RadCal: Design and Theory of Reliable Numerical Programming Languages with First-class Errors

Jurgen J. Vinju
NWO-I Centrum Wiskunde & Informatica
TU Eindhoven

Programming languages use floating point numbers that can behave weirdly,

and users also provide inaccurate inputs…

How to trust the outcomes of numerical software?

What if….

programming languages would implement correct and exact numbers only?

and **what if…**

programming languages would track inaccurate inputs to inaccurate outputs?

Design Elements

- decimal rationals 0.0001123
- decimal repetents 1 / 3 = 0.**(3)**
- midpoint radiuses 5 ± 0.1 == [5-0.1, 5+0.1]
- precision literals ±5.0 == 5±0.05
- **error obliviousness**
- algebraic laws

Definition 1. *A computation $f : \mathbb{M} \to \mathbb{M}$ is "error oblivious" when $\forall m_i \in \mathbb{Q}, r_i \in \mathbb{Q}_\infty$: $f(m_i \pm r_i) = m_o \pm r_o \iff f(m_i) = m_o$; the midpoint of the output of any computation is the same, whether or not the error radiuses are carried along.*

Law/Operator	$+$	$-$	$*$	$/$
Commutativity	$a+b=b+a$		$a*b=b*a$	
Associativity	$a+(b+c)=(a+b)+c$		$a*(b*c)=(a*b)*c$	
Distributivity	$a*(b+c)=a*b+a*c$		$a*(b+c)=a*b+a*c$	
Inversion	$(a+b)-b=a$	$(a-b)+b=a$	$(a*b)/b=a$	$(a/b)*b=a$
Idempotency	$a+0=a$	$a-0=a$	$a*1=a$	$a/1=a$
Division-by-zero				$x/0$ = **undefined**

$$-(a \pm b) = (-a) \pm b \quad \text{(negative-midpoints)}$$
$$a \pm (-b) = a \pm b \quad \text{(absolute-radius)}$$
$$a \pm (b \pm c) = a \pm (b + c) \quad \text{(radius-of-radius)}$$
$$(a \pm b) \pm c = a \pm max(b, c) \quad \text{(radius-on-top-of-radius)}$$
$$(a \pm r) + (b \pm q) = (a + b) \pm (r + q) \quad \text{(addition)}$$
$$(a \pm r) - (b \pm q) = (a - b) \pm (r + q) \quad \text{(subtraction)}$$
$$(a \pm r) * (b \pm q) = (a * b) \pm (r * b + a * q + r * q) \quad \text{(outer-multiplication)}$$
$$(a \pm r) * (b \pm q) = (a * b + r * q) \pm (r * b + a * q) \quad \text{(inner-multiplication)}$$
$$(a \pm r)/(b \pm q) = (a/b) \pm \left(\frac{r + \frac{a}{b} * q}{b - q}\right) \textbf{ when } sign(b-q) = sign(b+q) \quad \text{(division-by-non-zero)}$$
$$(a \pm r)/(b \pm q) = (a/b) \pm \infty \textbf{ when } sign(b-q) \neq sign(b+q) \quad \text{(division-intersects-zero)}$$

```
lexical Digits = [0-9]+ !>> [0-9];

lexical Number
  = Digits whole !>> "."
  | "." Digits part !>> "("
  | Digits whole "." Digits part !>> "("
  | Digits whole "." "(" Digits rep ")"
  | Digits whole "." Digits part "(" Digits rep ")"
  | "." Digits part "(" Digits rep ")"
  | "." "(" Digits rep ")"
  | non-assoc Number base [eE] [+\-]? sign Digits scale
  | non-assoc "±" Number number;

syntax Exp
  = number: Number number
  | projMid: : Exp exp ".mid"
  | projRad: : Exp exp ".rad"
  | neg : "-" Exp exp
  > left radius : Exp mid "±" Exp rad
  > left div : Exp lhs "/" Exp rhs
  > left mul : Exp lhs "*" Exp rhs
  > left (
      add : Exp lhs "+" Exp rhs
    | sub : Exp lhs "-" Exp rhs
   )
  > non-assoc (
      eq : Exp lhs "==" Exp rhs
    | neq : Exp lhs "!=" Exp rhs
    | le : Exp lhs "<=" Exp rhs
    | less : Exp lhs "<" Exp rhs
    | gr : Exp lhs ">" Exp rhs
    | ge : Exp lhs ">=" Exp rhs
    | cmp : Exp lhs "<->" Exp rhs
    | ncmp : Exp lhs ">-<" Exp rhs
   )
  > paren : "(" Exp exp ")";
```

Results

- axiomatized midpoint radius algebra based on rational numbers (fractions)
- readable (in)exact outputs 0.(3) ± 0.1
- unlike floats, RadCal behaves well with proven **associativity** and **commutativity**
- unlike intervals, RadCal has **distributivity** and (weak) **inversion**
- fully automatic accuracy tracking with "reasonably tight" bounds (!)
- ***error refactoring, static error analysis, dynamic error-guided optimizations***, all enabled by **"error obliviousness"** (midpoints are independent of the error estimates)

TODO: efficient implementation for the Rascal metaprogramming language on the JVM using fractions and automatically scaled bigintegers

TODO: evaluation of automated precision tracking on common statistical methods such as Pearson correlation

Disclaimer: RadCal is probably prohibitively slow for supercomputing purposes, and prohibitively expensive for optimally green computing.