Rigorous Polynomial Approximation Using Taylor Models in Coq

Érik Martin-Dorel

ENS de Lyon, LIP lab., AriC team
46 allée d'Italie, 69364 LYON CEDEX 07, France
erik.martin-dorel@ens-lyon.org
http://perso.ens-lyon.fr/erik.martin-dorel/

joint work with Nicolas Brisebarre, Mioara Joldeș, Micaela Mayero, Jean-Michel Muller, Ioana Pașca, Laurence Rideau & Laurent Théry

NFM 2012
Norfolk, Virginia, USA
3-5 April 2012

Outline

1. Introduction and Motivations
2. Presentation of Taylor Models
3. Formalization of Taylor Models in Coq
4. Benchmarks and Timings
5. Conclusion and Future Work

Outline

1. Introduction and Motivations
2. Presentation of Taylor Models
3. Formalization of Taylor Models in Coq
4. Benchmarks and Timings
5. Conclusion and Future Work

Context and Motivations

- Polynomial approximation
  - A common way to represent real functions on machines
  - Only solution for platforms where only $+$, $-$, $\times$ are available
  - Used by most computer algebra systems
- Bounds for approximation errors
  - Not always available or guaranteed to be accurate in numerical software
  - Yet they may be crucial to ensure the reliability of systems
  - A key part of the TaMaDi project formalization effort for computing the "worst-case accuracy" for all elementary functions

Rigorous Polynomial Approximation

In the setting of rigorous polynomial approximation (RPA):
Approximate the function while **fully controlling the error**

- May use **floating-point arithmetic** as support for efficient computation
- Systematically compute **interval enclosures** instead of mere approximations

Goal

Implement rigorous polynomial approximation in a formal setting, implying:

- Use techniques from **symbolic/numeric computation**: amenable to formal methods
- **Genericity**: implementation extensible and applicable to a large class of problems
- **Efficiency**: evaluate the computational capabilities of the formal proof assistant before starting proving anything
- **Formal verification**: ensure the provided error bound is (tight and) not underestimated

Floating-Point (FP) Arithmetic

Reals numbers can be approximated in machines by
floating-point numbers, which are rational numbers of the form

$$x = M \times 2^{e-p+1} \quad \text{with} \quad 2^{p-1} \leqslant |M| < 2^p$$

- the integer $p \geqslant 1$ is the precision of the considered FP format
- the integer M is the integral significand of x
- the integer e is the exponent of x

Interval Arithmetic (IA)

- Interval = pair of floating-point (FP) numbers
- E.g., $\pi \in [3.1415, 3.1416]$
- Operations on intervals (satisfying enclosure property), e.g.: $[2,4] - [0,1] = [1,4]$ (we have $\forall x \in [2,4], \forall y \in [0,1], x - y \in [1,4]$)
- Tool for bounding the range of functions
- A naive use of IA cannot be successful
- Dependency problem: for $F(x) := x - x$ and $\boldsymbol{X} = [1,5]$, the IA evaluation gives $F(\boldsymbol{X}) = [-4,4]$ while the image of $\boldsymbol{X}$ by F is $[0,0]$
- Moreover, IA is not directly applicable to bound the approximation error $e := p - f$ given that the values of f and p will be very near

Outline

1 Introduction and Motivations

2 Presentation of Taylor Models

3 Formalization of Taylor Models in Coq

4 Benchmarks and Timings

5 Conclusion and Future Work

Taylor Models

Definition

An order-n Taylor Model (TM) for a function $f : D \subset \mathbb{R} \to \mathbb{R}$ over $\boldsymbol{I}$ is a pair $(T, \boldsymbol{\Delta})$ where T is a degree-n polynomial and $\boldsymbol{\Delta}$ is an interval, such that $\forall x \in \boldsymbol{I},\ f(x) - T(x) \in \boldsymbol{\Delta}$.

The polynomial T is typically a Taylor expansion of f at $x_0 \in \boldsymbol{I}$ and the **interval remainder** $\boldsymbol{\Delta}$ provides an enclosure of the approximation error

Our Approach

As regards T: small **interval coefficients** with floating-point bounds

$\Longrightarrow$ rounding errors are directly handled by the interval arithmetic

$\Longrightarrow$ the true coefficients of the Taylor expansion lie inside these intervals

Taylor-Lagrange Remainder

Theorem (Taylor-Lagrange)

If f is $n+1$ times derivable on $\boldsymbol{I}$, then $\forall x \in \boldsymbol{I}$, $\exists \xi$ between x_0 and x s.t.:

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

Outline

For T: Compute interval enclosures of $\frac{f^{(i)}(x_0)}{i!}$, $i = 0, \ldots, n$.

