Coq.Interval

Floating-Point and Interval Arithmetic for the Coq Proof Assistant

Guillaume Melquiond

2011-06-14

Motivations

- Perform efficient numerical computations in Coq.
- Use them to prove theorems about real numbers,
  e.g. distance between a function and its approximation.

Motivations

- Perform efficient numerical computations in Coq.
  $\Longrightarrow$ floating-point arithmetic.
- Use them to prove theorems about real numbers,
  e.g. distance between a function and its approximation.

Motivations

- Perform efficient numerical computations in Coq.
  $\Longrightarrow$ floating-point arithmetic.

- Use them to prove theorems about real numbers,
  e.g. distance between a function and its approximation.
  $\Longrightarrow$ interval arithmetic.

Library Components

Coq.Interval contains:

- a kernel for floating-point arithmetic,
- a layer of interval arithmetic,
- a tactic for proving numerical bounds.

Outline

Floating-Point Numbers

Definition (Floating-point numbers)

Floating-point numbers in radix β ($\mathbb{F}_\beta$):

- Pairs of integers $(m, e) \in \mathbb{Z}^2$ interpreted as $m \cdot \beta^e \in \mathbb{R}$.
- Not-a-Number $\perp$ for exceptional behavior: $\frac{1}{0}$, $\sqrt{-1}$, etc.

Floating-Point Numbers

Definition (Floating-point numbers)

Floating-point numbers in radix β ($\mathbb{F}_\beta$):

- Pairs of integers $(m, e) \in \mathbb{Z}^2$ interpreted as $m \cdot \beta^e \in \mathbb{R}$.
- Not-a-Number $\perp$ for exceptional behavior: $\frac{1}{0}$, $\sqrt{-1}$, etc.

- Unbounded exponent range $\Longrightarrow$ no overflow nor underflow.

  No need for infinities nor signed zeros.

- Representations are not normalized.

Correct Rounding

Some real numbers are not representable as floating-point values.
E.g. $\frac{1}{3}$ for $\beta = 2$, $\sqrt{2}$ for any β.

Example (Radix, precision, and rounding direction)

In radix 10 and precision 4, π is rounded to:

- $3141 \cdot 10^{-3}$ when rounding toward $-\infty$ or zero,
- $3142 \cdot 10^{-3}$ when rounding toward $+\infty$ or to nearest.

Correct Rounding

Some real numbers are not representable as floating-point values.
E.g. $\frac{1}{3}$ for $\beta = 2$, $\sqrt{2}$ for any β.

Example (Radix, precision, and rounding direction)

In radix 10 and precision 4, π is rounded to:

- $3141 \cdot 10^{-3}$ when rounding toward $-\infty$ or zero,
- $3142 \cdot 10^{-3}$ when rounding toward $+\infty$ or to nearest.

Direction and precision are parameters of FP operations:

$$\text{Fadd } \beta \texttt{ rnd\_UP } 500 : \mathbb{F}_\beta \to \mathbb{F}_\beta \to \mathbb{F}_\beta.$$

No information about exactness (unlike MPFR). Add it?

Coq's Standard Real Numbers

Definition (Coq's real numbers)

- Abstract type with axiomatized total functions.
- Totally ordered (comparison in `Set`).
- Complete (existence of the supremum in `Set`).
- Archimedean (ceil function in `Set`).

No computational content.

Coq's Standard Real Numbers

Definition (Coq's real numbers)

- Abstract type with axiomatized total functions.
- Totally ordered (comparison in `Set`).
- Complete (existence of the supremum in `Set`).
- Archimedean (ceil function in `Set`).

No computational content.

Example (Divide by zero)

Nothing is known about 0^{-1}, except that is a real.
Therefore $0/0 = 0 \times 0^{-1} = 0$.

Extended Real Numbers

Definition (Extended real numbers)

$\overline{\mathbb{R}} = \mathbb{R} \cup \{\bot\}$.

Arithmetic functions are still total,
but they now return $\bot$ for undefined results, e.g. $0^{-1} = \bot$.

Extended Real Numbers

Definition (Extended real numbers)

$\overline{\mathbb{R}} = \mathbb{R} \cup \{\bot\}$.

Arithmetic functions are still total,
but they now return $\bot$ for undefined results, e.g. $0^{-1} = \bot$.

