Contributions to the Formal Verification of Arithmetic Algorithms

Érik Martin-Dorel

PhD advisors: Micaela Mayero & Jean-Michel Muller

2012-09-26

École Normale Supérieure de Lyon, AriC team, Laboratoire de l'Informatique du Parallélisme

Context and Motivations

Context:

- The SLZ algorithm for solving (offline) the Table Maker's Dilemma

→ Very long calculations using sophisticated, optimized methods

→ Either output some numerical data whose completeness cannot be directly verified, or output a yes/no answer

→ These results are crucial to build reliable and efficient floating-point implementations of mathematical functions with correct rounding

→ Impact on numerical software, including safety-critical systems

Goal:

- Guarantee the results that are produced by the SLZ algorithmic chain

→ Design certificates that fit in with independent verification

→ Use formal methods: the Coq proof assistant

Context and Motivations

Context:

- The SLZ algorithm for solving (offline) the Table Maker's Dilemma

→ Very long calculations using sophisticated, optimized methods

→ Either output some numerical data whose completeness cannot be directly verified, or output a yes/no answer

→ These results are crucial to build reliable and efficient floating-point implementations of mathematical functions with correct rounding

→ Impact on numerical software, including safety-critical systems

Goal:

- Guarantee the results that are produced by the SLZ algorithmic chain

→ Design certificates that fit in with independent verification

→ Use formal methods: the Coq proof assistant

The Coq proof assistant

We use Coq for

- programming
  - strongly typed functional language
  - computation
- proving
  - use higher order logic
  - build proofs interactively
  - program automatic tactics
  - check proofs

Computing within the Coq proof assistant

Coq comes with a primitive notion of computation, called reduction.

Three main reduction tactics are available:

1984: `compute`: reduction machine (inside the kernel)

2004: `vm_compute`: virtual machine (byte-code)

2011: `native_compute`: compilation (native-code)

Several levels of trust:

method	trust	speed
`compute`	+++	+
`vm_compute`	++	++
`native_compute`	+	+++

Numbers in Coq

1984:	`nat`	Peano
1994:	`positive`, `N`, `Z`	radix 2
1999:	`R`	a classical axiomatization of $\mathbb{R}$
2001:	`Float`	pair of integers
2008:	`bigN`, `bigZ`, `bigQ`	binary tree
2008:	`Interval`	parametric
2000:	C-CoRN	an intuitionistic axiomatization of $\mathbb{R}$
2008:	exact transcendental computation	exact reals

Floating-Point (FP) arithmetic

A finite, radix-β, precision-p FP number is a rational number of the form

$$x = M \times \beta^{e-p+1} \quad \text{with} \quad \begin{cases} (M, e) \in \mathbb{Z} \times \mathbb{Z} \\ |M| < \beta^p \\ e_{\mathsf{min}} \leqslant e \leqslant e_{\mathsf{max}} \end{cases} \tag{1}$$

- the smallest e satisfying (1) is called the exponent of x
- the corresponding M is called the integral significand of x
- x is said normal if $\beta^{p-1} \leqslant |M|$, otherwise it is subnormal and $e = e_{\mathsf{min}}$

Correct rounding

Definition (Rounding mode for a FP format $\mathbb{F}$)

A function $\circ : \mathbb{R} \to \mathbb{F} \cup \{\pm\infty\}$ satisfying

$$\begin{cases} \forall x, y \in \mathbb{R}, & x \leqslant y \implies \circ(x) \leqslant \circ(y), \\ \forall x \in \mathbb{R}, & x \in \mathbb{F} \implies \circ(x) = x. \end{cases}$$

Correct rounding

Definition (Rounding mode for a FP format $\mathbb{F}$)

An increasing function $\circ : \mathbb{R} \to \mathbb{F} \cup \{\pm\infty\}$ whose restriction to $\mathbb{F}$ is identity.

Example (Standard rounding modes)

toward $-\infty$: $\mathrm{RD}(x)$ is the largest FP number $\leqslant x$;

toward $+\infty$: $\mathrm{RU}(x)$ is the smallest FP number $\geqslant x$;

toward zero: $\mathrm{RZ}(x)$ is equal to $\mathrm{RD}(x)$ if $x \geqslant 0$, and to $\mathrm{RU}(x)$ if $x \leqslant 0$;

to nearest: $\mathrm{RN}(x)$ is the FP number closest to x. In case of a tie: the one whose integral significand is even ($\exists$ another tie-breaking rule)

Definition (Correctly rounded operation with respect to $\circ$)

For a given operation $* : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, an implementation that returns the value $\circ(x * y)$ for all $(x, y) \in \mathbb{F} \times \mathbb{F}$.

The IEEE 754 standard for floating-point arithmetic

IEEE 754-1985: requires correct rounding for $+$, $-$, $\times$, $\div$, $\sqrt{\cdot}$ and some conversions. Advantages:

- if the result of an operation is exactly representable, we get it;
- if we just use these correctly rounded operations, deterministic arithmetic

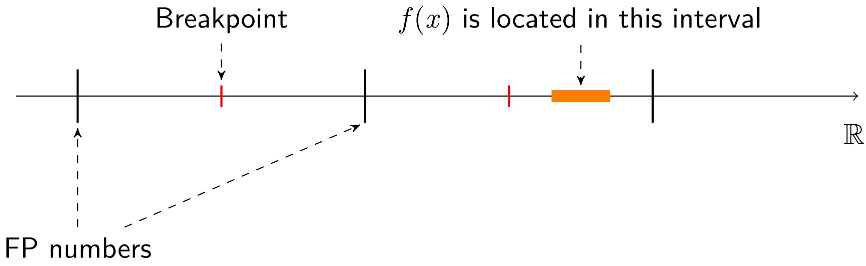
→ we can thus design algorithms and proofs that use the specifications;

- accuracy and portability are improved;

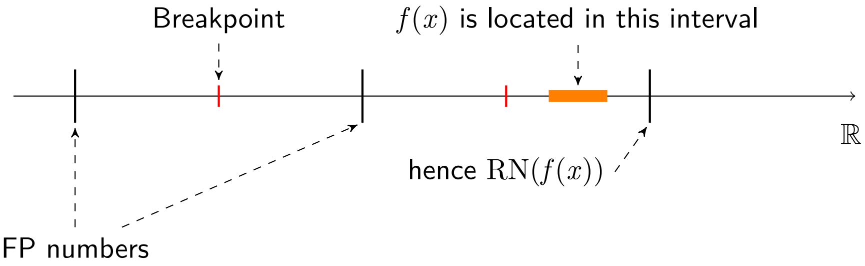
  ...