For $\boldsymbol{\Delta}$: Compute enclosure of $\Delta(x, \xi)$:

Compute enclosure of $\frac{f^{(n+1)}(\xi)}{(n+1)!}$ and deduce $\boldsymbol{\Delta} := \frac{f^{(n+1)}(\boldsymbol{I})}{(n+1)!}(\boldsymbol{I} - x_0)^{n+1}$

Taylor-Lagrange Remainder

Theorem (Taylor-Lagrange)

If f is $n+1$ times derivable on $\boldsymbol{I}$, then $\forall x \in \boldsymbol{I}$, $\exists \xi$ between x_0 and x s.t.:

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

Outline

For T: Compute interval enclosures of $\frac{f^{(i)}(x_0)}{i!}$, $i = 0, \ldots, n$.

For $\boldsymbol{\Delta}$: Compute enclosure of $\Delta(x, \xi)$:

Compute enclosure of $\frac{f^{(n+1)}(\xi)}{(n+1)!}$ and deduce $\boldsymbol{\Delta} := \frac{f^{(n+1)}(\boldsymbol{I})}{(n+1)!}(\boldsymbol{I} - x_0)^{n+1}$

Composite functions $\Rightarrow$ enclosure for $\boldsymbol{\Delta}$ can be **largely overestimated**

Methodology of Taylor Models

Define arithmetic operations on Taylor Models:

- $\texttt{TM}_{\texttt{add}}$, $\texttt{TM}_{\texttt{mul}}$, $\texttt{TM}_{\texttt{comp}}$, and $\texttt{TM}_{\texttt{div}}$
- E.g., $\texttt{TM}_{\texttt{add}} : \Big((P_1, \mathbf{\Delta_1}),\ (P_2, \mathbf{\Delta_2})\Big) \mapsto (P_1 + P_2, \mathbf{\Delta_1} + \mathbf{\Delta_2})$.

A two-fold approach:

- **Apply these operations recursively** on the structure of the function
- **Use Taylor-Lagrange remainder for atoms** (i.e., for base functions)

Methodology of Taylor Models

Define arithmetic operations on Taylor Models:

- $\texttt{TM}_{\texttt{add}}$, $\texttt{TM}_{\texttt{mul}}$, $\texttt{TM}_{\texttt{comp}}$, and $\texttt{TM}_{\texttt{div}}$
- E.g., $\texttt{TM}_{\texttt{add}} : \Big((P_1, \mathbf{\Delta_1}),\ (P_2, \mathbf{\Delta_2})\Big) \mapsto (P_1 + P_2, \mathbf{\Delta_1} + \mathbf{\Delta_2})$.

A two-fold approach:

- **Apply these operations recursively** on the structure of the function
- **Use Taylor-Lagrange remainder for atoms** (i.e., for base functions)

$\Rightarrow$ Need to consider a relevant class for base functions, so that:

- We can easily compute their successive derivatives
- The interval remainder computed for these atoms is thin enough

D-finite Functions

Definition

A D-finite function is a solution of a homogeneous linear ordinary differential equation with polynomial coefficients:

$$a_r(x)y^{(r)}(x) + \cdots + a_1(x)y'(x) + a_0(x)y(x) = 0, \qquad \text{for } a_k \in \mathbb{K}[X]$$

Example (exp)

The function $y = \exp$ is fully determined by $\{y' - y = 0,\ y(0) = 1\}$

Most common functions are D-finite ($\sin$, $\cos$, $\arcsin$, $\arccos$, $\sinh$, $\cosh$, $\operatorname{arcsinh}$, $\operatorname{arccosh}$, Si, Ci, Shi, Chi, $\arctan$, $\exp$, ln, Ei, erf, Ai, Bi, ...). $\tan$ is not.