Example (Differentiability)

For any function $f : \overline{\mathbb{R}} \to \overline{\mathbb{R}}$, there are derivative functions $f' : \overline{\mathbb{R}} \to \overline{\mathbb{R}}$ such that for all $x \in \mathbb{R}$, $f'(x) \neq \bot$ implies that f is real on a neighborhood of x and differentiable at x with value $f'(x)$.

Intervals with Floating-Point Bounds

Definition (Interval)

An interval is a closed connected subset of the real numbers, or a Not-an-Interval $\bot_{\mathbb{I}}$.

Some real intervals can be represented as pairs of FP numbers, with $\bot$ acting as an infinity:

- $[\bot, \bot] = \mathbb{R}$,
- $[a, \bot] = \{x \in \mathbb{R} \mid a \leq x\}$,
- $[\bot, b] = \{x \in \mathbb{R} \mid x \leq b\}$,
- $[a, b] = \{x \in \mathbb{R} \mid a \leq x \leq b\}$,
- $\bot_{\mathbb{I}} = \mathbb{R} \cup \{\bot\}$.

Inclusion Property

Definition (Inclusion property)

Operator $\diamond_{\mathbb{I}}$ satisfies the inclusion property with respect to $\diamond_{\overline{\mathbb{R}}}$ if

$$\forall X, Y \in \mathbb{I},\ \forall x \in X,\ \forall y \in Y,\ x \diamond_{\overline{\mathbb{R}}} y \in X \diamond_{\mathbb{I}} Y.$$

Reminder: $x \diamond_{\overline{\mathbb{R}}} y = \bot$ when $x \diamond_{\mathbb{R}} y$ is not defined.

Inclusion Property

Definition (Inclusion property)

Operator $\diamond_{\mathbb{I}}$ satisfies the inclusion property with respect to $\diamond_{\overline{\mathbb{R}}}$ if

$$\forall X, Y \in \mathbb{I},\ \forall x \in X,\ \forall y \in Y,\ x \diamond_{\overline{\mathbb{R}}} y \in X \diamond_{\mathbb{I}} Y.$$

Reminder: $x \diamond_{\overline{\mathbb{R}}} y = \bot$ when $x \diamond_{\mathbb{R}} y$ is not defined.

Example (Differentiability)

If $f'_{\mathbb{I}}([a, b])$ is not $\bot_{\mathbb{I}}$, then $f_{\mathbb{R}}$ is defined and derivable on $[a, b]$.

If $f'_{\mathbb{I}}([a, b]) \subseteq [0, \bot]$, then $f_{\mathbb{R}}$ is increasing on $[a, b]$.

Outline

Dependency: Flocq

The Flocq library (Boldo, Melquiond) provides:

- a multi-radix multi-format formalization of FP arithmetic, e.g. floating-point numbers with unbounded exponent range,

Dependency: Flocq

The Flocq library (Boldo, Melquiond) provides:

- a multi-radix multi-format formalization of FP arithmetic, e.g. floating-point numbers with unbounded exponent range,
- a definition of the usual rounding modes and their properties, e.g. $\bigtriangledown(x) \leq x \leq \bigtriangleup(x)$,

Dependency: Flocq

The Flocq library (Boldo, Melquiond) provides:

- a multi-radix multi-format formalization of FP arithmetic, e.g. floating-point numbers with unbounded exponent range,
- a definition of the usual rounding modes and their properties, e.g. $\bigtriangledown(x) \leq x \leq \bigtriangleup(x)$,
- a naive implementation of $\oplus$, $\otimes$, $\oslash$, $\circ(\sqrt{\Box})$ and their correctness proof.

Hierarchy

Layers of Coq.Interval:

1. reference implementation of FP basic operators,
   (instance of Flocq's operators for Coq.Interval's format)
2. specialized versions (e.g. $\beta = 2$) of these operators,
3. interval basic operators, (even radix only)
4. FP elementary functions, (return intervals)
5. interval elementary functions, (no argument reduction)
6. tactic using bisection and automatic differentiation.

Modular Design

Each layer accesses underlying layers through Coq modules.
$\Longrightarrow$ Ability to substitute optimized implementations.

Modular Design

Each layer accesses underlying layers through Coq modules.
$\Longrightarrow$ Ability to substitute optimized implementations.

