Numeric Programs

Guillaume Melquiond

MPRI 2-36-1 “Preuve de Programme”

January 23rd, 2013
Reminder of Previous Lectures

- ML-like programs:
Reminder of Previous Lectures

- ML-like programs:
  - mutable variables,
Reminder of Previous Lectures

- ML-like programs:
  - mutable variables,
  - recursive functions,
Reminder of Previous Lectures

- ML-like programs:
  - mutable variables,
  - recursive functions,
  - basic types: integers, reals,
Reminder of Previous Lectures

- ML-like programs:
  - mutable variables,
  - recursive functions,
  - basic types: integers, reals,
  - arrays,
Reminder of Previous Lectures

- ML-like programs:
  - mutable variables,
  - recursive functions,
  - basic types: integers, reals,
  - arrays,
  - algebraic datatypes.
Reminder of Previous Lectures

- ML-like programs:
  - mutable variables,
  - recursive functions,
  - basic types: integers, reals,
  - arrays,
  - algebraic datatypes.

- Program specification and verification:
Reminder of Previous Lectures

- ML-like programs:
  - mutable variables,
  - recursive functions,
  - basic types: integers, reals,
  - arrays,
  - algebraic datatypes.

- Program specification and verification:
  - Hoare logic: safety, validity, termination,
Reminder of Previous Lectures

- ML-like programs:
  - mutable variables,
  - recursive functions,
  - basic types: integers, reals,
  - arrays,
  - algebraic datatypes.

- Program specification and verification:
  - Hoare logic: safety, validity, termination,
  - weakest precondition computations,
Reminder of Previous Lectures

- **ML-like programs:**
  - mutable variables,
  - recursive functions,
  - basic types: integers, reals,
  - arrays,
  - algebraic datatypes.

- **Program specification and verification:**
  - Hoare logic: safety, validity, termination,
  - weakest precondition computations,
  - modular verification: function contract,
Reminder of Previous Lectures

- ML-like programs:
  - mutable variables,
  - recursive functions,
  - basic types: integers, reals,
  - arrays,
  - algebraic datatypes.

- Program **specification** and verification:
  - Hoare logic: safety, validity, termination,
  - weakest precondition computations,
  - modular verification: function contract,
  - abstract types and axioms,
Reminder of Previous Lectures

- ML-like programs:
  - mutable variables,
  - recursive functions,
  - basic types: integers, reals,
  - arrays,
  - algebraic datatypes.

- Program specification and verification:
  - Hoare logic: safety, validity, termination,
  - weakest precondition computations,
  - modular verification: function contract,
  - abstract types and axioms,
  - logic functions and predicates,
Reminder of Previous Lectures

- **ML-like programs:**
  - mutable variables,
  - recursive functions,
  - basic types: integers, reals,
  - arrays,
  - algebraic datatypes.

- **Program specification and verification:**
  - Hoare logic: safety, validity, termination,
  - weakest precondition computations,
  - modular verification: function contract,
  - abstract types and axioms,
  - logic functions and predicates,
  - recursive and inductive definitions.
Computers and Number Representations

- 32-bit signed integers in two-complement:
  - $1 + 1 \rightarrow 2$
  - $2^{147483647} + 1 \rightarrow -2^{147483648}$
  - $100000_2 \rightarrow 1410065408$
  - $-2^{147483648} \mod -1 \rightarrow$ boom (floating-point exn?!)

- IEEE-754 binary64 floating-point numbers:
  - $2 \times 2 \times \cdots \times 2 \rightarrow +\infty$
  - $1 \div 0 \rightarrow +\infty$
  - $1 \div -0 \rightarrow -\infty$
  - $0 \div 0 \rightarrow \text{NaN}$
Computers and Number Representations

- 32-bit signed integers in two-complement:
  - $1 + 1 \rightarrow 2$, 

- IEEE-754 binary64 floating-point numbers:
  - $2 \times 2 \times \cdots \times 2 \rightarrow +\infty$, 
  - $1 \div 0 \rightarrow +\infty$, 
  - $1 \div -0 \rightarrow -\infty$, 
  - $0 \div 0 \rightarrow \text{NaN}$.
Computers and Number Representations

- 32-bit signed integers in two-complement:
  - $1 + 1 \rightarrow 2$,
  - $2147483647 + 1 \rightarrow -2147483648$, 

IEEE-754 binary64 floating-point numbers:

- $2 \times 2 \times \cdots \times 2 \rightarrow +\infty$,
- $1 \div 0 \rightarrow +\infty$,
- $1 \div -0 \rightarrow -\infty$,
- $0 \div 0 \rightarrow \text{NaN}$. 
32-bit signed integers in two-complement:

- $1 + 1 \rightarrow 2$,
- $2147483647 + 1 \rightarrow -2147483648$,
- $100000^2 \rightarrow 1410065408$, 

IEEE-754 binary64 floating-point numbers:

- $2 \times 2 \times \cdots \times 2 \rightarrow +\infty$,
- $1 \div 0 \rightarrow +\infty$,
- $1 \div -0 \rightarrow -\infty$,
- $0 \div 0 \rightarrow NaN$. 