Taylor Series of D-finite Functions

Theorem

A function represented by a Taylor series $f(x) = \sum_{n=0}^{\infty} u_n \, (x - x_0)^n$ *is* D*-finite if and only if the sequence* (u_n) *of its Taylor coefficients satisfies a linear recurrence with polynomial coefficients.*

recurrence relation
initial conditions $\Big\}$ $\Rightarrow$ **fast numerical computation** of Taylor coefficients

Example ($\exp$)

Taylor series: $\exp(x) = \sum_{n=0}^{\infty} \frac{\exp(x_0)}{n!} \, (x - x_0)^n$

Recurrence: $\forall n \in \mathbb{N}, \; u_{n+1} = \frac{u_n}{n+1}$ Initial condition: $u_0 = \exp(x_0)$

Outline

1 Introduction and Motivations

2 Presentation of Taylor Models

3 Formalization of Taylor Models in Coq

4 Benchmarks and Timings

5 Conclusion and Future Work

Tools and Libraries Involved in our Formalization

We are using:

- The Coq/SSReflect proof assistant: an interactive theorem prover combining a higher-order logic and a richly-typed functional programming language;
- The CoqInterval library for Multiple-Precision Floating-Point Interval Arithmetic in Coq, based on:
- The Flocq library for Multiple-Precision Floating-Point Arithmetic, itself based on:
- The Reals library from the Coq Standard Library, which consists of a classical axiomatization of the real numbers

Focus on Genericity

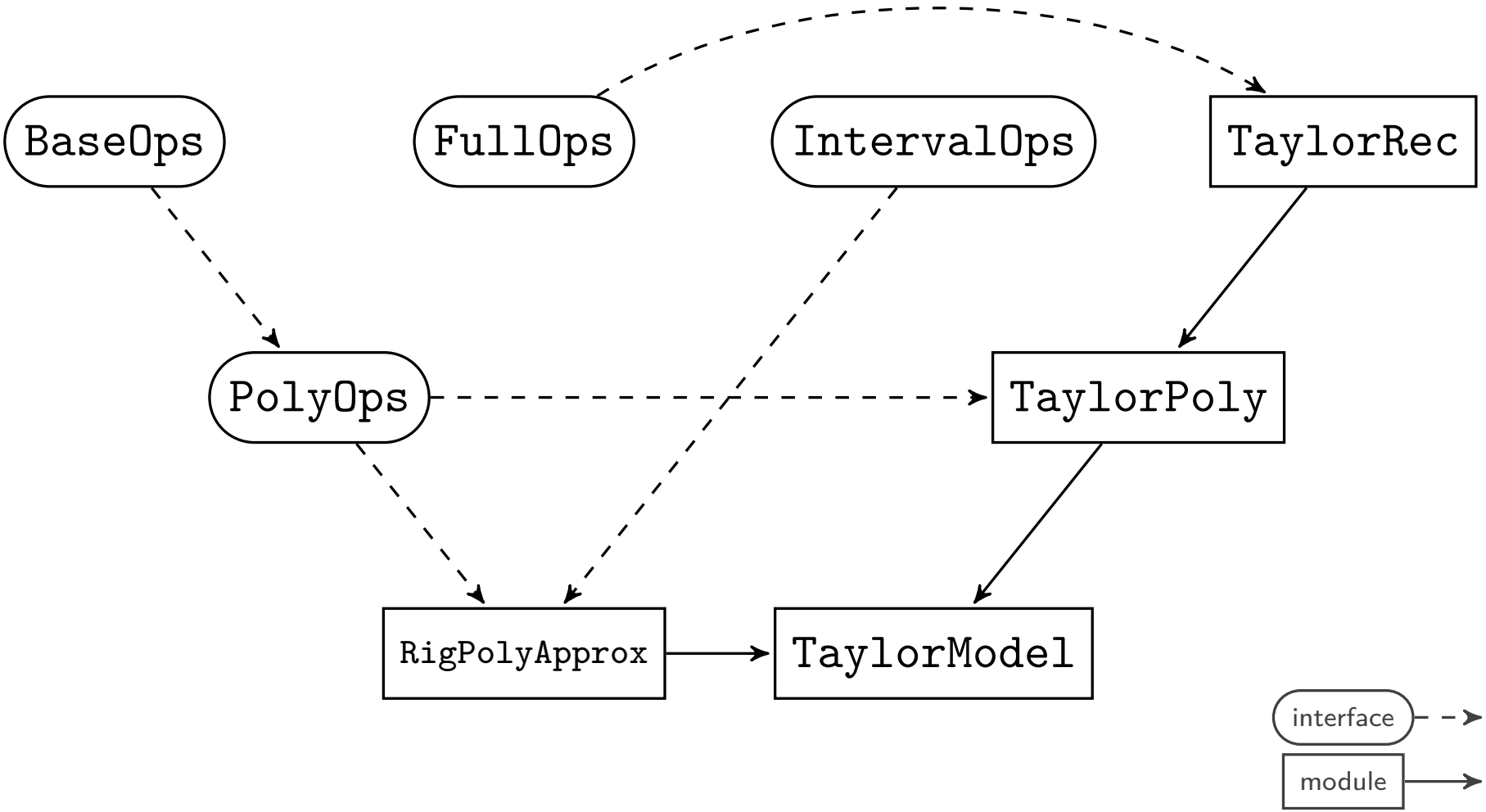
- Aim: being as **generic** as possible in our Coq formalization