IEEE 754-2008: recommends correct rounding for standard mathematical functions

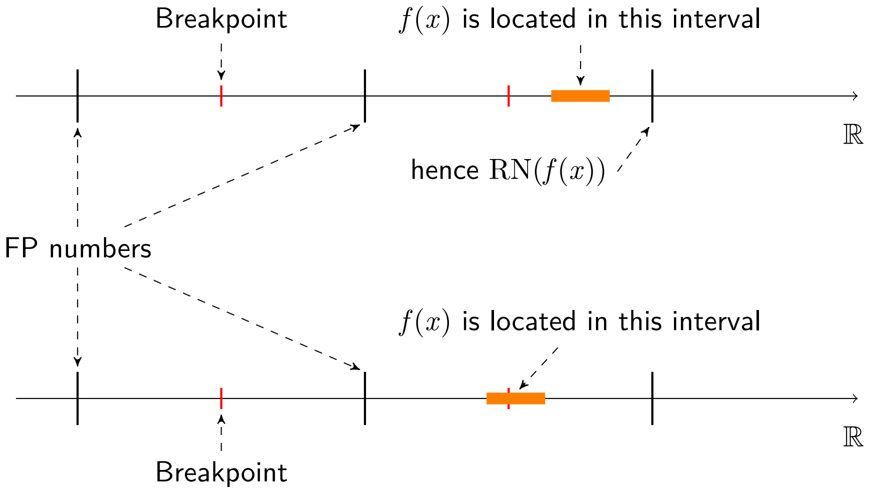
The Table Maker's Dilemma (TMD) (1/2)



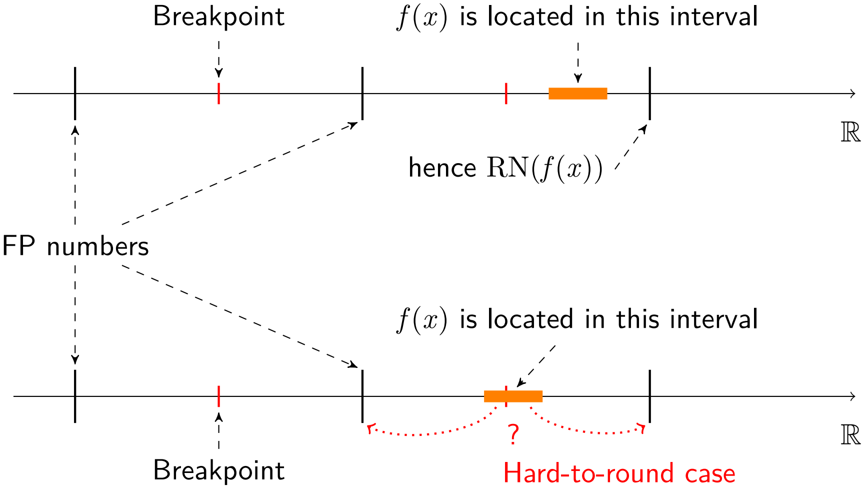
The Table Maker's Dilemma (TMD) (1/2)



The Table Maker's Dilemma (TMD) (1/2)



The Table Maker's Dilemma (TMD) (1/2)



The Table Maker's Dilemma (TMD) (2/2)

Solving the TMD = knowing the accuracy of the approximation that is required to avoid hard-to-round cases:

- either find the hardest-to-round cases of f: the FP values x such that $f(x)$ is closest to a breakpoint without being a breakpoint;
- or find a lower bound to the nonzero distance between $f(x)$ and a breakpoint.

The Table Maker's Dilemma (TMD) (2/2)

Solving the TMD = knowing the accuracy of the approximation that is required to avoid hard-to-round cases:

- either find the hardest-to-round cases of f: the FP values x such that $f(x)$ is closest to a breakpoint without being a breakpoint;
- or find a lower bound to the nonzero distance between $f(x)$ and a breakpoint.

Example of hardest-to-round (HR) case

The HR case of $\exp$ for decimal64 and rounding-to-nearest is:

$$x = 9.407822313572878 \times 10^{-2}$$

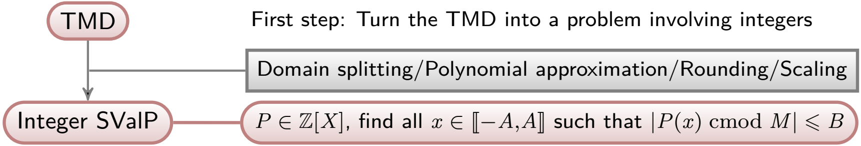
$$\exp(x) = 1.098645682066338\ 5\ 0000000000000000\ 278\ldots$$

The SLZ algorithm

TMD

First step: Turn the TMD into a problem involving integers

The SLZ algorithm



The SLZ algorithm

First step: Turn the TMD into a problem involving integers

TMD

↓ Domain splitting/Polynomial approximation/Rounding/Scaling

Integer SValP — $P \in \mathbb{Z}[X]$, find all $x \in [\![-A,A]\!]$ such that $|P(x) \text{ cmod } M| \leqslant B$

↓ $Q(X,Y) := P(X) - Y \in \mathbb{Z}[X,Y]$

Biv. Small Mod. Roots — Find all $(x,y) \in [\![-A,A]\!] \times [\![-B,B]\!]$ s.t. $Q(x,y) \equiv 0 \pmod{M}$

The SLZ algorithm

TMD

First step: Turn the TMD into a problem involving integers

Domain splitting/Polynomial approximation/Rounding/Scaling

Integer SValP

$P \in \mathbb{Z}[X]$, find all $x \in [\![-A, A]\!]$ such that $|P(x) \operatorname{cmod} M| \leqslant B$

$Q(X,Y) := P(X) - Y \in \mathbb{Z}[X,Y]$

Biv. Small Mod. Roots

Find all $(x, y) \in [\![-A, A]\!] \times [\![-B, B]\!]$ s.t. $Q(x, y) \equiv 0 \pmod{M}$