Example (Floating-point operations)

- An interface (module type) describes FP operations:
  - `FloatOps` is used by `FloatInterval`.
- Several implementations (modules) provides it:
  - `GenericFloat` α: any radix α, standard integers,
  - `SpecificFloat StdZRadix2`: radix 2, standard integers,
  - `SpecificFloat BigIntRadix2`: radix 2, fast integers.

Modular Design

Each layer accesses underlying layers through Coq modules.
$\Longrightarrow$ Ability to substitute optimized implementations.

Example (Floating-point operations)

- An interface (module type) describes FP operations:
  - `FloatOps` is used by `FloatInterval`.
- Several implementations (modules) provides it:
  - `GenericFloat` α: any radix α, standard integers,
  - `SpecificFloat StdZRadix2`: radix 2, standard integers,
  - `SpecificFloat BigIntRadix2`: radix 2, fast integers.

Over-engineered? Probably.

Standard Integers

Coq's standard integers are defined as lists of bits (little-endian) and computations are performed in the logical framework.

- Huge memory footprint (several words per bits).
- Naive arithmetic operators.
- High confidence.

Standard Integers

Coq's standard integers are defined as lists of bits (little-endian) and computations are performed in the logical framework.

- Huge memory footprint (several words per bits).
- Naive arithmetic operators.
- High confidence.

Could we trust the processor/environment/compiler/...
to perform native integer arithmetic instead?

Fast Integers: Native 31-bit Integers

Arithmetic on 31-bit integers has been formally defined in Coq.

- Addition returns an integer and a carry.
- Multiplication returns the high and low parts.
- ...

Fast Integers: Native 31-bit Integers

Arithmetic on 31-bit integers has been formally defined in Coq.

- Addition returns an integer and a carry.
- Multiplication returns the high and low parts.
- ...

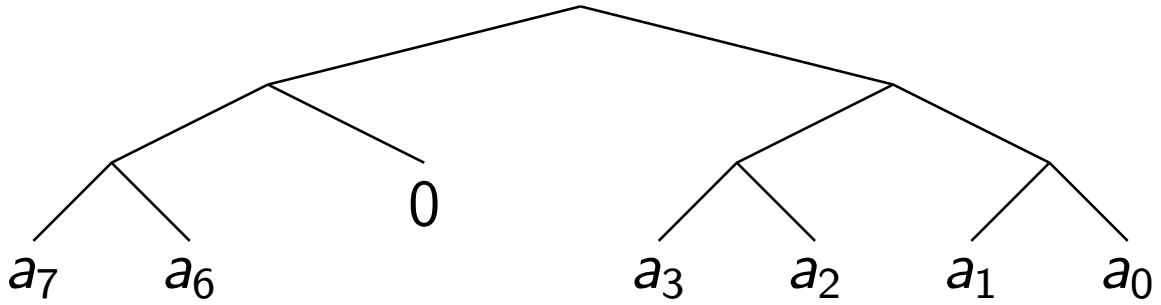
Spiwack has modified the virtual machine to handle them natively.

- Closed-term integers are compressed into machine words (Ocaml-like encoding: LSB is always set).
- Addition, division, rotation, leading-zero count, etc are delegated to the CPU.

Confidence?

Fast Integers: Trees of 31-bit Integers

Grégoire and Théry have defined arbitrary long integers as binary trees with `int31` leaves: $a = \sum_k a_k \cdot 2^{31 \cdot k}$



- Logarithmic complexity for accessing digits.
- Divide-and-conquer algorithms,
  e.g. Karatsuba's multiplication.

Outline

Addition and Multiplication

The set of floating-point numbers $\mathbb{F}_\beta - \{\bot\}$ is a ring for the addition and multiplication on real numbers.

FP addition and multiplication first compute the exact value:

- $(m_1, e_1) + (m_2, e_2) \to (m_1 \cdot \beta^{e_1 - e_2} + m_2, e_2)$ for $e_2 \leq e_1$.
- $(m_1, e_1) \times (m_2, e_2) \to (m_1 \cdot m_2, e_1 + e_2)$.

and then round it to the target precision.

Addition is inefficient when its operands have different magnitudes.

Division and Square Root

Division of (m_1, e_1) by (m_2, e_2):

1. Increase the width of the input: $(m_1', e_1') = (m_1 \cdot \beta^k, e_1 - k)$, so that $\lfloor m_1'/m_2 \rfloor$ has at least p digits.
2. Perform an integer operation on mantissa: $(\lfloor m_1'/m_2 \rfloor, e_1' - e_2)$.
3. Round the number to p digits, according to the sign of $(m_1' - m_2 \cdot \lfloor m_1'/m_2 \rfloor) - m_2/2$.