- TM: instance of Rigorous Polynomial Approximation (RPA)
- RPA: $(P, \mathbf{\Delta})$ where P is not necessarily a Taylor polynomial
- Genericity for the type of P, the implementation of such polynomials, as well as the type of $\mathbf{\Delta}$

Choice of a Modularization Mechanism

- Coq provides three mechanisms for modularization:
  - Type Classes
  - Structures
  - Modules
- Like the CoqInterval library, our formalization uses Modules,
  - less flexible than Type Classes or Structures (first-class), but with a
  - better computational behavior: Modules instantiations are performed statically $\Rightarrow$ the executed code is more compact
  - ability to switch implementations

A Generic Implementation of TMs: Modular Hierarchy



Example of Use

Definition (TM for exp)

```
Definition exp_rec (n : nat) (u : T) := tdiv u (tnat n). (*∈TaylorRec*)
Definition T_exp n u := trec1 exp_rec (texp u) n. (*∈TaylorPoly*)
Definition TM_exp n X X0 := RPA (T_exp n X0)(Trem T_exp n X X0). (*∈TaylorModel*)
```

Example (TM for exp on $[1/2, 1]$)

```
(* Library/modules preliminary invocations & global precision setting omitted *)
Let a := Float 1 (-1). (* =1/2 *)
Let b := Float 2 (-1). (* =1 *)
Let X := Ibnd a b.
Let c := I.midpoint (Ibnd a b). (* =(a+b)/2 *)
Let X0 := Ibnd c c.
Let deg := 10%nat.
Let tm := TM_exp deg X X0.
Eval native_compute in (approx tm).
Eval native_compute in (error tm).
```

Outline

1 Introduction and Motivations

2 Presentation of Taylor Models

3 Formalization of Taylor Models in Coq

4 Benchmarks and Timings

5 Conclusion and Future Work

Quick Presentation of the Sollya Tool

Sollya

- Written in C
- Based on the MPFI library (Multiple-Precision FP IA)
- Contains an implementation of Taylor Models
- In an imperative-programming framework
- Polynomials as arrays of coefficients
- Not formally proved

CoqApprox

- Formalized in Coq
- Based on the CoqInterval library
- Implements Taylor Models using a similar algorithm
- In a functional-programming framework
- Polynomials as lists of coefficients (linear access time)
- Formal verification in progress

Some Benchmarks for Base Functions

	Timing		Approximation error	
	Coq	Sollya	Coq	Sollya
arctan prec=120, deg=8 $\boldsymbol{I}=[1,2]$ split in 256	11.45s	1.03s	7.43×10^{-29}	2.93×10^{-29}
exp prec=600, deg=40 $\boldsymbol{I}=[\ln 2, 1]$ split in 256	38.10s	16.39s	6.23×10^{-182}	6.22×10^{-182}

- When the interval $\boldsymbol{I}$ has been split into sub-intervals (of equal length), the timings are for the total duration of the computations while the approximation error is for the last subinterval.

Some Benchmarks for Composite Functions

	Timing		Approximation error	
	Coq	Sollya	Coq	Sollya
exp × sin prec=200, deg=10 $\boldsymbol{I}=[1/2, 1]$ split in 2048	1m22s	12.05s	6.92×10^{-50}	6.10×10^{-50}
exp ∘ sin prec=200, deg=10 $\boldsymbol{I}=[1/2, 1]$ split in 2048	3m24s	12.19s	4.90×10^{-47}	4.92×10^{-47}

- When the interval $\boldsymbol{I}$ has been split into sub-intervals (of equal length), the timings are for the total duration of the computations while the approximation error is for the last subinterval.