Coppersmith's technique with parameter $\alpha > 0$:
Consider $Q_{i,j}(X,Y) = X^i Q(X,Y)^j M^{\alpha-j} \quad (j \leqslant \alpha)$.
Heuristically, find two $\mathbb{Z}$-linear combinations v_1, v_2 of $(Q_{i,j})$ s.t.

$$\forall x, y \in \mathbb{Z},\ |x| \leqslant A \ \wedge\ |y| \leqslant B \implies |v_k(x,y)| < M^\alpha.$$

Notice that the small modular roots of Q mod M also satisfy

$$v_k(x,y) \equiv 0 \pmod{M^\alpha}.$$

The SLZ algorithm

First step: Turn the TMD into a problem involving integers

TMD

↓ Domain splitting/Polynomial approximation/Rounding/Scaling

Integer SValP: $P \in \mathbb{Z}[X]$, find all $x \in [\![-A, A]\!]$ such that $|P(x) \operatorname{cmod} M| \leqslant B$

↓ $Q(X,Y) := P(X) - Y \in \mathbb{Z}[X,Y]$

Biv. Small Mod. Roots: Find all $(x,y) \in [\![-A,A]\!] \times [\![-B,B]\!]$ s.t. $Q(x,y) \equiv 0 \pmod M$

↓ **Coppersmith's technique** with parameter $\alpha > 0$:
Consider $Q_{i,j}(X,Y) = X^i Q(X,Y)^j M^{\alpha - j} \quad (j \leqslant \alpha)$.
Heuristically, find two $\mathbb{Z}$-linear combinations v_1, v_2 of $(Q_{i,j})$ s.t.

$$\forall x, y \in \mathbb{Z},\ |x| \leqslant A \ \wedge\ |y| \leqslant B \implies |v_k(x,y)| < M^\alpha.$$

Notice that the small modular roots of Q mod M also satisfy

$$v_k(x,y) \equiv 0 \pmod{M^\alpha}.$$

Order-2 Small Int. Roots: Find all $(x,y) \in [\![-A,A]\!] \times [\![-B,B]\!]$ s.t. $v_1(x,y) = 0 = v_2(x,y)$

↓ Bivariate Hensel lifting

The SLZ algorithm

CoqApprox

TMD

First step: Turn the TMD into a problem involving integers

Domain splitting/Polynomial approximation/Rounding/Scaling

Integer SValP

$P \in \mathbb{Z}[X]$, find all $x \in [\![-A, A]\!]$ such that $|P(x) \text{ cmod } M| \leqslant B$

$Q(X,Y) := P(X) - Y \in \mathbb{Z}[X,Y]$

Biv. Small Mod. Roots

Find all $(x,y) \in [\![-A, A]\!] \times [\![-B, B]\!]$ s.t. $Q(x,y) \equiv 0 \pmod{M}$

Coppersmith's technique with parameter $\alpha > 0$:
Consider $Q_{i,j}(X,Y) = X^i Q(X,Y)^j M^{\alpha - j} \quad (j \leqslant \alpha)$.
Heuristically, find two $\mathbb{Z}$-linear combinations v_1, v_2 of $(Q_{i,j})$ s.t.

$$\forall x, y \in \mathbb{Z},\ |x| \leqslant A \ \wedge\ |y| \leqslant B \implies |v_k(x,y)| < M^\alpha.$$

Notice that the small modular roots of Q mod M also satisfy

$$v_k(x,y) \equiv 0 \pmod{M^\alpha}.$$

Order-2 Small Int. Roots

Find all $(x,y) \in [\![-A, A]\!] \times [\![-B, B]\!]$ s.t. $v_1(x,y) = 0 = v_2(x,y)$

Bivariate Hensel lifting

CoqHensel

Outline

Outline

Rigorous approximation of functions by polynomials (1/2)

- 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 SLZ algorithm

Rigorous approximation of functions by polynomials (2/2)

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

Rigorous approximation of functions by polynomials (2/2)

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

From rigorous to formally verified polynomial approximation:

- A computational implementation of Taylor Models in Coq
- Formal proofs that the provided error bounds are not underestimated

Brief overview of Interval Arithmetic (IA)

- Interval = pair of real numbers (or floating-point numbers)
- E.g., $[3.1415,\ 3.1416] \ni \pi$
- Operations on intervals, e.g., $[2,4]-[0,1] := [2-1,4-0] = [1,4]$, with the enclosure property: $\forall x \in [2,4],\ \forall y \in [0,1],\ x-y \in [1,4]$.
- Tool for bounding the range of functions

- Dependency problem: for $f(x) = x \cdot (1-x)$ and $\boldsymbol{X} = [0,1]$, a naive use of IA gives $\mathrm{eval}(f, \boldsymbol{X}) = [0,1]$ while the image of $\boldsymbol{X}$ by f is $[0, \frac{1}{4}]$
- IA is not directly applicable to bound approximation errors $e := p - f$

Rigorous Polynomial Approximation

Definition

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

Rigorous Polynomial Approximation

Definition

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

Various possible instances of RPAs, depending on the polynomial basis and on the algorithms that are used:

Taylor Models: truncated Taylor series, naturally expressed in Taylor basis

Chebyshev Models: Chebyshev interpolants / truncated Chebyshev series

Rigorous Polynomial Approximation

Definition

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

Various possible instances of RPAs, depending on the polynomial basis and on the algorithms that are used:

Taylor Models: truncated Taylor series, naturally expressed in Taylor basis

Chebyshev Models: Chebyshev interpolants / truncated Chebyshev series

Taylor Models in CoqApprox

As regards $\boldsymbol{\Delta}$: interval remainder with floating-point bounds;
As regards P: small interval coefficients with floating-point bounds
$\Longrightarrow$ rounding errors are directly handled by the interval arithmetic

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.:

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

Outline

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

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

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

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.:

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

Outline

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

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

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

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

Methodology of Taylor Models

Define arithmetic operations on Taylor Models:

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

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:

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

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 (a.k.a. holonomic functions)

Definition

A D-finite function is a solution of a homogeneous linear ordinary differential equation with polynomial coefficients:
$a_r(x)y^{(r)}(x) + \cdots + a_1(x)y'(x) + a_0(x)y(x) = 0$, for given $a_k \in \mathbb{K}[X]$.