Division and Square Root

Division of (m_1, e_1) by (m_2, e_2):

1. Increase the width of the input: $(m_1', e_1') = (m_1 \cdot \beta^k, e_1 - k)$, so that $\lfloor m_1'/m_2 \rfloor$ has at least p digits.
2. Perform an integer operation on mantissa: $(\lfloor m_1'/m_2 \rfloor, e_1' - e_2)$.
3. Round the number to p digits, according to the sign of $(m_1' - m_2 \cdot \lfloor m_1'/m_2 \rfloor) - m_2/2$.

Square root of (m, e):

2. Perform an integer operation on mantissa: $(\lfloor \sqrt{m'} \rfloor, e'/2)$.
3. Round according to $(m' - \lfloor \sqrt{m'} \rfloor^2) - (2 \cdot \lfloor \sqrt{m'} \rfloor - 1)/2$.

Elementary Functions

Supported elementary functions: cos, sin, tan, arctan, exp.

No correct rounding:

- Result is an interval enclosing the mathematical value.
- Precision is only a hint for intermediate computations.

Computing Elementary Functions: arctan

Argument reduction until the input x is smaller than $\frac{1}{2}$:

$$\arctan x = \begin{cases} \frac{\pi}{4} + \arctan \frac{x-1}{x+1} & \text{for } x \in [\frac{1}{2}, 2] \\ \frac{\pi}{2} - \arctan \frac{1}{x} & \text{for } x \geq 2 \end{cases}$$

Computing Elementary Functions: arctan

Argument reduction until the input x is smaller than $\frac{1}{2}$:

$$\arctan x = \begin{cases} \frac{\pi}{4} + \arctan \frac{x-1}{x+1} & \text{for } x \in [\frac{1}{2}, 2] \\ \frac{\pi}{2} - \arctan \frac{1}{x} & \text{for } x \geq 2 \end{cases}$$

Evaluation of series until the truncated part ε is negligible:

$$\arctan x = x \cdot (1 - \frac{x^2}{3} + \cdots + \varepsilon) \quad \text{with } |\varepsilon| \leq \frac{x^{2n}}{2n+1} \leq \beta^{-p}$$

by interval arithmetic:

$$\arctan x \in X \cdot (1 - \frac{X^2}{3} + \cdots + [\pm\frac{|X|^{2n}}{2n+1}]) \text{ for } x \in X.$$

Computing Elementary Functions: arctan

Argument reduction until the input x is smaller than $\frac{1}{2}$:

$$\arctan x = \begin{cases} \frac{\pi}{4} + \arctan \frac{x-1}{x+1} & \text{for } x \in [\frac{1}{2}, 2] \\ \frac{\pi}{2} - \arctan \frac{1}{x} & \text{for } x \geq 2 \end{cases}$$

Evaluation of series until the truncated part ε is negligible:

$$\arctan x = x \cdot (1 - \frac{x^2}{3} + \cdots + \varepsilon) \quad \text{with } |\varepsilon| \leq \frac{x^{2n}}{2n+1} \leq \beta^{-p}$$

by interval arithmetic:

$$\arctan x \in X \cdot (1 - \frac{X^2}{3} + \cdots + [\pm\frac{|X|^{2n}}{2n+1}]) \text{ for } x \in X.$$

Evaluation of $\frac{\pi}{4}$ with Machin's formula: $4 \arctan \frac{1}{5} - \arctan \frac{1}{239}$.

Computing Elementary Functions: cos, sin, tan

Argument reduction until the input x is smaller than $\frac{1}{2}$:

$$\begin{cases} \cos x &= 2 \cdot (\cos \frac{x}{2})^2 - 1 \\ \operatorname{sign}(\sin x) &= \operatorname{sign}(\sin \frac{x}{2}) \cdot \operatorname{sign}(\cos \frac{x}{2}) \end{cases}$$

Computing Elementary Functions: cos, sin, tan

Argument reduction until the input x is smaller than $\frac{1}{2}$:

$$\begin{cases} \cos x &= 2 \cdot (\cos \frac{x}{2})^2 - 1 \\ \operatorname{sign}(\sin x) &= \operatorname{sign}(\sin \frac{x}{2}) \cdot \operatorname{sign}(\cos \frac{x}{2}) \end{cases}$$