32-bit signed integers in two-complement:
- $1 + 1 \rightarrow 2$,
- $2147483647 + 1 \rightarrow -2147483648$,
- $100000^2 \rightarrow 1410065408$,
- $-2147483648 \mod -1 \rightarrow$ boom (floating-point exn?!).
Computers and Number Representations

- 32-bit signed integers in two-complement:
  - $1 + 1 \rightarrow 2$,
  - $2147483647 + 1 \rightarrow -2147483648$,
  - $100000^2 \rightarrow 1410065408$,
  - $-2147483648 \mod -1 \rightarrow \text{boom (floating-point exn?!)}.\]

- IEEE-754 binary64 floating-point numbers:

  - $2 \times 2 \times \cdots \times 2 \rightarrow +\infty$,
  - $1 \div 0 \rightarrow +\infty$,
  - $1 \div -0 \rightarrow -\infty$,
  - $0 \div 0 \rightarrow \text{NaN}$.\]
32-bit signed integers in two-complement:
- \( 1 + 1 \rightarrow 2, \)
- \( 2^{16} + 1 \rightarrow -2^{16}, \)
- \( 100000^2 \rightarrow 1410065408, \)
- \(-2^{16} \Mod -1 \rightarrow \text{boom (floating-point exn?!)}.

IEEE-754 binary64 floating-point numbers:
- \( 2 \times 2 \times \cdots \times 2 \rightarrow +\infty, \)
Computers and Number Representations

- 32-bit signed integers in two-complement:
  - $1 + 1 \rightarrow 2$,
  - $2147483647 + 1 \rightarrow -2147483648$,
  - $100000^2 \rightarrow 1410065408$,
  - $-2147483648 \ mod \ -1 \rightarrow \text{boom (floating-point exn?!)}$.

- IEEE-754 binary64 floating-point numbers:
  - $2 \times 2 \times \cdots \times 2 \rightarrow +\infty$,
  - $1 \div 0 \rightarrow +\infty$. 
Computers and Number Representations

- 32-bit signed integers in two-complement:
  - $1 + 1 \rightarrow 2$,
  - $2147483647 + 1 \rightarrow -2147483648$,
  - $100000^2 \rightarrow 1410065408$,
  - $-2147483648 \mod -1 \rightarrow \text{boom (floating-point exn?!)}$.

- IEEE-754 binary64 floating-point numbers:
  - $2 \times 2 \times \cdots \times 2 \rightarrow +\infty$,
  - $1 \div 0 \rightarrow +\infty$,
  - $1 \div -0 \rightarrow -\infty$. 
Computers and Number Representations

- 32-bit signed integers in two-complement:
  - $1 + 1 \rightarrow 2$,
  - $2147483647 + 1 \rightarrow -2147483648$,
  - $100000^2 \rightarrow 1410065408$,
  - $-2147483648 \mod -1 \rightarrow$ boom (floating-point exn?!).

- IEEE-754 binary64 floating-point numbers:
  - $2 \times 2 \times \cdots \times 2 \rightarrow +\infty$,
  - $1 \div 0 \rightarrow +\infty$,
  - $1 \div -0 \rightarrow -\infty$,
  - $0 \div 0 \rightarrow NaN$. 
Some Numerical Failures

- 1983, truncation when computing an index of Vancouver Stock Exchange drops it to half its value after 6 months.
Some Numerical Failures

- 1983, truncation when computing an index of Vancouver Stock Exchange drops it to half its value after 6 months.
- 1987, the inflation in UK is computed with a rounding error: pensions are off by £100M for 21 months.
Some Numerical Failures

- 1983, truncation when computing an index of Vancouver Stock Exchange drops it to half its value after 6 months.
- 1987, the inflation in UK is computed with a rounding error: pensions are off by £100M for 21 months.
- 1991, during Gulf War 1, a Patriot system fails to intercept a Scud missile: 28 casualties.
- 1992, Green Party of Schleswig-Holstein seats in Parliament for a few hours, until a rounding error is discovered.
- 1995, Ariane 5 explodes during its maiden flight due to an overflow: insurance cost is $500M.
- 2007, Excel displays 77,1×850 as 100000.
Some Numerical Failures

- 1983, truncation when computing an index of Vancouver Stock Exchange drops it to half its value after 6 months.
- 1987, the inflation in UK is computed with a rounding error: pensions are off by £100M for 21 months.
- 1991, during Gulf War 1, a Patriot system fails to intercept a Scud missile: 28 casualties.
- 1992, Green Party of Schleswig-Holstein seats in Parliament for a few hours, until a rounding error is discovered.
Some Numerical Failures

▶ 1983, truncation when computing an index of Vancouver Stock Exchange drops it to half its value after 6 months.