Property

The Taylor coefficients of these functions satisfy a *linear recurrence with polynomial coefficients*

D-finite functions (a.k.a. holonomic functions)

Definition

A D-finite function is a solution of a homogeneous linear ordinary differential equation with polynomial coefficients:
$a_r(x)y^{(r)}(x) + \cdots + a_1(x)y'(x) + a_0(x)y(x) = 0$, for given $a_k \in \mathbb{K}[X]$.

Property

The Taylor coefficients of these functions satisfy a *linear recurrence with polynomial coefficients* $\rightarrow$ fast numerical computation of the coefficients

Example (the exponential function)

The Taylor coefficients of $\exp$ at x_0 satisfy the recurrence
$\forall n \in \mathbb{N},\ (n+1)u_{n+1} = u_n$, with $u_0 = \exp(x_0)$ as an initial condition.

$\ln, \sin, \arcsin, \sinh, \operatorname{arcsinh}, \arctan, \operatorname{arctanh}\ldots$ are D-finite; $\tan$ is not

Formally verified computation: CoqInterval

- Abstract interface for intervals
- Instantiation to intervals with floating-point bounds
- Formal verification with respect to the Reals library

for $x, y : \mathbb{R}$

and $\boldsymbol{X}, \boldsymbol{Y} : \mathbb{IR}$

$$x \in \boldsymbol{X} \ \wedge \ y \in \boldsymbol{Y} \implies x + y \in \boldsymbol{X} + \boldsymbol{Y}$$

$$x \in \boldsymbol{X} \implies \exp(x) \in \mathbf{exp}(\boldsymbol{X})$$

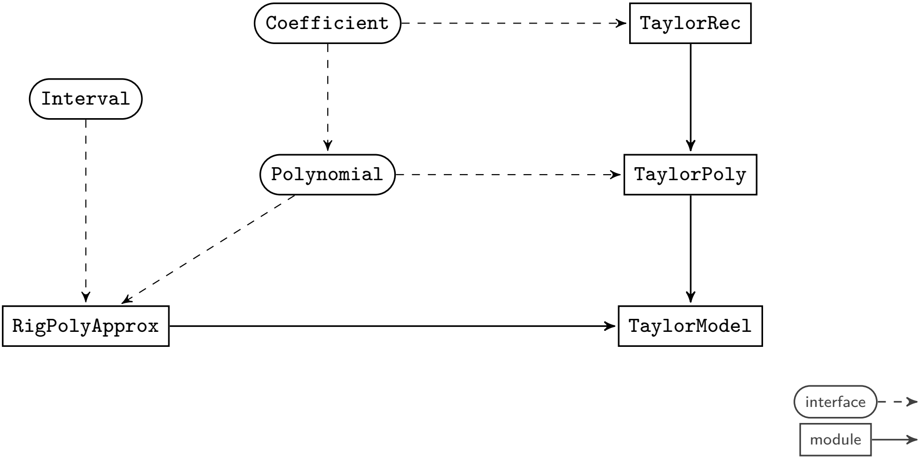
Implementation of Taylor Models in Coq

Focus on being generic:

- a Taylor Model is an instance of a Rigorous Polynomial Approximation, i.e., a pair $(P, \mathbf{\Delta})$
- generic with respect to
  - the type of coefficients of polynomial P,
  - the type of P and the implementation of related operations
  - the type of interval $\mathbf{\Delta}$

Prove correctness with respect to the standard Reals library

A modular implementation of Taylor Models



Comparison with a dedicated tool implemented in C

Sollya [S.Chevillard, M.Joldeș, C.Lauter]

- written in C
- based on the MPFI library
- contains an implementation of univariate Taylor Models
- in an imperative-programming framework
- polynomials as arrays of coefficients

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)

Coq is around 10 times slower than Sollya! It's very good!

Some benchmarks for base functions

	Timing		Approximation error		
	Coq	Sollya	Coq	Sollya	Mathematical
f = exp prec=1000, deg=70 $\boldsymbol{I}=[127/128, 1]$	0.716s	0.093s	1.80×2^{-906}	1.79×2^{-906}	1.79×2^{-906}
f = sin prec=1000, deg=70 $\boldsymbol{I}=[127/128, 1]$	2.636s	0.088s	1.45×2^{-908}	1.44×2^{-908}	1.44×2^{-908}
f = arctan prec=1000, deg=118 $\boldsymbol{I}=[127/128, 1]$	2.969s	0.420s	1.71×2^{-913}	1.30×2^{-967}	1.07×2^{-1001}

- with Coq v8.3pl4 using `vm_compute`,
- and Sollya v3.0 using `taylorform()`, along with `supnorm()` for last column.

Some benchmarks for composite functions

	Timing		Approximation error		
	Coq	Sollya	Coq	Sollya	Mathematical
f = exp $\times$ sin prec=400, deg=20 $\boldsymbol{I}$=$[127/128, 1]$	0.812s	0.013s	1.36×2^{-222}	1.36×2^{-222}	1.36×2^{-222}
f = exp $\times$ sin prec=400, deg=40 $\boldsymbol{I}$=$[127/128, 1]$	1.736s	0.040s	1.01×2^{-397}	1.53×2^{-397}	1.06×2^{-402}
f = exp $\circ$ sin prec=400, deg=20 $\boldsymbol{I}$=$[127/128, 1]$	7.165s	0.011s	1.56×2^{-192}	1.83×2^{-192}	1.56×2^{-192}
f = exp $\circ$ sin prec=400, deg=40 $\boldsymbol{I}$=$[127/128, 1]$	52.687s	0.065s	1.88×2^{-385}	1.38×2^{-384}	1.88×2^{-385}

- with Coq v8.3pl4 using `vm_compute`,
- and Sollya v3.0 using `taylorform()`, along with `supnorm()` for last column.

