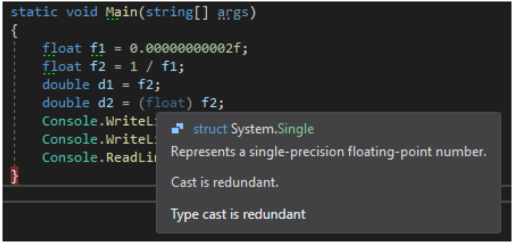
Type Systems for Numerical Error Analysis

Justin Hsu

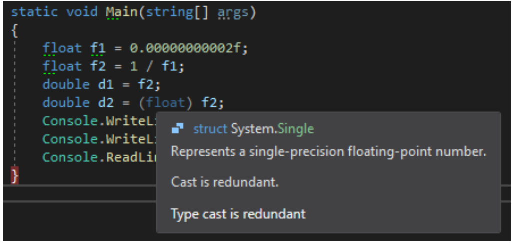
Cornell University

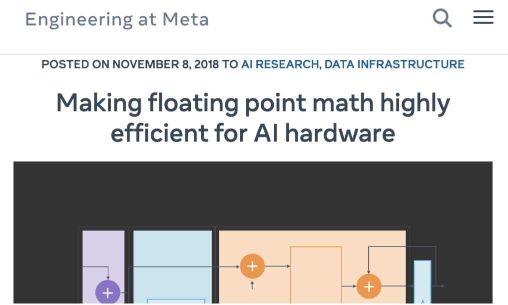
Floating Point (FP) Arithmetic Is Everywhere

Floating Point (FP) Arithmetic Is Everywhere

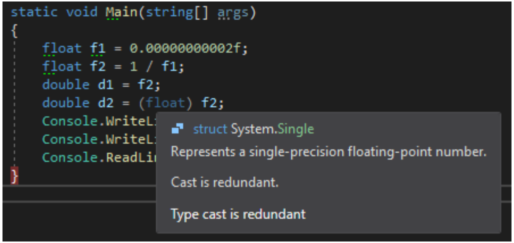


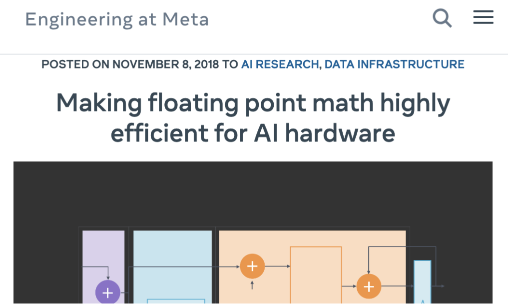
Floating Point (FP) Arithmetic Is Everywhere

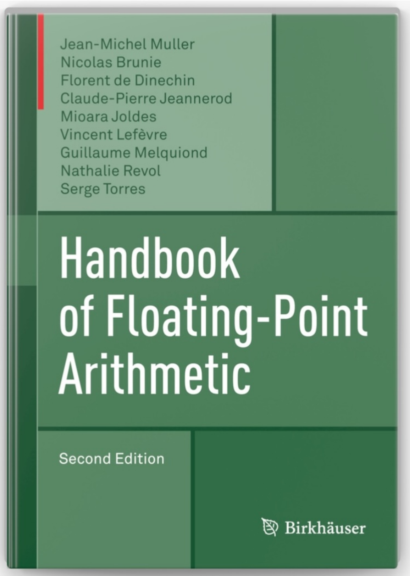




Floating Point (FP) Arithmetic Is Everywhere

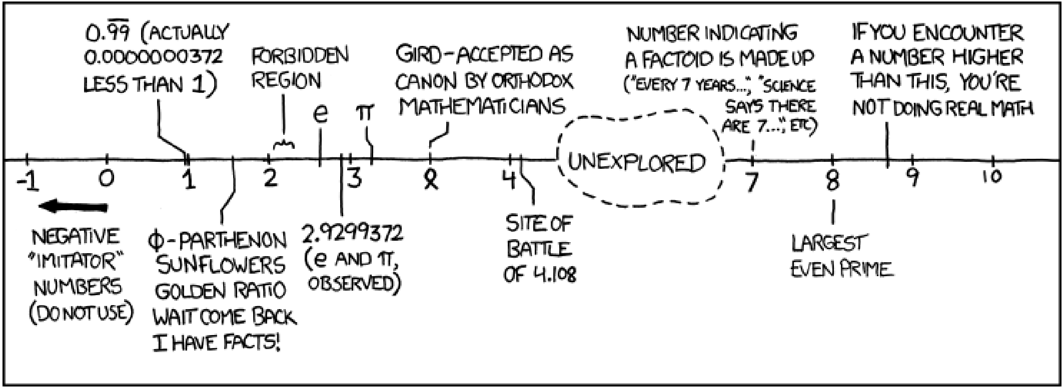






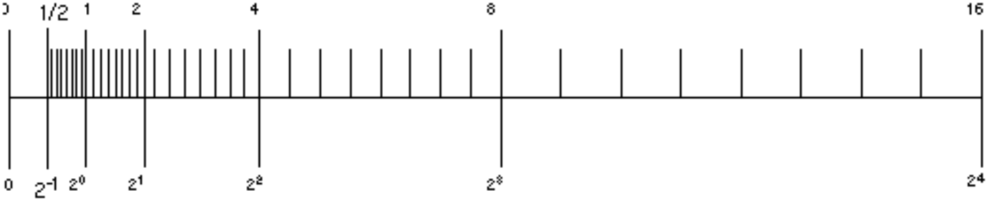
What Do the Floating Point Numbers Look Like?