▶ 1987, the inflation in UK is computed with a rounding error: pensions are off by £100M for 21 months.

▶ 1991, during Gulf War 1, a Patriot system fails to intercept a Scud missile: 28 casualties.

▶ 1992, Green Party of Schleswig-Holstein seats in Parliament for a few hours, until a rounding error is discovered.

▶ 1995, Ariane 5 explodes during its maiden flight due to an overflow: insurance cost is $500M.
Some Numerical Failures

- 1983, truncation when computing an index of Vancouver Stock Exchange drops it to half its value after 6 months.
- 1987, the inflation in UK is computed with a rounding error: pensions are off by £100M for 21 months.
- 1991, during Gulf War 1, a Patriot system fails to intercept a Scud missile: 28 casualties.
- 1992, Green Party of Schleswig-Holstein seats in Parliament for a few hours, until a rounding error is discovered.
- 1995, Ariane 5 explodes during its maiden flight due to an overflow: insurance cost is $500M.
- 2007, Excel displays $77.1 \times 850$ as 100000.
Some Numerical Failures

1991, during Gulf War 1, a Patriot system fails to intercept a Scud missile: 28 casualties.

Internal clock ticks every 0.1 second.
Time is tracked by fixed-point arith.: \(0.1 \approx 209715 \cdot 2^{-24}\).
Cumulated skew after 24h: \(-0.08\)s, distance: 160m.
System was supposed to be rebooted periodically.
Some Numerical Failures

- 1991, during Gulf War 1, a Patriot system fails to intercept a Scud missile: 28 casualties.
  Internal clock ticks every 0.1 second.
  Time is tracked by fixed-point arith.: $0.1 \approx 209715 \cdot 2^{-24}$.
  Cumulated skew after 24h: $-0.08\text{s}$, distance: 160m.
  System was supposed to be rebooted periodically.

- 2007, Excel displays $77.1 \times 850$ as 100000.
  Bug in binary/decimal conversion.
  Failing inputs: 12 FP numbers.
  Probability to uncover them by random testing: $10^{-18}$. 
Outline

Handling Machine Integers

Floating-Point Computations

Numerical Analysis

Automation

Numerical Algorithms
Binary Search

Exercise: Find appropriate precondition, postcondition, loop invariant, and variant, for this program:

```ocaml
function binary_search(a:map int, n v:int): int
body
  try
    let ref l = 0 in
    let ref u = n - 1 in
    while l ≤ u do
      let m = div (l + u) 2 in
      if a[m] < v then
        l := m + 1
      else if a[m] > v then
        u := m - 1
      else
        raise (Break m)
    done;
    raise Not_found
  with Break i → i
```
Target Type: int32

- 32-bit signed integers in two-complement representation: integers between $-2^{31}$ and $2^{31} - 1$. 

- If the mathematical result of an operation fits in that range, that is the computed result.
- Otherwise, an overflow occurs.
- Behavior depends on language and environment: modulo arith, saturated arith, abrupt termination, etc.

- A program is safe if no overflow occurs.
Target Type: int32

- 32-bit signed integers in two-complement representation: integers between \(-2^{31}\) and \(2^{31} - 1\).

- If the mathematical result of an operation fits in that range, that is the computed result.
Target Type: int32

- 32-bit signed integers in two-complement representation: integers between $-2^{31}$ and $2^{31} - 1$.

- If the **mathematical** result of an operation fits in that range, that is the **computed** result.

- Otherwise, an **overflow** occurs. Behavior depends on language and environment: modulo arith, saturated arith, abrupt termination, etc.
Target Type: int32

- 32-bit signed integers in two-complement representation: integers between $-2^{31}$ and $2^{31} - 1$.

- If the mathematical result of an operation fits in that range, that is the computed result.

- Otherwise, an overflow occurs. Behavior depends on language and environment: modulo arith, saturated arith, abrupt termination, etc.

A program is safe if no overflow occurs.
Idea: replace all arithmetic operations by abstract functions with preconditions. \( x - y \) becomes `int32_sub(x, y)`.

```plaintext
function int32_sub(x: int, y: int): int
  requires -2^{31} \leq x - y < 2^{31}
  ensures result = x - y
```
Safety Checking, Try 2

Idea: replace

- type `int` with an abstract type coercible to it,
- all operations by abstract functions with preconditions,

and add an axiom about the range of `int32`.

```plaintext
type int32
function of_int32(x: int32): int
axiom bounded_int32:
  forall x: int32. \(-2^{31} \leq \text{of\_int32}(x) < 2^{31}\)

function int32_sub(x: int32, y: int32): int32
  requires \(-2^{31} \leq \text{of\_int32}(x) + \text{of\_int32}(y) < 2^{31}\)
  ensures \(\text{of\_int32(result)} = \text{of\_int32}(x) - \text{of\_int32}(y)\)
```
Exercises