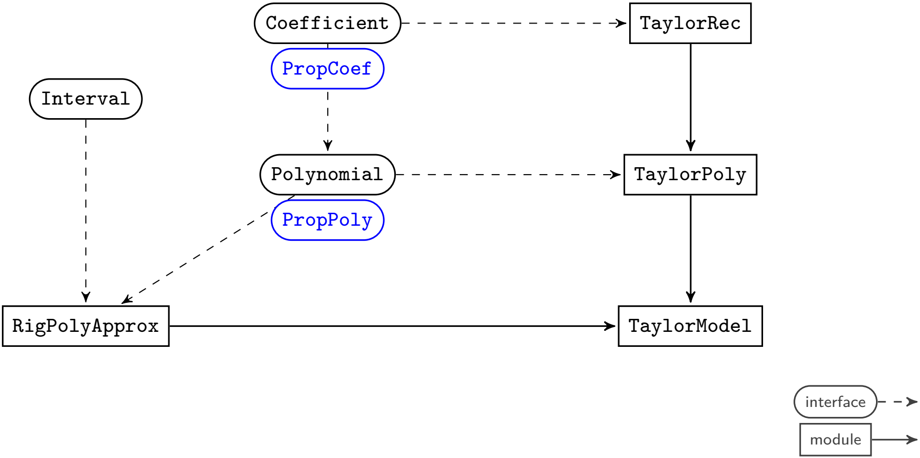
Proving Taylor Models in Coq

Definition

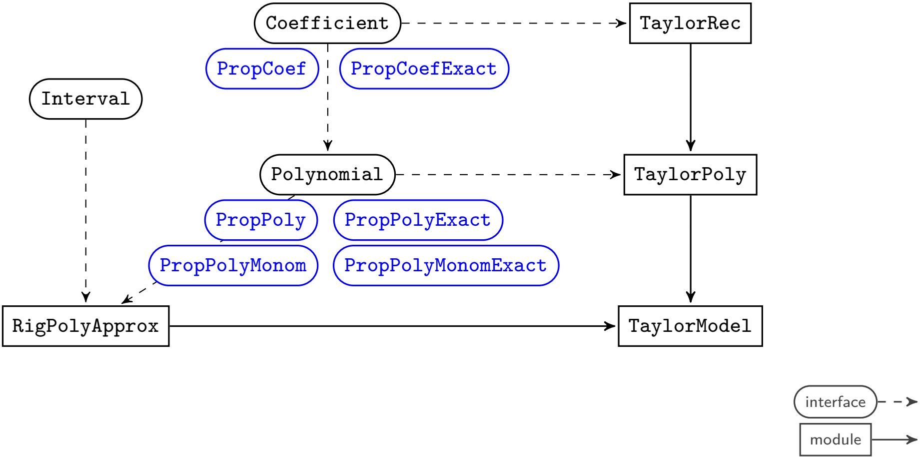
Let $f : \boldsymbol{I} \to \mathbb{R}$ be a function, $\boldsymbol{x_0}$ be a small interval around an expansion point x_0. Let T be a polynomial with interval coefficients $\boldsymbol{a_0}, \ldots, \boldsymbol{a_n}$ and $\boldsymbol{\Delta}$ an interval. We say that $(T, \boldsymbol{\Delta})$ is a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

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

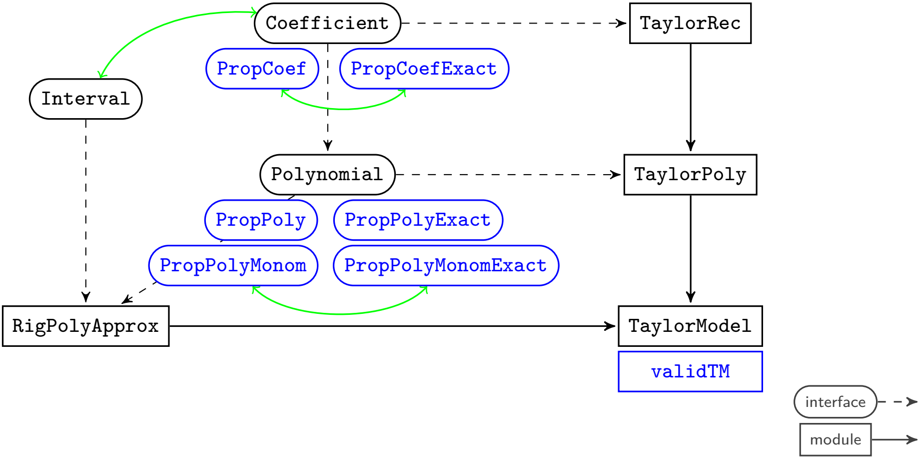
Extending the hierarchy to handle proofs



Extending the hierarchy to handle proofs



Extending the hierarchy to handle proofs



Idea of the proof of TMs for the exponential

$\mathtt{TM}_{\mathtt{exp}}(\boldsymbol{x_0}, \boldsymbol{I}, n) := (\boldsymbol{a_0} :: \ldots :: \boldsymbol{a_n}, \boldsymbol{\Delta})$ with

$$\boldsymbol{x_0} \subset \boldsymbol{I}, \quad \boldsymbol{a_0} = \mathbf{exp}(\boldsymbol{x_0}), \quad \boldsymbol{a_{n+1}} = \frac{\boldsymbol{a_n}}{n+1}, \quad \boldsymbol{\Delta} = \frac{\mathbf{exp}(\boldsymbol{I})}{(n+1)!} \times (\boldsymbol{I} - \boldsymbol{x_0})^{n+1}.$$

Idea of the proof of TMs for the exponential

$\mathtt{TM}_{\mathtt{exp}}(\boldsymbol{x_0}, \boldsymbol{I}, n) := (\boldsymbol{a_0} :: \ldots :: \boldsymbol{a_n}, \boldsymbol{\Delta})$ with

$$\boldsymbol{x_0} \subset \boldsymbol{I}, \quad \boldsymbol{a_0} = \mathbf{exp}(\boldsymbol{x_0}), \quad \boldsymbol{a_{n+1}} = \frac{\boldsymbol{a_n}}{n+1}, \quad \boldsymbol{\Delta} = \frac{\mathbf{exp}(\boldsymbol{I})}{(n+1)!} \times (\boldsymbol{I} - \boldsymbol{x_0})^{n+1}.$$

We want to show that $\mathtt{TM}_{\mathtt{exp}}(\boldsymbol{x_0}, \boldsymbol{I}, n)$ is a valid TM for $\exp$:

- $\boldsymbol{x_0} \subset \boldsymbol{I}$,
- $0 \in \boldsymbol{\Delta}$,
- $\forall \xi_0 \in \boldsymbol{x_0}, \exists \alpha_0 \in \boldsymbol{a_0}, \ldots, \alpha_n \in \boldsymbol{a_n},$

  $$\forall x \in \boldsymbol{I},\ \exp(x) - \sum_{i=0}^{n} \alpha_i \,(x - \xi_0)^i \in \boldsymbol{\Delta}.$$