Result reconstruction:

$$\begin{aligned} \sin x &= \operatorname{sign}(\sin x) \cdot \sqrt{1 - (\cos x)^2} \\ \tan x &= \operatorname{sign}(\sin x) \cdot \operatorname{sign}(\cos x) \cdot \sqrt{(\cos x)^{-2} - 1} \end{aligned}$$

Computing Elementary Functions: cos, sin, tan

Argument reduction until the input x is smaller than $\frac{1}{2}$:

$$\begin{cases} \cos x &= 2 \cdot (\cos \frac{x}{2})^2 - 1 \\ \operatorname{sign}(\sin x) &= \operatorname{sign}(\sin \frac{x}{2}) \cdot \operatorname{sign}(\cos \frac{x}{2}) \end{cases}$$

Result reconstruction:

$$\begin{aligned} \sin x &= \operatorname{sign}(\sin x) \cdot \sqrt{1 - (\cos x)^2} \\ \tan x &= \operatorname{sign}(\sin x) \cdot \operatorname{sign}(\cos x) \cdot \sqrt{(\cos x)^{-2} - 1} \end{aligned}$$

Evaluation of series:

- cos,
- sin if no argument reduction,
- $\tan x = \sin x / \sqrt{1 - \sin^2 x}$ if no argument reduction.

Computing Elementary Functions

- For exp, argument is reduced to $[2^{-8}, 0]$ by inverting/squaring, then alternating series are used.

Computing Elementary Functions

- For exp, argument is reduced to $[2^{-8}, 0]$ by inverting/squaring, then alternating series are used.
- For cos, sin, tan, reducing argument closer to 0 (e.g. 2^{-8}) does not bring any sensible speed up.

Computing Elementary Functions

- For exp, argument is reduced to $[2^{-8}, 0]$ by inverting/squaring, then alternating series are used.
- For cos, sin, tan, reducing argument closer to 0 (e.g. 2^{-8}) does not bring any sensible speed up.

  Using $\mathrm{umc}(2x) = 1 - \cos(2x) = 2 \cdot \mathrm{umc}\, x \cdot (2 - \mathrm{umc}\, x)$ does not help either.

  Why?

Rough Performances (Coq 8.3pl1)

Tests against MPFR 3.0.0 data sets (53-bit radix-2 mantissas).

Naive strategy for correct rounding:

Start with an intermediate precision $p \leftarrow 63$.
While the bounds do not round to the same 53-bit FP number,
retry with an intermediate precision $p \leftarrow p \times 1.5$. (63, 94, 141, 211, ...)

Rough Performances (Coq 8.3pl1)

Tests against MPFR 3.0.0 data sets (53-bit radix-2 mantissas).

Naive strategy for correct rounding:

Start with an intermediate precision $p \leftarrow 63$.
While the bounds do not round to the same 53-bit FP number,
retry with an intermediate precision $p \leftarrow p \times 1.5$. (63, 94, 141, 211, ...)

Time for passing the testsuite:

	Tests	MPFR	Coq fast	Coq std
exp	1330	10.2 ms	×890	×4.9
cos	1418	10.6 ms	×1200	×5.5
sin	1257	10.1 ms	×1400	×6.4
tan	1596	15.7 ms	×1200	×7.1
arctan	870	17.9 ms	×320	×3.7

Rough Performances (Coq 8.3pl1)

Tests against MPFR 3.0.0 data sets (53-bit radix-2 mantissas).

Naive strategy for correct rounding:

Start with an intermediate precision $p \leftarrow 63$.
While the bounds do not round to the same 53-bit FP number,
retry with an intermediate precision $p \leftarrow p \times 1.5$. (63, 94, 141, 211, ...)

Time for passing the testsuite:

	Tests	MPFR	Coq fast	Coq std
exp	1330	10.2 ms	×890	×4.9
cos	1418	10.6 ms	×1200	×5.5
sin	1257	10.1 ms	×1400	×6.4
tan	1596	15.7 ms	×1200	×7.1
arctan	870	17.9 ms	×320	×3.7

Why aren't Coq standard integers much slower?