What Do the Floating Point Numbers Look Like?



— https://xkcd.com/899

What the Floating Point Numbers Actually Look Like



— Sun Microsystems, Inc.

FP Computations Have Roundoff Error

Can only represent finitely many numbers: $\mathbb{F} \subseteq \mathbb{R}$

- Number of representable reals depends on precision (double, float, half, etc.)
- FP arithmetic operations must round to represent result

FP versions of standard arithmetic operations satisfy:

$$a \oplus_{\mathbb{F}} b = (1 + \delta) \cdot (a \oplus_{\mathbb{R}} b) + \epsilon \qquad a \otimes_{\mathbb{F}} b = (1 + \delta) \cdot (a \otimes_{\mathbb{R}} b) + \epsilon$$

Parameters δ, ϵ can depend on a, b, but are bounded by a constant

Problem: Can We Statically Bound the Amount of Roundoff Error?

Helpful to numerical programmers

- Provide guidance on how much precision is needed
- Identify sources of error, reason about error propagation

Problem: Can We Statically Bound the Amount of Roundoff Error?

Helpful to numerical programmers

- Provide guidance on how much precision is needed
- Identify sources of error, reason about error propagation

Long history of verification methods

- Abstract interpretation, interval arithmetic (e.g., Gappa, PRECiSA)
- SMT-based approaches (e.g., Daisy, Rosa)
- Global optimization, semi-definite programming (e.g., FPTaylor, Real2Float)
- Interactive theorem proving (e.g., Isabelle/HOL, VCFloat)

Three Main Challenges

Three Main Challenges

Scalability

- Precise reasoning about floating-point error is expensive
- SMT and optimization-based approaches: hours for 10s of operations

Three Main Challenges

Scalability

- Precise reasoning about floating-point error is expensive
- SMT and optimization-based approaches: hours for 10s of operations

Accuracy

- Bounds on error are more useful if they are tight: not too conservative
- Interval-arithmetic approaches scalable, but often loose bounds

Three Main Challenges

Scalability

- Precise reasoning about floating-point error is expensive
- SMT and optimization-based approaches: hours for 10s of operations

Accuracy

- Bounds on error are more useful if they are tight: not too conservative
- Interval-arithmetic approaches scalable, but often loose bounds

Expressiveness

- Most tools support absolute error, but relative error is more natural
- Target programs are limited: often straight-line programs

Today: the NumFuzz Type System for Bounding FP Error

- Goal: Forward Error Analysis
- Ingredient 1: Sensitivity Analysis
- Ingredient 2: Error Analysis

Joint Work with Excellent Coauthors!

Ariel Kellison

Laura Zielinski

David Bindel

Forward Error Analysis:

A Quick Introduction

Goal: Bound the Distance between Ideal and Approximate

A given program P can be executed ideally or approximately (FP)

- For program P, define ideal $[\![P]\!]_{id}$ and approximate (FP) $[\![P]\!]_{fp}$ semantics
- Example: $[\![x \oplus y]\!]_{id}$ is exact (real) addition, while $[\![x \oplus y]\!]_{fp}$ is FP addition

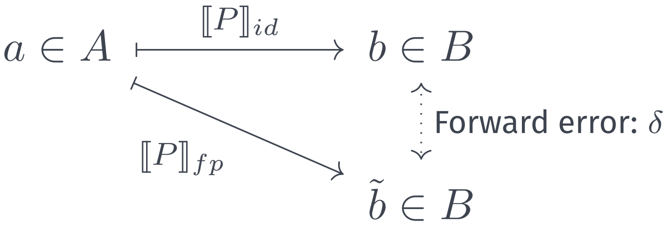
Forward error: maximum distance between $[\![P]\!]_{id}$ and $[\![P]\!]_{fp}$ on same input

Goal: Bound the Distance between Ideal and Approximate

A given program P can be executed ideally or approximately (FP)

- For program P, define ideal $[\![P]\!]_{id}$ and approximate (FP) $[\![P]\!]_{fp}$ semantics
- Example: $[\![x \oplus y]\!]_{id}$ is exact (real) addition, while $[\![x \oplus y]\!]_{fp}$ is FP addition

Forward error: maximum distance between $[\![P]\!]_{id}$ and $[\![P]\!]_{fp}$ on same input



Question 1: How Is Error Introduced?

Not all operations introduce floating-point error

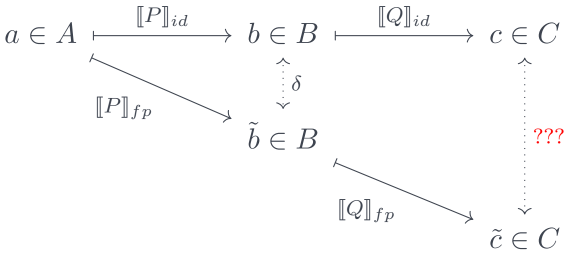
- Primitive arithmetic operations: introduce floating-point error
- Other “regular” operations (assignment, pairing, projection, etc.): no FP error

Error-producing operations depend on application, compiler, hardware, ...