1. How to handle int32 constants in programs?

2. How to specify saturating arithmetic?
Outline

Handling Machine Integers

Floating-Point Computations

Numerical Analysis

Automation

Numerical Algorithms
Floating-Point Arithmetic

- Limited range $\Rightarrow$ exceptional behaviors.
- Limited precision $\Rightarrow$ inaccurate results.
Floating-Point Data

IEEE-754 Binary Floating-Point Arithmetic.
Width: $1 + w_e + w_m = 32$, or 64, or 128.
Bias: $2^{w_e-1} - 1$. Precision: $p = w_m + 1$. 
Floating-Point Data

IEEE-754 Binary Floating-Point Arithmetic.
Width: \(1 + w_e + w_m = 32\), or 64, or 128.
Bias: \(2^{w_e-1} - 1\). Precision: \(p = w_m + 1\).

A floating-point datum

<table>
<thead>
<tr>
<th>sign (s)</th>
<th>biased exponent (e') ((w_e) bits)</th>
<th>mantissa (m) ((w_m) bits)</th>
</tr>
</thead>
</table>

represents
Floating-Point Data

IEEE-754 Binary Floating-Point Arithmetic.
Width: $1 + w_e + w_m = 32$, or 64, or 128.
Bias: $2^{w_e-1} - 1$. Precision: $p = w_m + 1$.

A floating-point datum

<table>
<thead>
<tr>
<th>sign $s$</th>
<th>biased exponent $e'$ ($w_e$ bits)</th>
<th>mantissa $m$ ($w_m$ bits)</th>
</tr>
</thead>
</table>

represents

- if $0 < e' < 2^{w_e} - 1$, the real $(-1)^s \cdot 1.m' \cdot 2^{e' - \text{bias}}$, normal
Floating-Point Data

IEEE-754 Binary Floating-Point Arithmetic.
Width: \(1 + w_e + w_m = 32\), or 64, or 128.
Bias: \(2^{w_e-1} - 1\). Precision: \(p = w_m + 1\).

A floating-point datum

<table>
<thead>
<tr>
<th>sign (s)</th>
<th>biased exponent (e') ((w_e) bits)</th>
<th>mantissa (m) ((w_m) bits)</th>
</tr>
</thead>
</table>

represents

- if \(0 < e' < 2^{w_e} - 1\), the real \((-1)^s \cdot 1.m' \cdot 2^{e' - bias}\), normal
- if \(e' = 0\),
  - ±0 if \(m' = 0\), zeros
  - the real \((-1)^s \cdot 0.m' \cdot 2^{-bias+1}\) otherwise, subnormal
- Not-a-Number otherwise. NaN
Floating-Point Data

IEEE-754 Binary Floating-Point Arithmetic.
Width: $1 + w_e + w_m = 32$, or 64, or 128.
Bias: $2^{w_e-1} - 1$. Precision: $p = w_m + 1$.

A floating-point datum

<table>
<thead>
<tr>
<th>sign $s$</th>
<th>biased exponent $e'$ ($w_e$ bits)</th>
<th>mantissa $m$ ($w_m$ bits)</th>
</tr>
</thead>
</table>

represents

- if $0 < e' < 2^{w_e} - 1$, the real $(-1)^s \cdot 1.m' \cdot 2^{e' - \text{bias}}$, normal
- if $e' = 0$,
  - $\pm 0$ if $m' = 0$, zeros
  - the real $(-1)^s \cdot 0.m' \cdot 2^{-\text{bias}+1}$ otherwise, subnormal
- if $e' = 2^{w_e} - 1$,
  - $(-1)^s \cdot \infty$ if $m' = 0$, infinity
  - *Not-a-Number* otherwise.
Floating-Point Data

\[ (-1)^s \times 2^{e-B} \times 1.f \]

\[ (-1)^1 \times 2^{198-127} \times 1.10010011110000111000000_2 \]

\[ -2^{54} \times 206727 \approx -3.7 \times 10^{21} \]
Semantics for the Finite Case

A floating-point operator shall behave as if it was first computing the infinitely-precise value and then rounding it so that it fits in the destination floating-point format.
Semantics for the Finite Case

A floating-point operator shall behave as if it was first computing the infinitely-precise value and then rounding it so that it fits in the destination floating-point format.