Outline

Enclosures of Real-Valued Expressions

Definition (Real-valued expression)

Straight-line program of

- variables: x, y, ...
- unary operators: $-\Box$, $\sqrt{\Box}$, $|\Box|$, $\Box^2$, $\Box^{-1}$,
- binary operators: $+$, $-$, $\times$, $\div$,
- transcendental functions: cos, sin, tan, arctan.

Straight-line programs are directed acyclic graphs;
Common sub-expressions are shared and evaluated only once.

Enclosures of Real-Valued Expressions

Definition (Expression enclosures)

$$a \leq f(x, y, \ldots) \leq b$$

with a and b floating-point numbers ($m \cdot 2^e$; $m, e \in \mathbb{Z}$) or $\pm\infty$.

(Note: Undefined values, e.g. $1/0$, are not bounded.)

Goal: Automatically prove the following proposition

$$\frac{a_x \leq x \leq b_x \quad a_y \leq y \leq b_y \quad \ldots}{a \leq f(x, y, \ldots) \leq b}$$

Interval Arithmetic

Theorem (Containment)

Given a SLP *prog* defined on expressions *inputs*,
if $\forall i,\ input_i \in range_i$, then

$$\text{eval}(prog, inputs) \in \text{eval}(prog, ranges)$$

Interval Arithmetic

Theorem (Containment)

Given a SLP *prog* defined on expressions *inputs*,
if $\forall i,\ input_i \in range_i$, then

$$\text{eval}(prog, inputs) \in \text{eval}(prog, ranges)$$

Theorem (Containment 2)

Given a SLP *prog* defined on expressions *inputs*,
if $\forall i,\ input_i \in range_i$, then

$$\texttt{subset}\ (\texttt{eval}\ prog\ ranges)\ output = true \Rightarrow$$
$$\texttt{eval}\ prog\ inputs \in output$$

Purely computational approach.

Sharpening Intervals

Naive interval arithmetic does not keep tracks of correlations due to binary operators:

For $x \in [0,1]$, $\quad x + (-x) \in [0,1] + [-1,0] = [-1,1]$.

Sharpening Intervals

Naive interval arithmetic does not keep tracks of correlations due to binary operators:

For $x \in [0,1]$, $\quad x + (-x) \in [0,1] + [-1,0] = [-1,1]$.

Implemented improvements:

1. Bisection:
   Recursively split the domain into smaller sub-domains, until the proposition is proved on all the sub-domains.

Sharpening Intervals

Naive interval arithmetic does not keep tracks of correlations due to binary operators:

For $x \in [0,1]$, $\quad x + (-x) \in [0,1] + [-1,0] = [-1,1]$.

Implemented improvements:

1. Bisection:
   Recursively split the domain into smaller sub-domains, until the proposition is proved on all the sub-domains.
2. First-order approximation:
   $f(x_0 + h) = f(x_0) + h \cdot f'(x_0 + \xi)$ with $\xi \in [0, h]$ (or $[h, 0]$),
   so $\forall x \in X, \quad f(x) \in f(x_0) + (X - x_0) \cdot f'(X)$ with $x_0 \in X$.

First-Order Approximation

Enclosure $Y' = f'(X)$ is obtained by changing operators in `eval`:

- $-\langle X, X' \rangle = \langle -X, -X' \rangle$,
- $\langle X_1, X_1' \rangle \times \langle X_2, X_2' \rangle = \langle X_1 \times X_2, X_1' \times X_2 + X_1 \times X_2' \rangle$,
- $\tan\langle X, X' \rangle = \langle Y, X' \times (1 + Y^2) \rangle$ with $Y = \tan X$,
- ...

First-Order Approximation

Enclosure $Y' = f'(X)$ is obtained by changing operators in `eval`:

- $-\langle X, X' \rangle = \langle -X, -X' \rangle$,
- $\langle X_1, X_1' \rangle \times \langle X_2, X_2' \rangle = \langle X_1 \times X_2, X_1' \times X_2 + X_1 \times X_2' \rangle$,
- $\tan\langle X, X' \rangle = \langle Y, X' \times (1 + Y^2) \rangle$ with $Y = \tan X$,
- ...

Could it be adapted to higher-order differentials?
If so, is it worth it?