- Example: multiply-add versus fused multiply-add
- Want flexibility to model different kinds of error-producing operations

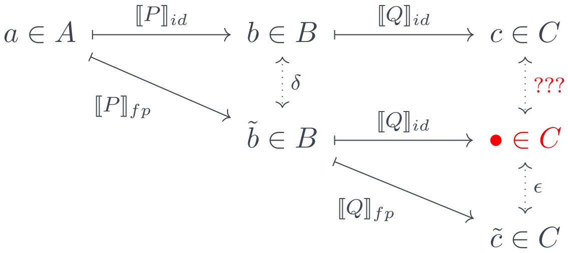
Question 2: How Do Error Bounds Compose?

Example: P has forward error δ and Q has forward error ϵ



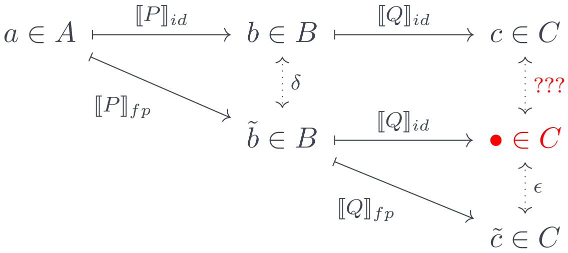
Question 2: How Do Error Bounds Compose?

Example: P has forward error δ and Q has forward error ϵ



Question 2: How Do Error Bounds Compose?

Example: P has forward error δ and Q has forward error ϵ



Bound error by showing Lipschitz guarantee for ideal behavior $[\![Q]\!]_{id}$

Ingredient 1: Sensitivity Analysis

Fuzz: A Linear Type System for Sensitivity Analysis

Fuzz: A Linear Type System for Sensitivity Analysis

A functional programming language

- Lambda calculus with pairs, enums, functions, lists, recursive datatypes, etc.
- Support for higher-order functions and patterns (e.g., maps, folds)

Fuzz: A Linear Type System for Sensitivity Analysis

A functional programming language

- Lambda calculus with pairs, enums, functions, lists, recursive datatypes, etc.
- Support for higher-order functions and patterns (e.g., maps, folds)

A linear type system based on Bounded Linear Logic

- Each type is equipped with a metric
- Type system tracks sensitivity of each variable via number of uses

Fuzz: A Linear Type System for Sensitivity Analysis

A functional programming language

- Lambda calculus with pairs, enums, functions, lists, recursive datatypes, etc.
- Support for higher-order functions and patterns (e.g., maps, folds)

A linear type system based on Bounded Linear Logic

- Each type is equipped with a metric
- Type system tracks sensitivity of each variable via number of uses

Originally: verifying differential privacy (Reed and Pierce, 2010)

- Later: other notions of privacy, generalizing to effects and "coeffects", etc.
- Efficient typechecking (linear in size of program), few annotations required

Example: Numeric Types

Example: Numeric Types

Numbers under absolute distance $\mathbf{num}_{abs}$

- Carrier set: elements of $\mathbf{num}_{abs}$ drawn from real numbers $\mathbb{R}$
- Metric: standard distance $d(a, b) \triangleq |a - b|$

Example: Numeric Types

Numbers under absolute distance $\mathbf{num}_{abs}$

- Carrier set: elements of $\mathbf{num}_{abs}$ drawn from real numbers $\mathbb{R}$
- Metric: standard distance $d(a, b) \triangleq |a - b|$

Numbers under relative distance $\mathbf{num}_{rel}$

- Carrier set: elements of $\mathbf{num}_{rel}$ drawn from non-negative reals $\mathbb{R}^+$
- Metric: relative distance $d(a, b) \triangleq |ln(a) - ln(b)| = |ln(a/b)|$
- Known as the relative precision (RP) distance (Olver, 1978)

The RP Distance: A Closer Look

What does RP measure?

RP distance at most ϵ means points differ by at most $\exp(\epsilon) \approx (1+\epsilon)$ factor:

$$RP(a,b) \leq \epsilon \iff |ln(a/b)| \leq \epsilon \iff \exp(-\epsilon) \leq a/b \leq \exp(\epsilon)$$

The RP Distance: A Closer Look

What does RP measure?

RP distance at most ϵ means points differ by at most $\exp(\epsilon) \approx (1+\epsilon)$ factor:

$$RP(a,b) \leq \epsilon \iff |ln(a/b)| \leq \epsilon \iff \exp(-\epsilon) \leq a/b \leq \exp(\epsilon)$$

Why does RP involve logarithms?

RP distance satisfies the triangle inequality:

The RP Distance: A Closer Look

What does RP measure?

RP distance at most ϵ means points differ by at most $\exp(\epsilon) \approx (1+\epsilon)$ factor:

$$RP(a,b) \leq \epsilon \iff |ln(a/b)| \leq \epsilon \iff \exp(-\epsilon) \leq a/b \leq \exp(\epsilon)$$

Why does RP involve logarithms?

RP distance satisfies the triangle inequality:

- Given: $RP(a,b) \leq \epsilon$ and $RP(b,c) \leq \delta$

The RP Distance: A Closer Look

What does RP measure?