Rounding of a real number $x$:

Overflows are not considered when defining rounding: exponents are supposed to have no upper bound!
Partial Specification

Same as with integers, we specify FP operations so that no overflow occurs.

```plaintext
type bin32
function of_bin32(x: bin32): real
axiom finite_bin32: forall x: bin32. ???

function rnd...(x: real): real
axiom about_rnd...: ???

function bin32_sub(x: bin32, y: bin32): bin32
  requires abs(rnd...(of_bin32(x) - of_bin32(y))) ≤ ...
  ensures of_bin32(result) =
    rnd(of_bin32(x) - of_bin32(y))
```
Simplifications

Floating-point numbers as a subset $\mathbb{F}$ of real numbers:

- neither infinities nor NaNs,

Canonical representation:

- either $2^{p-1} \leq |m| < 2^p$ and $e \geq e_{\text{min}}$, normal
- or $|m| < 2^{p-1}$ and $e = e_{\text{min}}$, subnormal
Simplifications

Floating-point numbers as a subset $\mathbb{F}$ of real numbers:

- neither infinities nor NaNs,
- no signed zeros,
Simplifications

Floating-point numbers as a subset $\mathbb{F}$ of real numbers:

► neither infinities nor NaNs,
► no signed zeros,
► no upper bound.
Simplifications

Floating-point numbers as a subset $\mathbb{F}$ of real numbers:

- neither infinities nor NaNs,
- no signed zeros,
- no upper bound.

\[
\mathbb{F} = \{ m \cdot 2^e \in \mathbb{R}; \ |m| < 2^p \land e \geq e_{\text{min}} \}
\]
Simplifications

Floating-point numbers as a subset $\mathbb{F}$ of real numbers:

- neither infinities nor NaNs,
- no signed zeros,
- no upper bound.

$\mathbb{F} = \{ m \cdot 2^e \in \mathbb{R}; |m| < 2^p \land e \geq e_{\text{min}} \}$

Canonical representation:

- either $2^{p-1} \leq |m| < 2^p$ and $e \geq e_{\text{min}}$, normal
- or $|m| < 2^{p-1}$ and $e = e_{\text{min}}$, subnormal
Usual Properties: Representation and Successors

Given a representable number $x = m_x \cdot 2^{e_x} \geq 0$,

1. $y = (m_x + 1) \cdot 2^{e_x} \in \mathbb{F}$,
2. $m_x \cdot 2^{e_x}$ canonic $\Rightarrow \exists z \in \mathbb{F}$, $x < z < y$. 
Usual Properties: Representation and Successors

Given a representable number $x = m_x \cdot 2^{e_x} \geq 0$,

1. $y = (m_x + 1) \cdot 2^{e_x} \in \mathbb{F}$,

2. $m_x \cdot 2^{e_x}$ canonic $\Rightarrow \exists z \in \mathbb{F}, x < z < y$.

Proof:

1. Hyp: $0 \leq m_x < 2^p$ et $e_x \geq e_{\text{min}}$.
   If $|m_x + 1| < 2^p$, then $y = (m_x + 1) \cdot 2^{e_x} \in \mathbb{F}$.
   Otherwise $m_x + 1 = 2^p$, so $y = 1 \cdot 2^{e_x+p} \in \mathbb{F}$. 
Usual Properties: Representation and Successors

Given a representable number $x = m_x \cdot 2^{e_x} \geq 0,$

1. $y = (m_x + 1) \cdot 2^{e_x} \in \mathbb{F},$
2. $m_x \cdot 2^{e_x}$ canonic $\Rightarrow \exists z \in \mathbb{F}, \ x < z < y.$

Proof:

1. Hyp: $0 \leq m_x < 2^p$ et $e_x \geq e_{min}.$
   If $|m_x + 1| < 2^p,$ then $y = (m_x + 1) \cdot 2^{e_x} \in \mathbb{F}.$
   Otherwise $m_x + 1 = 2^p,$ so $y = 1 \cdot 2^{e_x+p} \in \mathbb{F}.$

2. Hyp: $2^{p-1} \leq m_x < 2^p$ or $e_x = e_{min}.$
   If $m_x \cdot 2^{e_x} < m_z \cdot 2^{e_z} < (m_x + 1) \cdot 2^{e_x},$
   then $e_z > e_x$ and $m_z > 2^{e_x-e_z} m_x \geq 2m_x.$
Usual Properties: Rounding Modes

Faithful rounding:

- $\nabla(x) = \max\{y \in F \mid y \leq x\}$,
- $\Delta(x) = \min\{y \in F \mid y \geq x\}$,
- either $\text{rnd}(x) = \nabla(x)$ or $\text{rnd}(x) = \Delta(x)$. 

Idempotency: $\forall x \in F$, $\text{rnd}(\text{rnd}(x)) = \text{rnd}(x)$

Local monotonicity: $\forall x, y \in \mathbb{R}, y \in [\text{rnd}(x), x] \Rightarrow \text{rnd}(y) = \text{rnd}(x)$
Usual Properties: Rounding Modes

Faithful rounding:

- \( \nabla(x) = \max\{y \in \mathbb{F} \mid y \leq x\} \),
- \( \Delta(x) = \min\{y \in \mathbb{F} \mid y \geq x\} \),
- either \( \text{rnd}(x) = \nabla(x) \) or \( \text{rnd}(x) = \Delta(x) \).

