1]$ order=100 prec=100	11.15s	**0.570s**	20	$1.45{\cdot}2^{26}$	$1.12{\cdot}2^{-64}$	$\mathbf{1.82{\cdot}2^{-96}}$
$f(x)=\frac{1}{\sqrt{x}}$ (as a composite function) $\boldsymbol{I}=[1,3]$ order=100 prec=125	37.683s	**0.424s**	89	$1.98{\cdot}2^{-12}$	$1.27{\cdot}2^{-105}$	$1.27{\cdot}2^{-105}$

The first column of Table II gives the function and the interval $\boldsymbol{I}$. In these experiments, we have chosen to develop the Taylor models at the middle of the interval, that is $\boldsymbol{x_0} := [c, c]$ where $c := \frac{1}{2}(\inf \boldsymbol{I} + \sup \boldsymbol{I})$. This column also gives the order of the TM, and the working precision for floating-point operations in radix 2.

The three subsequent columns give the timing for computing the respective Taylor model, and the ratio $\mathrm{Time}_{\mathrm{COQ}}/\mathrm{Time}_{\mathrm{Sollya}}$. We thus notice that these timings are of the same order of magnitude for all basic functions (the shortest ones being printed in bold). We also notice that on these examples ranging up to degree 100, the COQ timings for composite functions are only 3 to 89 times slower than the Sollya implementation, which is reasonable, given that the COQ implementation is being executed in a trusted environment with restricted computing power.

The last three columns give the magnitude of the error interval that takes into account both the method error and the rounding errors involved in the approximation. To sum up, it corresponds to the value $\max\{|\inf \boldsymbol{\Delta'}|, |\sup \boldsymbol{\Delta'}|\}$ rounded towards $+\infty$, where $\boldsymbol{\Delta'}$ has been computed to satisfy Definition 2. The column "naive COQ" corresponds to our former implementation of Taylor Models for basic functions that simply relies on a naive enclosure of the Taylor–Lagrange remainder, while the column "COQ" corresponds to the optimised implementation, based on Algorithm 1. For the sake of readability, all these bounds are written as a power of 2 multiplied by a decimal number between 1 and 2.

First, we can notice that the optimisation due to Algorithm 1 significantly improves the bounds for some basic functions such as the reciprocal function or the square root, as well as for the composite functions that involve division. Then, we notice that for each basic function, the error bounds computed by COQ, resp. Sollya, have exactly the same order of magnitude. As regards the composite functions, the same remark applies, except that Sollya's bounds for $\tan x = \frac{\sin x}{\sin x}$ are tighter than those of COQ. We expect that this latter difference is explained by a difference in the implemented algorithms, since our tangent is seen as a composite function. Finally, the availability of a Taylor model algorithm for $x \mapsto \frac{1}{\sqrt{x}}$ seen as a basic function allows one to check, as expected, that this algorithm is much faster than considering $x \mapsto \frac{1}{\sqrt{x}}$ as a composite function. Yet as regards the provided error bounds, we do not notice any difference for this particular function.

VI. RELATED WORKS

To our knowledge, the first formalisation of multivariate Taylor models is described in [13]. It relies on exact real arithmetic, but no formal proof is available. A formally-proved implementation of univariate Taylor models in the PVS proof assistant is presented in [14]. However, the Taylor models are defined in an ad-hoc way for few functions and with 6 as maximal degree. Another formally-proved implementation is described in [15]. The coefficients are axiomatised floating-point numbers so the formalisation is not directly executable. Finally, more recently, a formally-proved library to solve nonlinear inequalities using Taylor approximations in HOL LIGHT has been proposed in [16]. This last work focuses on small-degree, multivariate polynomials to automatically solve inequations that occur in the proof of the Kepler conjecture. In the application we envision, i.e. certifying polynomial approximations, we are mostly interested in high-degree (up to 90 in some cases) univariate Taylor models. The computing power provided by COQ is then crucial.

VII. Conclusion and Future Work

Our initial interest in developing a certified library that manipulates Taylor models comes from a long-term project to validate the worst cases for correct rounding of elementary functions. A first step in this validation is to be able to certify the quality of an approximation. Given a function f, an approximation P, a bound ϵ and an interval $\boldsymbol{I}$, we would like to derive automatically within COQ that $|f(x) - P(x)| < \epsilon$ for $x \in \boldsymbol{I}$. What we have achieved with our library of Taylor models is to be able to prove automatically something like $|f(x) - TM_f(x)| \leqslant \epsilon_1$ for $x \in \boldsymbol{I}$. For this work, we have followed a "prototype and prove" strategy. In the first stage, we have used the COQ system as a mere programming language to implement our library. This is the prototyping phase that was described in [4]. We ended up with a library that was worth proving. The second stage (and the most time-consuming one) was to formally verify this library. The resulting library has the following characteristics:

- It is generic. Thanks to our design based on modules, we can easily and independently change various aspects of the library (representation of the polynomials, representation of coefficients, representation of intervals). Also, we have taken a great care to develop proofs that are as generic as possible.
- It is efficient enough. We have made a number of tests that seems to indicate that we are only one order of magnitude slower than the implementation of Sollya.
- It has been formally verified. We have correctness proofs for the operations on Taylor models (namely $+, \times, \circ, \div$) as well as for the basic functions $x \mapsto \frac{1}{x}$, $\sqrt{\cdot}$, $\frac{1}{\sqrt{\cdot}}$, exp, sin and cos.
- It returns sharp bounds. In particular, we have performed a number of tests to demonstrate the impact of an optimisation of our algorithms (based on a result from Roland Zumkeller's thesis).