RP distance at most ϵ means points differ by at most $\exp(\epsilon) \approx (1+\epsilon)$ factor:

$$RP(a,b) \leq \epsilon \iff |ln(a/b)| \leq \epsilon \iff \exp(-\epsilon) \leq a/b \leq \exp(\epsilon)$$

Why does RP involve logarithms?

RP distance satisfies the triangle inequality:

- Given: $RP(a,b) \leq \epsilon$ and $RP(b,c) \leq \delta$
- By definition: $|ln(a) - ln(b)| \leq \epsilon$ and $|ln(b) - ln(c)| \leq \delta$

The RP Distance: A Closer Look

What does RP measure?

RP distance at most ϵ means points differ by at most $\exp(\epsilon) \approx (1+\epsilon)$ factor:

$$RP(a,b) \leq \epsilon \iff |ln(a/b)| \leq \epsilon \iff \exp(-\epsilon) \leq a/b \leq \exp(\epsilon)$$

Why does RP involve logarithms?

RP distance satisfies the triangle inequality:

- Given: $RP(a,b) \leq \epsilon$ and $RP(b,c) \leq \delta$
- By definition: $|ln(a) - ln(b)| \leq \epsilon$ and $|ln(b) - ln(c)| \leq \delta$
- Triangle inequality: $|ln(a) - ln(c)| \leq \epsilon + \delta$

The RP Distance: A Closer Look

What does RP measure?

RP distance at most ϵ means points differ by at most $\exp(\epsilon) \approx (1+\epsilon)$ factor:

$$RP(a,b) \leq \epsilon \iff |ln(a/b)| \leq \epsilon \iff \exp(-\epsilon) \leq a/b \leq \exp(\epsilon)$$

Why does RP involve logarithms?

RP distance satisfies the triangle inequality:

- Given: $RP(a,b) \leq \epsilon$ and $RP(b,c) \leq \delta$
- By definition: $|ln(a) - ln(b)| \leq \epsilon$ and $|ln(b) - ln(c)| \leq \delta$
- Triangle inequality: $|ln(a) - ln(c)| \leq \epsilon + \delta$
- Thus by definition: $RP(a,c) \leq \epsilon + \delta$.

Example: Richer Datatypes

Tuples of numbers

- Sum of distances (L_1 metric): $\mathbf{num} \otimes \cdots \otimes \mathbf{num}$
- Max of distances (L_∞ metric): $\mathbf{num} \,\&\, \cdots \,\&\, \mathbf{num}$

Example: Richer Datatypes

Tuples of numbers

- Sum of distances (L_1 metric): $\mathbf{num} \otimes \cdots \otimes \mathbf{num}$
- Max of distances (L_∞ metric): $\mathbf{num} \,\&\, \cdots \,\&\, \mathbf{num}$

Products and Sums

- Two kinds of products (pairs) $A \otimes B$ and $A \,\&\, B$
- Sums (enums) have type $A + B$ (either an A or a B)

Example: Richer Datatypes

Tuples of numbers

- Sum of distances (L_1 metric): $\mathbf{num} \otimes \cdots \otimes \mathbf{num}$
- Max of distances (L_∞ metric): $\mathbf{num} \,\&\, \cdots \,\&\, \mathbf{num}$

Products and Sums

- Two kinds of products (pairs) $A \otimes B$ and $A \,\&\, B$
- Sums (enums) have type $A + B$ (either an A or a B)

Functions

- (Linear) functions from A to B have type $A \multimap B$
- All linear functions are non-expansive (1-Lipschitz)
- More generally: $!_r A \multimap B$ is type of r-sensitive functions for $r \in \mathbb{R}$ or ∞

Example: Typing Addition

Under absolute metric: take sum of changes in input ($\otimes$)

$$\mathbf{add} : \mathbf{num}_{abs} \otimes \mathbf{num}_{abs} \multimap \mathbf{num}_{abs}$$

Change arguments by ϵ and δ absolute: change result by $\epsilon + \delta$ absolute.

Example: Typing Addition

Under absolute metric: take sum of changes in input ($\otimes$)

$$\mathbf{add} : \mathbf{num}_{abs} \otimes \mathbf{num}_{abs} \multimap \mathbf{num}_{abs}$$

Change arguments by ϵ and δ absolute: change result by $\epsilon + \delta$ absolute.

Under relative precision: take max of changes in input ($\&$)

$$\mathbf{add} : \mathbf{num}_{rel} \& \mathbf{num}_{rel} \multimap \mathbf{num}_{rel}$$

Change arguments by $\exp(\epsilon)$, $\exp(\delta)$ factors: change result by $\exp(\max(\epsilon, \delta))$ factor.

Example: Typing Multiplication

Under absolute metric: not Lipschitz (“sensitivity is ∞”)

$$\mathbf{mul} : !_{\infty}(\mathbf{num}_{abs} \otimes \mathbf{num}_{abs}) \multimap \mathbf{num}_{abs}$$

Change arguments by ϵ and δ absolute: change result by unbounded amount.

Example: Typing Multiplication

Under absolute metric: not Lipschitz (“sensitivity is ∞”)

$$\textbf{mul} : !_{\infty}(\textbf{num}_{abs} \otimes \textbf{num}_{abs}) \multimap \textbf{num}_{abs}$$

Change arguments by ϵ and δ absolute: change result by unbounded amount.