Idempotency:

\[ \forall x \in \mathbb{F}, \text{rnd}(x) = x \]
Usual Properties: Rounding Modes

**Faithful rounding:**
- $\nabla(x) = \max\{y \in \mathbb{F} \mid y \leq x\}$,
- $\Delta(x) = \min\{y \in \mathbb{F} \mid y \geq x\}$,
- either $\text{rnd}(x) = \nabla(x)$ or $\text{rnd}(x) = \Delta(x)$.

**Idempotency:**
- $\forall x \in \mathbb{F}, \text{rnd}(x) = x$

**Local monotonicity:**
- $\forall x, y \in \mathbb{R}, y \in [\text{rnd}(x), x] \Rightarrow \text{rnd}(y) = \text{rnd}(x)$
Usual Properties: Monotonicity

∀x, y ∈ ℝ, x ≤ y ⇒ rnd(x) ≤ rnd(y)
Usual Properties: Monotonicity

\[ \forall x, y \in \mathbb{R}, \ x \leq y \Rightarrow \text{rnd}(x) \leq \text{rnd}(y) \]

Proof:

- If \( \nabla(x) < \nabla(y) \),
  1. \( x < \nabla(y) \) by definition of \( \nabla(x) \),
  2. \( \text{rnd}(x) \leq \Delta(x) \leq \nabla(y) \leq \Box(y) \).
Usual Properties: Monotonicity

\[ \forall x, y \in \mathbb{R}, \quad x \leq y \Rightarrow \text{rnd}(x) \leq \text{rnd}(y) \]

Proof:

- If \( \nabla(x) < \nabla(y) \),
  1. \( x < \nabla(y) \) by definition of \( \nabla(x) \),
  2. \( \text{rnd}(x) \leq \Delta(x) \leq \nabla(y) \leq \Box(y) \).

- If \( \nabla(x) \geq \nabla(y) \),
Usual Properties: Monotonicity

\[ \forall x, y \in \mathbb{R}, \, x \leq y \Rightarrow \text{rnd}(x) \leq \text{rnd}(y) \]

Proof:

- If \( \bigtriangledown(x) \leq \bigtriangledown(y) \),
  1. \( x < \bigtriangledown(y) \) by definition of \( \bigtriangledown(x) \),
  2. \( \text{rnd}(x) \leq \Delta(x) \leq \bigtriangledown(y) \leq \square(y) \).

- If \( \bigtriangledown(x) \geq \bigtriangledown(y) \),
  1. \( \bigtriangledown(x) = \bigtriangledown(y) \) by definition of \( \bigtriangledown(y) \),
Usual Properties: Monotonicity

\[ \forall x, y \in \mathbb{R}, \ x \leq y \Rightarrow \text{rnd}(x) \leq \text{rnd}(y) \]

Proof:

- If \( \bigtriangledown(x) < \bigtriangledown(y) \),
  1. \( x < \bigtriangledown(y) \) by definition of \( \bigtriangledown(x) \),
  2. \( \text{rnd}(x) \leq \Delta(x) \leq \bigtriangledown(y) \leq \Box(y) \).

- If \( \bigtriangledown(x) \geq \bigtriangledown(y) \),
  1. \( \bigtriangledown(x) = \bigtriangledown(y) \) by definition of \( \bigtriangledown(y) \),
  2. \( \Delta(x) = \Delta(y) \) by idempotency or successor,
Usual Properties: Monotonicity

\[\forall x, y \in \mathbb{R}, \quad x \leq y \Rightarrow \text{rnd}(x) \leq \text{rnd}(y)\]

Proof:

- If \(\nabla(x) < \nabla(y)\),
  1. \(x < \nabla(y)\) by definition of \(\nabla(x)\),
  2. \(\text{rnd}(x) \leq \Delta(x) \leq \nabla(y) \leq \Box(y)\).

- If \(\nabla(x) \geq \nabla(y)\),
  1. \(\nabla(x) = \nabla(y)\) by definition of \(\nabla(y)\),
  2. \(\Delta(x) = \Delta(y)\) by idempotency or successor,
  3. if \(\text{rnd}(y) = \Delta(y)\), then \(\text{rnd}(x) \leq \text{rnd}(y)\),
Usual Properties: Monotonicity

∀x, y ∈ ℝ, x ≤ y ⇒ rnd(x) ≤ rnd(y)

Proof:

► If ▽(x) < ▽(y),
1. x < ▽(y) by definition of ▽(x),
2. rnd(x) ≤ △(x) ≤ ▽(y) ≤ □(y).