Idea of the proof of TMs for the exponential

$\texttt{TM}_{\texttt{exp}}(\boldsymbol{x_0}, \boldsymbol{I}, n) := (\boldsymbol{a_0} :: \ldots :: \boldsymbol{a_n}, \boldsymbol{\Delta})$ with

$$\boldsymbol{x_0} \subset \boldsymbol{I}, \quad \boldsymbol{a_0} = \mathbf{exp}(\boldsymbol{x_0}), \quad \boldsymbol{a_{n+1}} = \frac{\boldsymbol{a_n}}{n+1}, \quad \boldsymbol{\Delta} = \frac{\mathbf{exp}(\boldsymbol{I})}{(n+1)!} \times (\boldsymbol{I} - \boldsymbol{x_0})^{n+1}.$$

We want to show that $\texttt{TM}_{\texttt{exp}}(\boldsymbol{x_0}, \boldsymbol{I}, n)$ is a valid TM for $\exp$:

- $\boldsymbol{x_0} \subset \boldsymbol{I}$,
- $0 \in \boldsymbol{\Delta}$,
- $\forall \xi_0 \in \boldsymbol{x_0}, \exists \alpha_0 \in \boldsymbol{a_0}, \ldots, \alpha_n \in \boldsymbol{a_n}$,

  $$\forall x \in \boldsymbol{I},\ \exp(x) - \sum_{i=0}^{n} \alpha_i \, (x - \xi_0)^i \in \boldsymbol{\Delta}.$$

$\exists \alpha_i = \dfrac{\exp(\xi_0)}{i!} \in \boldsymbol{a_i}$ such that for all $x \in \boldsymbol{I}$,

$$\exp(x) - \sum_{i=0}^{n} \frac{\exp(\xi_0)}{i!} (x - \xi_0)^i = \frac{\exp(\xi)}{(n+1)!} \times (x - \xi_0)^{n+1} \text{ for some } \xi \in \boldsymbol{I}.$$

Generalization to an arbitrary D-finite function f

Difficulties:

- Find minimal assumptions on the function f
  - the derivative is compatible with the recurrence relation
  - we have a compatible interval evaluator for f
- Provide the Taylor-Lagrange theorem for standard Reals

$\leadsto$ Generic proof for first-order and second-order recurrences.

Proofs for composite functions

Proof of the algorithm for each algebraic rule

- $\mathtt{TM}_{\mathtt{add}}$: straightforward
- $\mathtt{TM}_{\mathtt{mul}}$: rely on truncated multiplication of polynomials
- $\mathtt{TM}_{\mathtt{comp}}$: rely on $\mathtt{TM}_{\mathtt{mul}}$, $\mathtt{TM}_{\mathtt{add}}$ and TMs for constant functions
- $\mathtt{TM}_{\mathtt{div}}$: it's a TM for $f \times \left(\left(x \mapsto \frac{1}{x}\right) \circ g\right)$

Functions missing from support libraries

Functions missing from the Reals library

- cannot provide a proof for the Taylor Model
- adding them is so far done in a case-by-case manner

$\rightarrow$ find a generic way of adding a new function to Reals

$\rightarrow$ e.g. by using a differential equation or a recurrence relation as definition

Functions missing from support libraries

Functions missing from the Reals library

- cannot provide a proof for the Taylor Model
- adding them is so far done in a case-by-case manner

→ find a generic way of adding a new function to Reals

→ e.g. by using a differential equation or a recurrence relation as definition

Functions missing from CoqInterval

- cannot provide an initial value for the Taylor Model

→ just implement the missing functions in CoqInterval

→ may use other techniques (e.g., fixed point theorems)

Outline

Goal: certifying the SLZ algorithm

Integer SValP — $P \in \mathbb{Z}[X]$, find all $x \in [\![-A, A]\!]$ such that $|P(x) \operatorname{cmod} M| \leqslant B$

$Q(X,Y) := P(X) - Y \in \mathbb{Z}[X,Y]$

Biv. Small Mod. Roots — Find all $(x, y) \in [\![-A, A]\!] \times [\![-B, B]\!]$ s.t. $Q(x, y) \equiv 0 \pmod{M}$

Coppersmith's technique with parameter $\alpha > 0$:
Consider $Q_{i,j}(X,Y) = X^i Q(X,Y)^j M^{\alpha - j} \quad (j \leqslant \alpha)$.
Heuristically, find two $\mathbb{Z}$-linear combinations v_1, v_2 of $(Q_{i,j})$ s.t.

$$\forall x, y \in \mathbb{Z},\ |x| \leqslant A \ \wedge\ |y| \leqslant B \implies |v_k(x, y)| < M^\alpha.$$

Notice that the small modular roots of Q mod M also satisfy

$$v_k(x, y) \equiv 0 \pmod{M^\alpha}.$$

Order-2 Small Int. Roots — Find all $(x, y) \in [\![-A, A]\!] \times [\![-B, B]\!]$ s.t. $v_1(x, y) = 0 = v_2(x, y)$

Bivariate Hensel lifting

Goal: certifying the SLZ algorithm

Integer SValP: $P \in \mathbb{Z}[X]$, find all $x \in [\![-A,A]\!]$ such that $|P(x) \text{ cmod } M| \leqslant B$

↓ $Q(X,Y) := P(X) - Y \in \mathbb{Z}[X,Y]$

Biv. Small Mod. Roots: Find all $(x,y) \in [\![-A,A]\!] \times [\![-B,B]\!]$ s.t. $Q(x,y) \equiv 0 \pmod{M}$

↓ **Coppersmith's technique** with parameter $\alpha > 0$:
Consider $Q_{i,j}(X,Y) = X^i Q(X,Y)^j M^{\alpha-j} \quad (j \leqslant \alpha)$.
Heuristically, find two $\mathbb{Z}$-linear combinations v_1, v_2 of $(Q_{i,j})$ s.t.

$$\forall x,y \in \mathbb{Z},\ |x| \leqslant A \ \wedge\ |y| \leqslant B \implies |v_k(x,y)| < M^\alpha.$$

Notice that the small modular roots of Q mod M also satisfy

$$v_k(x,y) \equiv 0 \pmod{M^\alpha}.$$

Order-2 Small Int. Roots: Find all $(x,y) \in [\![-A,A]\!] \times [\![-B,B]\!]$ s.t. $v_1(x,y) = 0 = v_2(x,y)$

↓ Bivariate Hensel lifting

certificate

Main steps of the formalization

1. Define bivariate Hensel lifting *as a fixpoint*;
2. Prove bivariate Hensel's lemma;
3. Define order-2 SIntRootP certificates *as an inductive type*;
4. Define order-2 SIntRootP checker *as a Boolean predicate*;
5. Prove its soundness: if a certificate is *accepted* then it is *valid*;
6. Define ISValP certificates;
7. Define ISValP checker;
8. Prove its soundness;
9. Redo steps 3 and 4, 6 and 7 in a generic way to allow one to instantiate the checkers with efficient datatypes;
10. Derive the final correctness proofs, using steps 5 and 8 as well as a series of *homomorphisms lemmas rewritings*.