Under relative precision: take sum of changes input ($\otimes$)

$$\textbf{mul} : \textbf{num}_{rel} \otimes \textbf{num}_{rel} \multimap \textbf{num}_{rel}$$

Change arguments by $\exp(\epsilon)$, $\exp(\delta)$ factors: change result by $\exp(\epsilon + \delta)$ factor.

Typing Judgments and Soundness in Fuzz

Judgments record sensitivity with respect to each variable

- Contexts: lists of variables with type and sensitivity $r \in \mathbb{R}$

$$\Gamma = x_1 :_{r_1} A_1, \ldots, x_n :_{r_n} A_n$$

- Judgments: program has a type in a context

$$x_1 :_{r_1} A_1, \ldots, x_n :_{r_n} A_n \vdash e : B$$

Typing Judgments and Soundness in Fuzz

Judgments record sensitivity with respect to each variable

- Contexts: lists of variables with type and sensitivity $r \in \mathbb{R}$

$$\Gamma = x_1 :_{r_1} A_1, \ldots, x_n :_{r_n} A_n$$

- Judgments: program has a type in a context

$$x_1 :_{r_1} A_1, \ldots, x_n :_{r_n} A_n \vdash e : B$$

Soundness theorem (Reed and Pierce, 2010)

Suppose $x :_r A \vdash e(x) : B$. Then for any two values $a_1, a_2 : A$, we have:

$$d_B(e(a_1), e(a_2)) \leq r \cdot d_A(a_1, a_2).$$

In other words, well-typed programs e are r-Lipschitz functions.

Categorical Summary: Sensitivity Analysis

Category $\mathbf{EPMet}$ of Extended Pseudo-metric Spaces

- Extended: metric can assign distance infinity
- Pseudo: don't require reflexivity, distance between distinct points can be zero
- Morphisms: non-expansive ("short") maps

Categorical Summary: Sensitivity Analysis

Category $\mathbf{EPMet}$ of Extended Pseudo-metric Spaces

- Extended: metric can assign distance infinity
- Pseudo: don't require reflexivity, distance between distinct points can be zero
- Morphisms: non-expansive ("short") maps

Good category for linear logic

- Symmetric monoidal closed structure $(\otimes, \multimap)$
- Cartesian structure (not closed), coproducts

Categorical Summary: Sensitivity Analysis

Category $\mathbf{EPMet}$ of Extended Pseudo-metric Spaces

- Extended: metric can assign distance infinity
- Pseudo: don't require reflexivity, distance between distinct points can be zero
- Morphisms: non-expansive ("short") maps

Good category for linear logic

- Symmetric monoidal closed structure $(\otimes, \multimap)$
- Cartesian structure (not closed), coproducts

Graded comonad from scaling

- Functors $!_r : \mathbf{EPMet} \to \mathbf{EPMet}$ take (A, d) to $(A, r \cdot d)$
- $\mathbb{R}^{\geq 0}$-graded exponential comonad (Brunel, Gaboardi, Mazza, Zdancewic 2014)

Ingredient 2:
Error Analysis

From Fuzz to NumFuzz: A Type for Tracking Error

So far: types describe data and metric, but not error

- Goal: extend types with quantitative error bounds
- Get static bounds on amount of roundoff error by inferring types

From Fuzz to NumFuzz: A Type for Tracking Error

So far: types describe data and metric, but not error

- Goal: extend types with quantitative error bounds
- Get static bounds on amount of roundoff error by inferring types

Idea: add a new family of error types $\text{Err}_\delta(A)$

- A is any type, and $\delta \in \mathbb{R}$ is a numeric bound
- Think: pairs $(a, \tilde{a}) : A \times A$ of exact and approximate values, $d_A(a, \tilde{a}) \leq \delta$.

Typing Rules: Introducing Error

Typing Rules: Introducing Error

An ideal computation produces no error

$$\frac{\Gamma \vdash e : A}{\Gamma \vdash \mathbf{ret}(e) : \mathsf{Err}_0(A)}$$

Typing Rules: Introducing Error

An ideal computation produces no error

$$\frac{\Gamma \vdash e : A}{\Gamma \vdash \mathbf{ret}(e) : \mathsf{Err}_0(A)}$$

Rounding operation can generate error

$$\frac{\Gamma \vdash e : \mathbf{num}_{rel}}{\Gamma \vdash \mathbf{rnd}(e) : \mathsf{Err}_u(\mathbf{num}_{rel})}$$

Error parameter u depends on particular setting (precision, rounding mode, etc.).