► If ▽(x) ≥ ▽(y),
1. ▽(x) = ▽(y) by definition of ▽(y),
2. △(x) = △(y) by idempotency or successor,
3. if rnd(y) = △(y), then rnd(x) ≤ rnd(y),
4. otherwise rnd(x) = rnd(y) by local monotonicity.
Usual Properties: Monotonicity

Monotonicity:

\[ \forall x, y \in \mathbb{R}, \ x \leq y \Rightarrow \text{rnd}(x) \leq \text{rnd}(y) \]

Ordering with respect to representable numbers:

\[ \forall x \in F, \ \forall y \in \mathbb{R}, \ x \leq y \Rightarrow x \leq \text{rnd}(y) \]
Usual Properties: Round-Off Errors

Rounding to nearest:
For all \( x \in \mathbb{R} \), there are \( \varepsilon \) and \( \delta \) such that

\[
\text{rnd}(x) = x \cdot (1 + \varepsilon) + \delta \quad \text{and} \quad |\varepsilon| \leq 2^{-p} \quad \text{and} \quad |\delta| \leq 2^{e_{\text{min}} - 1}
\]

Moreover, \( \delta = 0 \) or \( \varepsilon = 0 \).
Usual Properties: Round-Off Errors

Rounding to nearest:
For all $x \in \mathbb{F}$, there are $\varepsilon$ and $\delta$ such that

$$\text{rnd}(x) = x \cdot (1 + \varepsilon) + \delta \quad \text{and} \quad |\varepsilon| \leq 2^{-p} \quad \text{and} \quad |\delta| \leq 2^{e_{\text{min}} - 1}$$

Moreover, $\delta = 0$ or $\varepsilon = 0$.

Proof:

1. Hyp: $0 < x \not\in \mathbb{F}$.
   $$\nabla(x) = m \cdot 2^e \quad \text{and} \quad \Delta(x) = (m + 1) \cdot 2^{e+1}.$$
Usual Properties: Round-Off Errors

Rounding to nearest:
For all $x \in \mathbb{F}$, there are $\varepsilon$ and $\delta$ such that

$$\text{rnd}(x) = x \cdot (1 + \varepsilon) + \delta \quad \text{and} \quad |\varepsilon| \leq 2^{-p} \quad \text{and} \quad |\delta| \leq 2^{e_{\min} - 1}$$

Moreover, $\delta = 0$ or $\varepsilon = 0$.

Proof:
1. Hyp: $0 < x \notin \mathbb{F}$.
   $$\nabla(x) = m \cdot 2^e \quad \text{and} \quad \Delta(x) = (m + 1) \cdot 2^{e+1}.$$ 
2. $|\text{rnd}(x) - x| \leq (\Delta(x) - \nabla(x))/2 = 2^{e-1}$. 
Usual Properties: Round-Off Errors

Rounding to nearest:
For all \( x \in \mathbb{F} \), there are \( \varepsilon \) and \( \delta \) such that

\[
\text{rnd}(x) = x \cdot (1 + \varepsilon) + \delta \quad \text{and} \quad |\varepsilon| \leq 2^{-p} \quad \text{and} \quad |\delta| \leq 2^{e_{\text{min}}-1}
\]

Moreover, \( \delta = 0 \) or \( \varepsilon = 0 \).

Proof:

1. Hyp: \( 0 < x \notin \mathbb{F} \).
   \[
   \nabla(x) = m \cdot 2^e \quad \text{and} \quad \Delta(x) = (m+1) \cdot 2^{e+1}.
   \]
2. \( |\text{rnd}(x) - x| \leq (\Delta(x) - \nabla(x))/2 = 2^{e-1} \).
   - If \( \nabla(x) \) is subnormal, \( e = e_{\text{min}} \).
     \( \varepsilon = 0 \) and \( \delta = \text{rnd}(x) - x \) so \( |\delta| \leq 2^{e_{\text{min}}-1} \).
Usual Properties: Round-Off Errors

Rounding to nearest:
For all \( x \in \), there are \( \varepsilon \) and \( \delta \) such that

\[
\text{rnd}(x) = x \cdot (1 + \varepsilon) + \delta \quad \text{and} \quad |\varepsilon| \leq 2^{-p} \quad \text{and} \quad |\delta| \leq 2^{e_{\text{min}}-1}
\]

Moreover, \( \delta = 0 \) or \( \varepsilon = 0 \).

Proof:

1. Hyp: \( 0 < x \not\in \mathbb{F} \).
   \( \nabla(x) = m \cdot 2^e \) and \( \Delta(x) = (m + 1) \cdot 2^{e+1} \).
2. \( |\text{rnd}(x) - x| \leq (\Delta(x) - \nabla(x))/2 = 2^{e-1} \).
   - If \( \nabla(x) \) is subnormal, \( e = e_{\text{min}} \).
     \( \varepsilon = 0 \) and \( \delta = \text{rnd}(x) - x \) so \( |\delta| \leq 2^{e_{\text{min}}-1} \).
   - If \( \nabla(x) \) is normal, \( 2^{p-1} \leq m \).
     \( \delta = 0 \) and \( \varepsilon = (\text{rnd}(x) - x)/x \) so
     \( |\varepsilon| \leq 2^{e-1}/(2^{p-1} \cdot 2^e) = 2^{-p} \).
Usual Properties: Round-Off Errors

Rounding to nearest:
For all $x \in \mathbb{R}$, there are $\varepsilon$ and $\delta$ such that

$$
\text{rnd}(x) = x \cdot (1 + \varepsilon) + \delta \quad \text{and} \quad |\varepsilon| \leq 2^{-p} \quad \text{and} \quad |\delta| \leq 2^{e_{\text{min}} - 1}
$$

Moreover, $\delta = 0$ or $\varepsilon = 0$. 