Outline

1 Introduction and Motivations

2 Presentation of Taylor Models

3 Formalization of Taylor Models in Coq

4 Benchmarks and Timings

5 Conclusion and Future Work

Towards Fully Formally Verified Polynomial Approximation

We can formalize the predicate saying that a TM is valid (i.e., whose error is not underestimated) using the following definition:

Definition

Let $f : \boldsymbol{I} \to \mathbb{R}$ be a function, $\boldsymbol{x_0}$ be a small interval around an expansion point x_0. Let T be a polynomial with interval coefficients $\boldsymbol{a_0}, \ldots, \boldsymbol{a_n}$ and $\boldsymbol{\Delta}$ an interval. We say that $(T, \boldsymbol{\Delta})$ is a Taylor model of f at $\boldsymbol{x_0}$ on $\boldsymbol{I}$ when

$$\begin{cases} \boldsymbol{x_0} \subseteq \boldsymbol{I}, \\ 0 \in \boldsymbol{\Delta}, \\ \forall \xi_0 \in \boldsymbol{x_0}, \exists \alpha_0 \in \boldsymbol{a_0}, \ldots, \alpha_n \in \boldsymbol{a_n}, \forall x \in \boldsymbol{I}, \quad f(x) - \sum_{i=0}^{n} \alpha_i \left(x - \xi_0\right)^i \in \boldsymbol{\Delta}. \end{cases}$$

Proofs are in Progress

Fun/Op	Available in CoqInterval	Implemented in CoqApprox	Proved in CoqApprox
cst	☒	☒	☒
id	☒	☒	☒
inv	☒	☒	☒
sqrt	☒	☒	☐
$\frac{1}{\sqrt{\cdot}}$	☒	☒	☐
exp	☒	☒	☒
sin	☒	☒	☒
cos	☒	☒	☒
arctan	☒	☒	☐
ln	☐	☒	☐
arcsin	☐	☒	☐
arccos	☐	☒	☐
$\mathtt{TM}_{\mathtt{add}}$		☒	☒
$\mathtt{TM}_{\mathtt{mul}}$		☒	☐
$\mathtt{TM}_{\mathtt{comp}}$		☒	☐
$\mathtt{TM}_{\mathtt{div}}$		☒	☐

Conclusion

- An implementation of Taylor Models inside the Coq proof assistant taking advantage of the CoqInterval library for Interval Arithmetic
- **Symbolic/numeric approach** based on D-finite recurrences
- **Genericity** achieved thanks to a heavy use of Coq modules
- **Efficiency**: timings in Coq are just one order of magnitude slower than a conventional implementation in C
- Implementation carried on in a framework suitable for formal proof

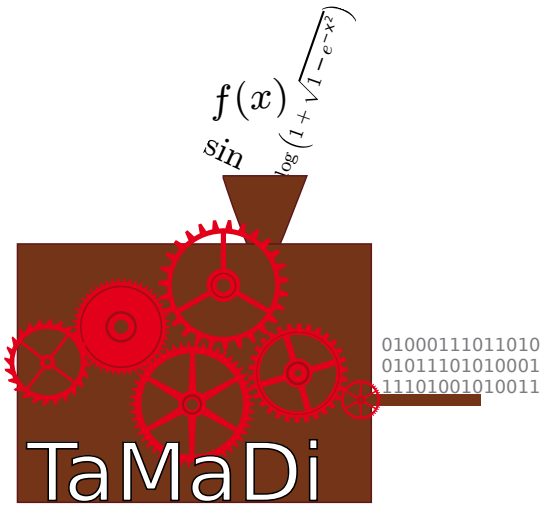
Future Work

- Add more functions
- Finish the proofs

- Use persistant arrays instead of lists?
- Optimize the multiplication algorithm for composite functions
- Consider "Chebyshev Models" which should provide tighter remainders

- Aim: Use our CoqApprox library in a full verification chain for TaMaDi

End of the Talk



Thank you for your attention!

The TaMaDi project homepage:
`https://tamadiwiki.ens-lyon.fr/`
`http://tamadi.gforge.inria.fr/CoqApprox/`