Sequencing: Key Interaction between Error Types and Sensitivity

Composing two functions

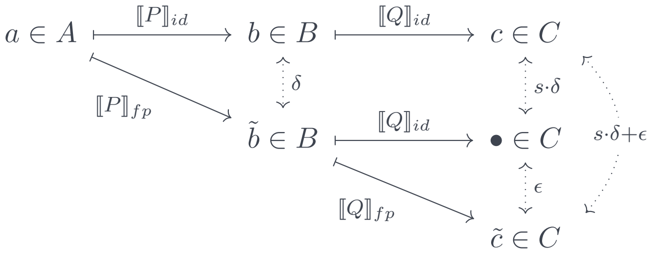
- Program P has forward error δ, and ideal semantics is r-sensitive
- Program Q has forward error ϵ, and ideal semantics is s-sensitive
- Composition $P; Q$ should have forward error $s \cdot \delta + \epsilon$

Sequencing: Key Interaction between Error Types and Sensitivity

Composing two functions

- Program P has forward error δ, and ideal semantics is r-sensitive
- Program Q has forward error ϵ, and ideal semantics is s-sensitive
- Composition $P; Q$ should have forward error $s \cdot \delta + \epsilon$

In pictures



Typing Rule: Sequencing

Assuming that:

- Program P has forward error δ, and ideal semantics is r-sensitive

$$x :_r A \vdash P(x) : \mathsf{Err}_\delta(B)$$

- Program Q has forward error ϵ, and ideal semantics is s-sensitive

$$y :_s B \vdash Q(y) : \mathsf{Err}_\epsilon(C)$$

Typing Rule: Sequencing

Assuming that:

- Program P has forward error δ, and ideal semantics is r-sensitive

$$x :_r A \vdash P(x) : \mathsf{Err}_\delta(B)$$

- Program Q has forward error ϵ, and ideal semantics is s-sensitive

$$y :_s B \vdash Q(y) : \mathsf{Err}_\epsilon(C)$$

Conclude that:

- Composition $P; Q$ should have forward error $s \cdot \delta + \epsilon$

$$x :_{r \cdot s} A \vdash \mathbf{bind}\ y = P(x)\ \mathbf{in}\ Q(y) : \mathsf{Err}_{s \cdot \delta + \epsilon}(C)$$

Interpreting the Error Type: The Graded Neighborhood Monad

Neighborhood Monad: A Graded Monad on $\mathbf{EPMet}$

Grades: $(\mathbb{R}^{\geq 0}, 0, +)$

- Monoid of non-negative real numbers under addition
- Think: upper bound on distance between ideal and approximate

Family of functors: $\{E_r : \mathbf{EPMet} \to \mathbf{EPMet}\}$

- E_r takes (A, d) to metric space of pairs:

$$\{(a, \tilde{a}) \in A \times A \mid d(a, \tilde{a}) \leq r\}$$

  Distance on pairs: distance d between first (ideal) components.
- E_r takes $f : A \to B$ to:

$$E_r(f)(a, \tilde{a}) = (f(a), f(\tilde{a}))$$

  Since f is non-expansive, this is a map from $E_r(A)$ to $E_r(B)$.

Neighborhood Monad: Unit and Multiplication

Graded unit map

- Think: ideal value equal to the approximate value
- Unit map $\eta_A : A \to E_0 A$ defined as:

$$A \ni a \mapsto (a, a) \in E_0 A$$

Neighborhood Monad: Unit and Multiplication

Graded unit map

- Think: ideal value equal to the approximate value
- Unit map $\eta_A : A \to E_0 A$ defined as:

$$A \ni a \mapsto (a, a) \in E_0 A$$

Graded multiplication map

- Think: the "ideal" ideal value, and the "approximate" approximate value
- Graded multiplication map $\mu_{r,s,A} : E_r E_s A \to E_{r+s} A$ defined as:

$$E_r E_s A \ni ((a, \tilde{a}), (b, \tilde{b})) \mapsto (a, \tilde{b}) \in E_{r+s} A$$

Relies crucially on triangle inequality.

Neighborhood Monad: Other Structures

Graded strengths: interaction with products in $\mathbf{EPMet}$

- Maps $st_{r,A} : A \otimes E_r B \to E_r(A \otimes B)$ defined as:

$$A \otimes E_r B \ni (a, (b, \tilde{b})) \mapsto ((a, b), (a, \tilde{b})) \in E_r(A \otimes B)$$

- Similar map for Cartesian product $A \times B$.

Graded distributive law: interaction with scaling comonad

- Key map: $\lambda_{r,s,A} : !_r E_s A \to E_{s \cdot r} !_r A$
- Cf. Gaboardi, Katsumata, Orchard, Breuvart, Uustalu (2016)

NumFuzz:
Example Programs

Example: Arithmetic Operations

IEEE Std 754-2008
IEEE Standard for Floating-Point Arithmetic

5. Operations

5.1 Overview

All conforming implementations of this standard shall provide the operations listed in this clause for all supported arithmetic formats, except as stated below. Each of the computational operations that return a numeric result specified by this standard shall be performed as if it first produced an intermediate result correct to infinite precision and with unbounded range, and then rounded that intermediate result, if necessary, to fit in the destination's format (see 4 and 7). Clause 6 augments the following specifications to cover ±0, ±∞, and NaN. Clause 7 describes default exception handling.