Example: Method Error

Global positioning requires knowing the local radius of Earth:

$$r_p(\phi) = \frac{a}{\sqrt{1 + (1 - f)^2 \cdot \tan^2 \phi}}$$

This can be approximated by a degree-5 polynomial P with single-precision coefficients: $P(\phi_m^2 - \phi^2) = \tilde{r_p}(\phi)$.

$$P(x) = \frac{4439091}{4} + x \cdot \left(\frac{9023647}{4} + x \cdot \left(\cdots \frac{6661427}{131072} \right) \right)$$

Example: Method Error

Global positioning requires knowing the local radius of Earth:

$$r_p(\phi) = \frac{a}{\sqrt{1 + (1-f)^2 \cdot \tan^2 \phi}}$$

This can be approximated by a degree-5 polynomial P with single-precision coefficients: $P(\phi_m^2 - \phi^2) = \tilde{r_p}(\phi)$.

$$P(x) = \frac{4439091}{4} + x \cdot \left(\frac{9023647}{4} + x \cdot \left(\cdots \frac{6661427}{131072} \right) \right)$$

Lemma (Relative method error)

$\left| \frac{r_p(\phi) - \tilde{r_p}(\phi)}{r_p(\phi)} \right| \leq 23 \cdot 2^{-24}$ *when* $0 \leq \phi \leq \phi_m = \frac{715}{512}$.

Example: Method Error

Inequality proved with order-1 Taylor interval approximations on $\sim 10^5$ sub-intervals of $[0, \phi_m]$.

Example: Method Error

Inequality proved with order-1 Taylor interval approximations on $\sim 10^5$ sub-intervals of $[0, \phi_m]$.

1. Daumas, Melquiond, Muñoz, in PVS (2005).

   An oracle chooses the best sub-intervals and
   it generates one PVS script per sub-interval.

   Verification: several hours on a 48-core parallel computer.

Example: Method Error

Inequality proved with order-1 Taylor interval approximations on $\sim 10^5$ sub-intervals of $[0, \phi_m]$.

1. Daumas, Melquiond, Muñoz, in PVS (2005).

   An oracle chooses the best sub-intervals and
   it generates one PVS script per sub-interval.

   Verification: several hours on a 48-core parallel computer.

2. In Coq (2008), a few minutes on a laptop computer.

   Differences:

   - $\ominus$ no oracle (hence some useless computations),
   - $\oplus$ floating-point arithmetic instead of rationals.

Example: Polynomial Approximation

```
Require Import Reals.
Require Import Interval_tactic.
Open Local Scope R_scope.

Definition a := 6378137.
Definition f := 1000000000/298257223563.
Definition umf2 := (1 - f)².
Definition max := 715/512.
Definition rp phi := a / sqrt (1 + umf2 * (tan phi)²).
Definition arp phi :=
  let x := max² - phi² in
  4439091/4 + x * (9023647/4 + x * (
    13868737/64 + x * (13233647/2048 + x * (
      -1898597/16384 + x * (-6661427/131072))))).

Goal
  forall phi, 0 <= phi <= max ->
  Rabs ((rp phi - arp phi) / rp phi) <= 23/16777216.
Proof.
  unfold rp, arp, umf2, a, f, max. intros.
  Time interval with (i_bisect_diff phi).  (* 80s *)
Time Qed.                                  (*  0s *)
```

Tactic parameters:

- bisection and order-1 evaluation on ϕ,
- minimal relative width of intervals: 2^{-15},
- floating-point precision: 30 bits.

Outline

Library Status

- Fully proved:
  - generic FP kernel,
  - tactic algorithm (except $|\cdot|'$ and arctan′),
  - interval kernel,
  - argument reduction for elementary FP functions.
- Partly proved:
  - specialized FP kernels,
  - interval elementary functions.
- Not proved at all:
  - FP series evaluation.

Perspectives / Wish List

- A generic process for evaluating alternating series.
  If possible, no redundant computations.

Perspectives / Wish List

- A generic process for evaluating alternating series.
  If possible, no redundant computations.
- Additive argument reduction for elementary functions?

Perspectives / Wish List

- A generic process for evaluating alternating series.
  If possible, no redundant computations.
- Additive argument reduction for elementary functions?
- A tactic using "higher-order" models.

Perspectives / Wish List

- A generic process for evaluating alternating series.
  If possible, no redundant computations.
- Additive argument reduction for elementary functions?
- A tactic using "higher-order" models.
- A tactic for multivariate problems.

Questions?

http://www.lri.fr/~melquion/soft/coq-interval/