Main steps of the formalization

1. Define bivariate Hensel lifting *as a fixpoint*;
2. Prove bivariate Hensel's lemma;
3. Define order-2 SIntRootP certificates *as an inductive type*;
4. Define order-2 SIntRootP checker *as a Boolean predicate*;
5. Prove its soundness: if a certificate is *accepted* then it is *valid*;
6. Define ISValP certificates;
7. Define ISValP checker;
8. Prove its soundness;
9. Redo steps 3 and 4, 6 and 7 in a generic way to allow one to instantiate the checkers with efficient datatypes;
10. Derive the final correctness proofs, using steps 5 and 8 as well as a series of *homomorphisms lemmas rewritings*.

Main steps of the formalization

1. Define bivariate Hensel lifting *as a fixpoint*;
2. Prove bivariate Hensel's lemma;
3. Define order-2 SIntRootP certificates *as an inductive type*;
4. Define order-2 SIntRootP checker *as a Boolean predicate*;
5. Prove its soundness: if a certificate is *accepted* then it is *valid*;
6. Define ISValP certificates;
7. Define ISValP checker;
8. Prove its soundness;
9. Redo steps 3 and 4, 6 and 7 in a generic way to allow one to instantiate the checkers with efficient datatypes;
10. Derive the final correctness proofs, using steps 5 and 8 as well as a series of *homomorphisms lemmas rewritings*.

Bivariate Hensel lifting

Algorithm 1: Bivariate Hensel lifting (quadratic version)

Input : $P_1, P_2 \in \mathbb{Z}[X,Y]$,
$p \in \mathbb{P}$,
$(u_k, v_k) \in \mathbb{Z}^2$ s.t. $P_i(u_k, v_k) \equiv 0 \pmod{p^{2^k}}, i = 1, 2$,
and $\det J_{P_1,P_2}(u_k, v_k) \not\equiv 0 \pmod{p}$.

Output: $(u_{k+1}, v_{k+1}) \in \mathbb{Z}^2$ s.t. $P_i(u_{k+1}, v_{k+1}) \equiv 0 \pmod{p^{2^{k+1}}}, i = 1, 2.$

$$\begin{pmatrix} u_{k+1} \\ v_{k+1} \end{pmatrix} \leftarrow \begin{pmatrix} u_k \\ v_k \end{pmatrix} - \Big[J_{P_1,P_2}(u_k, v_k) \Big]^{-1}_{p^{2^{k+1}}} \begin{pmatrix} P_1(u_k, v_k) \\ P_2(u_k, v_k) \end{pmatrix} \bmod p^{2^{k+1}}$$

Hensel's lemma: a uniqueness result for modular roots

Let $P_1, P_2 \in \mathbb{Z}[X,Y]$ and let p be a prime satisfying

$\forall z, t \in \mathbb{Z}, P_1(z,t) \equiv 0 \equiv P_2(z,t) \pmod{p} \Rightarrow \det J_{P_1,P_2}(z,t) \not\equiv 0 \pmod{p}$.

For any $(x,y) \in \mathbb{Z} \times \mathbb{Z}$, if we have $P_1(x,y) \equiv 0 \equiv P_2(x,y) \pmod{p^{2^k}}$ for a given $k \in \mathbb{N}$, then for

$$\begin{pmatrix} u_0 \\ v_0 \end{pmatrix} := \begin{pmatrix} x \bmod p \\ y \bmod p \end{pmatrix},$$

the sequence $(u_i, v_i)_i$ defined by the recurrence relation

$$\forall i \in [\![0, k[\![\,,\ \begin{pmatrix} u_{i+1} \\ v_{i+1} \end{pmatrix} := \begin{pmatrix} u_i \\ v_i \end{pmatrix} - \Big[J_{P_1,P_2}(u_i, v_i) \Big]^{-1}_{p^{2^{i+1}}} \begin{pmatrix} P_1(u_i, v_i) \\ P_2(u_i, v_i) \end{pmatrix} \bmod p^{2^{i+1}}$$

satisfies:

$$\forall i \in [\![0, k]\!]\,,\ \begin{pmatrix} u_i \\ v_i \end{pmatrix} = \begin{pmatrix} x \bmod p^{2^i} \\ y \bmod p^{2^i} \end{pmatrix}.$$

Order-2 SIntRootP certificates

```
Record bivCertif : Set := BivCertif
{ bc_P1 : {bipoly Z}
; bc_P2 : {bipoly Z}
; bc_A : Z
; bc_B : Z
; bc_p : nat
; bc_k : nat
; bc_L : seq (Z * Z * bool)
}.
```

Order-2 SIntRootP certificates checker

Our implemented checker will accept such a certificate $(P_1, P_2, A, B, p, k, L)$ iff

- $p \in \mathbb{P}$
- $p^{2^k} > 2A$ and $p^{2^k} > 2B$
- L contains only simultaneous roots of (P_1, P_2) modulo p^{2^k}, of absolute value $\leqslant p^{2^k}/2$, and all roots modulo p are present
- for all $(u, v, b) \in L$,
  - $J_{P_1,P_2}(u, v)$ is invertible modulo p
  - the Boolean b is true iff (u, v) is an actual root in $\mathbb{Z}$

ISValP certificates

```
Record cert_ISValP : Set := Cert_ISValP
{ c_P : {poly Z} (* hence Q(X,Y) = P(Y) − X *)
; c_M : Z
; c_alpha : positive
; c_A : Z
; c_B : Z
; c_u1 : {bipoly Z} (* in basis M^(α−i) × Q^i(X,Y) × Y^j *)
; c_u2 : {bipoly Z} (* in basis M^(α−i) × Q^i(X,Y) × Y^j *)
; c_p : nat
; c_k : nat
; c_L : seq (Z * Z * bool)
}.
```