Example: Defining Correctly-Rounded Operations

Example: Defining Correctly-Rounded Operations

Addition

$$\mathbf{addfp}(a, b) \quad \triangleq \quad \mathbf{let}\ z = \mathbf{add}(a, b)\ \mathbf{in}\ \mathbf{rnd}(z)$$

Example: Defining Correctly-Rounded Operations

Addition

$$\mathbf{addfp}(a, b) \triangleq \mathbf{let}\ z = \mathbf{add}(a, b)\ \mathbf{in}\ \mathbf{rnd}(z)$$

Multiplication

$$\mathbf{mulfp}(a, b) \triangleq \mathbf{let}\ z = \mathbf{mul}(a, b)\ \mathbf{in}\ \mathbf{rnd}(z)$$

Example: Defining Correctly-Rounded Operations

Addition

$$\mathbf{addfp}(a, b) \triangleq \mathbf{let}\ z = \mathbf{add}(a, b)\ \mathbf{in}\ \mathbf{rnd}(z)$$

Multiplication

$$\mathbf{mulfp}(a, b) \triangleq \mathbf{let}\ z = \mathbf{mul}(a, b)\ \mathbf{in}\ \mathbf{rnd}(z)$$

Types of FP operations: type of ideal operation, plus rounding

- Upshot: cleanly separate ideal operation from rounding behavior

Example: Multiply-then-add

Compute $a \otimes b \oplus c$ as FP multiply, then FP add

$$\begin{aligned} \mathbf{ma}(a, b, c) \triangleq\ & \mathbf{bind}\ m = \mathbf{mulfp}(a, b)\ \mathbf{in} \\ & \mathbf{bind}\ n = \mathbf{addfp}(m, c)\ \mathbf{in} \\ & \mathbf{ret}(n) \end{aligned}$$

Example: Multiply-then-add

Compute $a \otimes b \oplus c$ as FP multiply, then FP add

$$\begin{aligned} \mathbf{ma}(a, b, c) \triangleq\ & \mathbf{bind}\ m = \mathbf{mulfp}(a, b)\ \mathbf{in} \\ & \mathbf{bind}\ n = \mathbf{addfp}(m, c)\ \mathbf{in} \\ & \mathbf{ret}(n) \end{aligned}$$

Overall type computed from types of FP operations

- As expected: incur error from two rounding operation

Example: Fused multiply-add (FMA)

Compute $a \otimes b \oplus c$: Exact multiply, then exact add, then round

$$\begin{aligned}
\mathbf{fma}(a, b, c) \quad \triangleq \quad & \mathbf{let}\ m = \mathbf{mul}(a, b)\ \mathbf{in} \\
& \mathbf{let}\ n = \mathbf{add}(m, c)\ \mathbf{in} \\
& \mathbf{rnd}(n)
\end{aligned}$$

Example: Fused multiply-add (FMA)

Compute $a \otimes b \oplus c$: Exact multiply, then exact add, then round

$$\begin{aligned}\mathbf{fma}(a, b, c) \triangleq\ & \mathbf{let}\ m = \mathbf{mul}(a, b)\ \mathbf{in} \\ & \mathbf{let}\ n = \mathbf{add}(m, c)\ \mathbf{in} \\ & \mathbf{rnd}(n)\end{aligned}$$

Overall type computed from types of exact operations and round

- As expected: incur error from one rounding operation

Soundness Theorem: the Error Type Bounds the Forward Error

Define two operational semantics: ideal and approximate (FP)

- $e \Downarrow_{id} v$ means: e evaluates to v under ideal semantics
- $e \Downarrow_{fp} v$ means: e evaluates to v under FP semantics

Soundness Theorem: the Error Type Bounds the Forward Error

Define two operational semantics: ideal and approximate (FP)

- $e \Downarrow_{id} v$ means: e evaluates to v under ideal semantics
- $e \Downarrow_{fp} v$ means: e evaluates to v under FP semantics

Theorem (error soundness)

Suppose $\vdash e : \text{Err}_\delta(\mathbf{num})$ is a well-typed program. Then $e \Downarrow_{id} v_{id}$ and $e \Downarrow_{fp} v_{fp}$, and $d_{\mathbf{num}}(v_{id}, v_{fp}) \leq \delta$. Note: holds for $\mathbf{num}_{abs}$ or $\mathbf{num}_{rel}$.

NumFuzz:
Empirical Evaluation

Prototype Implementation of NumFuzz

Build on prior implementations of Fuzz

- Modified an existing OCaml implementation of DFuzz

Requires minimal annotations

- Just need to annotate types of function arguments (but not sensitivities)

Efficient type checking/inference algorithm

- Automatically infers error types $\mathrm{Err}_\delta(A)$, including error bound δ
- Algorithm just involves counting usages, no optimization or SMT