The library could still be further improved. Some interesting basic functions are still missing like arctan, tan and log. Also, Karatsuba algorithm could be implemented in order to multiply two Taylor models more efficiently. Nevertheless, our next priority is to complete our validation by providing an automatic tool within COQ to bound a polynomial on an interval. Different standard techniques exist and have already been applied in a formal setting, such as sum of squares [17] or Bernstein polynomials [18]. Combined with our Taylor models which prove $|f(x) - TM_f(x)| \leqslant \epsilon_1$ for $x \in \boldsymbol{I}$, we could also derive that $|TM_f(x) - P(x)| < \epsilon_2$. Then, a simple application of the triangle inequality would lead to the expected validation $|f(x) - P(x)| < \epsilon$ for a judicious choice of ϵ_1 and ϵ_2.

Acknowledgment

The formalisation presented here would not have been possible without the help of Nicolas Brisebarre, Mioara Joldeş, and Jean-Michel Muller. We deeply benefit from their expertise on Taylor models so that the whole formal development, including the machine-checked proofs of the correctness of their algorithms, could successfully be carried out.

References

[1] M. Joldeş, "Rigourous Polynomial Approximations and Applications," Ph.D. dissertation, ENS Lyon, 2011. [Online]. Available: http://tel.archives-ouvertes.fr/tel-00657843/en/

[2] K. Makino and M. Berz, "Taylor Models and Other Validated Functional Inclusion Methods," *International Journal of Pure and Applied Mathematics*, vol. 4, no. 4, pp. 379–456, 2003.

[3] K. Makino, "Rigorous Analysis of Nonlinear Motion in Particle Accelerators," Ph.D. dissertation, Michigan State University, East Lansing, Michigan, USA, 1998.

[4] N. Brisebarre, M. Joldeş, É. Martin-Dorel, M. Mayero, J.-M. Muller, I. Paşca, L. Rideau, and L. Théry, "Rigorous Polynomial Approximation Using Taylor Models in Coq," in *NFM'12*, ser. LNCS, vol. 7226. Springer, 2012, pp. 85–99.

[5] M. Mayero, "Formalisation et automatisation de preuves en analyses réelle et numérique," Ph.D. dissertation, Paris VI University, France, 2001.

[6] G. Melquiond, "Floating-point arithmetic in the Coq system," *Information and Computation*, vol. 216, pp. 14–23, 2012.

[7] G. Gonthier, A. Mahboubi, and E. Tassi, "A Small Scale Reflection Extension for the Coq system," INRIA, Research Report RR-6455, 2008. [Online]. Available: http://hal.inria.fr/inria-00258384/en/

[8] É. Martin-Dorel, "Contributions to the Formal Verification of Arithmetic Algorithms," Ph.D. dissertation, ENS Lyon, Sep. 2012. [Online]. Available: http://tel.archives-ouvertes.fr/tel-00745553/en/

[9] S. Chevillard, J. Harrison, M. Joldeş, and C. Lauter, "Efficient and accurate computation of upper bounds of approximation errors," *Theoretical Computer Science*, vol. 16, no. 412, pp. 1523–1543, 2011.

[10] R. Zumkeller, "Global Optimization in Type Theory," Ph.D. dissertation, École polytechnique, France, 2008. [Online]. Available: http://alacave.net/~roland/FormalGlobalOpt.pdf

[11] M. Boespflug, M. Dénès, and B. Grégoire, "Full Reduction at Full Throttle," in *CPP*, ser. LNCS, vol. 7086. Springer, 2011, pp. 362–377.

[12] S. Chevillard, M. Joldeş, and C. Lauter, "Sollya: An Environment for the Development of Numerical Codes," in *Mathematical Software - ICMS 2010*, ser. LNCS, vol. 6327. Springer, September 2010, pp. 28–31.

[13] R. Zumkeller, "Formal Global Optimisation with Taylor Models," in *IJCAR*, ser. LNCS, U. Furbach and N. Shankar, Eds., vol. 4130. Springer, 2006, pp. 408–422.

[14] F. Cháves, "Utilisation et certification de l'arithmétique d'intervalles dans un assistant de preuves," Ph.D. dissertation, ENS Lyon, Sep. 2007. [Online]. Available: http://tel.archives-ouvertes.fr/tel-00177109/en/

[15] P. Collins, M. Niqui, and N. Revol, "A Validated Real Function Calculus," *Mathematics in Computer Science*, vol. 5, no. 4, pp. 437–467, 2011.

[16] A. Solovyev and T. C. Hales, "Formal Verification of Nonlinear Inequalities with Taylor Interval Approximations," in *NFM'13*, ser. LNCS, vol. 7871. Springer, 2013, pp. 383–397.

[17] J. Harrison, "Verifying Nonlinear Real Formulas Via Sums of Squares," in *TPHOLs*, ser. LNCS, vol. 4732, 2007, pp. 102–118.

[18] C. Muñoz and A. Narkawicz, "Formalization of Bernstein Polynomials and Applications to Global Optimization," *Journal of Automated Reasoning*, pp. 1–46, 2012.