ISValP certificates checker

```
Definition check_ISValP (C : cert_ISValP) : bool :=
  let: Cert_ISValP P M alpha A B u1 u2 p k L := C in
  let Q := poly_cons P (bipolyC (-1)) in
  let v1 := (bipoly_precalc_alpha u1 alpha M) \Po Q in
  let v2 := (bipoly_precalc_alpha u2 alpha M) \Po Q in
  let Ma := Zpower_pos M alpha in
  let C' := BivCertif v1 v2 A B p k L in
  [&& 0 < M,
  bimaphorner Zabs A B v1 < Zabs Ma,
  bimaphorner Zabs A B v2 < Zabs Ma
  & biv_check C'].
```

Concepts and libraries involved in the bivariate proofs

- Signed integers $(\mathbb{Z})$ with exponentiation and modulus $\rightsquigarrow$ `ssrzarith`

- Small natural numbers $(\mathbb{N})$ with primality predicate $\rightsquigarrow$ `ssrnat`, `prime`
- Rings $\mathbb{Z}/p^m\mathbb{Z}$, modular inversion and divisibility results $\rightsquigarrow$ `zmodp`, `div`

- Ring $\mathbb{Z}[X, Y]$ of bivariate polynomials over $\mathbb{Z}$, with Horner evaluation and Taylor theorem $\rightsquigarrow$ `bipoly`, based on `poly` and `ssralg`
- Need to manipulate a number of summations, typically after the invocations of Taylor theorem $\rightsquigarrow$ `bigop`

- We also developed some material specific to 2-by-2 matrices, including a modular version of Cramer rule whose correctness proof is

$$\forall A \in \mathcal{M}_2(\mathbb{Z}),\ u \in \mathbb{Z}^2,\ k \in \mathbb{N},\ \det A \not\equiv 0 \pmod{p} \Rightarrow A\left(A^{-1}u\right) \equiv u \pmod{p^{2^{k+1}}}$$

A generic implementation for effective certificates checkers

- Most of `poly` data structures are not computational
- Goal 1: allow to check integral-roots certificates inside Coq
- Goal 2: allow to easily change data structures to speedup computation

→ Define generic checkers once-and-for-all and instantiate them with the desired integer operations to avoid duplication of code

→ Proof: Reuse the reference lemmas proved with SSReflect datatypes and the rewriting lemmas that link both implementations:

```
Module Type CalcRingSig.
Parameters (T : Type) (R : comRingType) (toR : T -> R).
Parameter tadd : T -> T -> T.
Parameter toR_add :
  forall a b, toR (tadd a b) = (toR a + toR b)%R.
...
```

An implementation of "Integers Plus Positive Exponent"

- Big ISValP certificates $\rightsquigarrow$ coefficients scaled with a big power of 2 (e.g., $(2n+1)\times 2^{10629}$)
- Develop a specialized instance of computational integers to handle these integers

$\rightarrow$ Consider pairs $(m, e) \in$ `bigZ` $\times$ `bigN` for unevaluated dyadic numbers $m \times 2^e$ with $e \geqslant 0$

$\rightarrow$ Implement a generic module using a subset of the CoqInterval library

```
Module CalcRingIPPE (Import C : FloatCarrier)
  (Import E : CalcRingExpo C) <: CalcRingIntSig.
Notation typeZ := smantissa_type.
Record T := TZN { TZ : typeZ; TN : typeN }.
...
```

$\rightsquigarrow$ Speedup of 2x

Benchmarks for the ISValP certificates checker ($f = \exp$)

Inst.	prec	prec′	$\deg(P)$	$\max_i(\lvert P_i \rvert)$	M	A	B
#1	53	100	2	$\lessapprox 1.68 \times 2^{237}$	2^{185}	2^{139}	2^{12}
#2	53	100	2	$\lessapprox 1.22 \times 2^{237}$	2^{185}	2^{139}	2^{12}
#3	53	300	12	$\lessapprox 1.36 \times 2^{996}$	2^{942}	2^{696}	2^{32}
#4	113	3000	90	$\lessapprox 1.36 \times 2^{13661}$	2^{13547}	2^{10661}	2^{72}

Inst.	α	M^α	p	k	$\# L$	time to parse	time to return true
#1	2	2^{370}	5	6	1	0.096s	0.092s
#2	2	2^{370}	7	6	2	0.132s	0.112s
			3	7	1	0.112s	0.092s
			23	5	0	0.088s	0.172s
#3	4	2^{3768}	5	9	0	0.420s	2.348s
#4	6	2^{81282}	5	14	0	17.4s	3h12m42s

Outline

Contributions

1. **CoqApprox**: a modular formalization of Taylor Models in the Coq proof assistant
   - with a generic approach involving D-finite functions
   - taking advantage of the CoqInterval library for interval arithmetic

   $\rightarrow$ ability to compute some formally verified TMs in Coq

2. **CoqHensel**: formalization of some effective checkers in Coq for small-integral-roots problems as well as ISValP
   - using Hensel lifting as a certifying algorithm
   - relying on `ZArith`, `BigZ`, CoqInterval as well as SSReflect

   $\rightarrow$ ensure that no hard-to-round case for correct rounding has been forgotten

& Augmented computation of $\sqrt{x^2 + y^2}$ **&** Fast2Sum with double roundings

Perspectives

1. For CoqApprox:
   - add more functions
   - combine TMs with some Sums-of-Squares technique
   - implement Chebyshev Models $\rightsquigarrow$ tighter remainders
   - investigate ways to ease the definition of RPAs from the ODE
   - investigate ways to verify error bounds *a posteriori*
2. For CoqHensel:
   - implement a fast algorithm for the multiplication over $\mathbb{Z}[X]$, and/or for the composition over $\mathbb{Z}[X, Y]$
   - combine CoqHensel & CoqApprox to get a complete TMD checker
   - consider a possible extension of Hensel lifting to rational roots of polynomials
3. On formal floating-point:
   - formalize Thm 7.3 (TwoSum with double roundings), Thm 6.4 (2D norms)
   - investigate ways to ease similar formal proofs

End of the Talk

Thank you for your attention!

The TaMaDi project homepage:
`http://tamadi.gforge.inria.fr/`