Good Performance for Relative Error on Standard Benchmarks

Benchmark	Ops	Bound			Ratio	Timing (s)		
		Λ_{num}	FPTaylor	Gappa		Λ_{num}	FPTaylor	Gappa
hypot*	4	5.55e-16	5.17e-16	**4.46e-16**	1.3	0.002	3.55	0.069
x_by_xy*	3	4.44e-16	fail	**2.22e-16**	2	0.002	-	0.034
one_by_sqrtxx	3	5.55e-16	5.09e-13	**3.33e-16**	1.7	0.002	3.34	0.047
sqrt_add*	5	9.99e-16	6.66e-16	**5.54e-16**	1.5	0.003	3.28	0.092
test02_sum8*	8	1.55e-15	9.32e-14	1.55e-15	1	0.002	14.61	0.244
nonlin1*	2	4.44e-16	4.49e-16	**2.22e-16**	2	0.003	3.24	0.040
test05_nonlin1*	2	4.44e-16	4.46e-16	**2.22e-16**	2	0.008	3.27	0.042
verhulst*	4	8.88e-16	7.38e-16	**4.44e-16**	2	0.002	3.25	0.069
predatorPrey*	7	1.55e-15	4.21e-11	**8.88e-16**	1.7	0.002	3.28	0.114
test06_sums4_sum1*	4	6.66e-16	6.71e-16	6.66e-16	1	0.003	3.84	0.069
test06_sums4_sum2*	4	6.66e-16	1.78e-14	**4.44e-16**	1.5	0.002	11.02	0.055
i4*	4	4.44e-16	4.50e-16	4.44e-16	1	0.002	3.30	0.055
Horner2	4	4.44e-16	6.49e-11	4.44e-16	1	0.002	11.72	0.052
Horner2_with_error	4	1.55e-15	1.61e-10	**1.11e-15**	1.4	0.002	19.56	0.119
Horner5	10	1.11e-15	1.62e-01	1.11e-15	1	0.003	22.08	0.209
Horner10	20	2.22e-15	1.14e+13	2.22e-15	1	0.003	40.68	0.650
Horner20	40	4.44e-15	2.53e+43	4.44e-15	1	0.003	109.42	2.246

Scales to Large Programs

Benchmark	Ops	Bound (Λ_{num})	Bound (Std.)	Timing (s)	
				Λ_{num}	SATIRE
Horner50[a]	100	1.11e-14	1.11e-14	9e-03	5
MatrixMultiply4	112	1.55e-15	8.88e-16	3e-03	-
Horner75	150	1.66e-14	1.66e-14	2e-02	-
Horner100	200	2.22e-14	2.22e-14	4e-02	-
SerialSum[a]	1023	2.27e-13	2.27e-13	5	5407
Poly50[a]	1325	2.94e-13	-	2.12	3
MatrixMultiply16	7936	6.88e-15	3.55e-15	4e-02	-
MatrixMultiply64[a]	520192	2.82e-14	1.42e-14	10	65
MatrixMultiply128[a]	4177920	5.66e-14	2.84e-14	1080	763

Infers Tight Bounds on Relative Error

Benchmark	Ops	Bound (Λ_{num})	Bound (Std.)	Timing (s)	
				Λ_{num}	SATIRE
Horner50[a]	100	1.11e-14	1.11e-14	9e-03	5
MatrixMultiply4	112	1.55e-15	8.88e-16	3e-03	-
Horner75	150	1.66e-14	1.66e-14	2e-02	-
Horner100	200	2.22e-14	2.22e-14	4e-02	-
SerialSum[a]	1023	2.27e-13	2.27e-13	5	5407
Poly50[a]	1325	2.94e-13	-	2.12	3
MatrixMultiply16	7936	6.88e-15	3.55e-15	4e-02	-
MatrixMultiply64[a]	520192	2.82e-14	1.42e-14	10	65
MatrixMultiply128[a]	4177920	5.66e-14	2.84e-14	1080	763

Current Directions and Conclusions

Backward Error Analysis

More subtle notion of error in numerical analysis (Wilkinson, 1950s)

- Forward error: how much does ideal output differ from approximate output?
- Backward error: is the approximate output exactly correct for a nearby input?

Bean: A Language for Backward Error Analysis (PLDI 2025)

- A mixed linear/non-linear type system for backward error analysis
- Semantics in a novel category of error lenses (cf. bidirectional programming)
- First fully automated analysis for backward error, scales to large programs

More Details in the Papers!

Numerical Fuzz: A Type System for Rounding Error Analysis (PLDI 2024)

- More about the semantics, extensions of error monad to other effects
- Details about implementation, many more benchmarks

Current directions

- Supporting subtraction: not Lipschitz sensitive
- Reasoning about underflows/subnormals? Running-error bounds?
- Neighborhood monad: can we generalize?

Big Picture: Correctness for Numerical Programs

Big Picture: Correctness for Numerical Programs

Numerical programs are hard

- Hard to program: multiple kinds of approximation, stability, performance
- Hard to debug: hard to tell if answers are wrong, hard to localize and fix bugs

Big Picture: Correctness for Numerical Programs

Numerical programs are hard

- Hard to program: multiple kinds of approximation, stability, performance
- Hard to debug: hard to tell if answers are wrong, hard to localize and fix bugs

Numerical programs are interesting

- Besides FP error: truncation, approximation, iteration, Monte Carlo, ...
- Properties with mathematical/geometrical/physical flavor (e.g., conservation)

Big Picture: Correctness for Numerical Programs

Numerical programs are hard

- Hard to program: multiple kinds of approximation, stability, performance
- Hard to debug: hard to tell if answers are wrong, hard to localize and fix bugs

Numerical programs are interesting

- Besides FP error: truncation, approximation, iteration, Monte Carlo, …
- Properties with mathematical/geometrical/physical flavor (e.g., conservation)

Numerical programs are important

- From scientific and physical simulation to digital hardware and arithmetic
- Correctness is key: important decisions depend on these computations
- Performance is critical: large scale systems, limited by time and space

Interested in Learning More?

Athena-Types

Wiser types for numerical analysis.

`https://github.com/Athena-Types`

Type Systems for Numerical Error Analysis

Justin Hsu
Cornell University