Usual Properties: Round-Off Errors

Rounding to nearest:
For all \( x \in \), there are \( \varepsilon \) and \( \delta \) such that

\[
\text{rnd}(x) = x \cdot (1 + \varepsilon) + \delta \quad \text{and} \quad |\varepsilon| \leq 2^{-p} \quad \text{and} \quad |\delta| \leq 2^{e_{\text{min}}-1}
\]

Moreover, \( \delta = 0 \) or \( \varepsilon = 0 \).

Directed rounding:
For all \( x \in \), there are \( \varepsilon \) and \( \delta \) such that

\[
\text{rnd}(x) = x \cdot (1 + \varepsilon) + \delta \quad \text{and} \quad |\varepsilon| \leq 2^{-p+1} \quad \text{and} \quad |\delta| \leq 2^{e_{\text{min}}}
\]

Moreover, \( \delta = 0 \) or \( \varepsilon = 0 \).
Usual Properties: Subnormal Addition

Sums in the subnormal range are representable:

\[ \forall x, y \in \mathbb{F}, \quad |x + y| \leq 2^{e_{\min} + p} \Rightarrow x + y \in \mathbb{F} \]
Usual Properties: Subnormal Addition

Sums in the subnormal range are representable:

$$\forall x, y \in \mathbb{F}, \ |x + y| \leq 2^{e_{\text{min}}+p} \Rightarrow x + y \in \mathbb{F}$$

Proof:

1. \(x = m_x \cdot 2^{e_x}\) and \(y = m_y \cdot 2^{e_y}\).
2. \(m = m_x \cdot 2^{e_x-e_{\text{min}}} + m_y \cdot 2^{e_y-e_{\text{min}}}\) and \(x + y = m \cdot 2^{e_{\text{min}}}\).
3. \(|m| \leq 2^p\) so \(x + y \in \mathbb{F}\).
Usual Properties: Subnormal Addition

Sums in the subnormal range are representable:

$$\forall x, y \in \mathbb{F}, \ |x + y| \leq 2^{e_{\text{min}}+p} \Rightarrow x + y \in \mathbb{F}$$

Proof:

1. $x = m_x \cdot 2^{e_x}$ and $y = m_y \cdot 2^{e_y}$.
2. $m = m_x \cdot 2^{e_x-e_{\text{min}}} + m_y \cdot 2^{e_y-e_{\text{min}}}$ and $x + y = m \cdot 2^{e_{\text{min}}}$.
3. $|m| \leq 2^p$ so $x + y \in \mathbb{F}$.

Round-off error for addition:

$$\forall x, y \in \mathbb{F}, \ \exists \varepsilon, \ \circ (x + y) = (x + y) \cdot (1 + \varepsilon) \ \text{and} \ \ |\varepsilon| \leq 2^{-p}$$
Outline

Handling Machine Integers

Floating-Point Computations

Numerical Analysis

Automation

Numerical Algorithms
Numerical Errors

Given two real numbers $u$ and $v$,

- absolute error: $u - v$, $|u - v|$
- relative error: $u/v - 1$, …
Numerical Errors

Given two real numbers $u$ and $v$,

- **absolute error**: $u - v$, $|u - v|$
- **relative error**: $u/v - 1$, …

Combining errors:

- $u - w = (u - v) + (v - w)$
- $|u - w| \leq |u - v| + |v - w|$
- $u/w - 1 = (u/v - 1) + (v/w - 1) + (u/v - 1) \cdot (v/w - 1)$
Numerical Errors

Given two real numbers $u$ and $v$,

- **absolute error**: $u - v$, $|u - v|
- **relative error**: $u / v - 1$, …

Combining errors:

- $u - w = (u - v) + (v - w)$
- $|u - w| \leq |u - v| + |v - w|
- $u / w - 1 = (u / v - 1) + (v / w - 1) + (u / v - 1) \cdot (v / w - 1)$

Remark: $\text{rnd}(u) - v = (\text{rnd}(u) - u) + (u - v)$
Numerical Analysis

Notations:
- a mathematical function $f(x)$,
- a floating-point program $	ilde{f}(x)$,
- the infinitely-precise evaluation $\hat{f}(x)$ of $\tilde{f}(x)$.
Numerical Analysis

Notations:
- a mathematical function $f(x)$,
- a floating-point program $\tilde{f}(x)$,
- the infinitely-precise evaluation $\hat{f}(x)$ of $\tilde{f}(x)$.

Definitions:
- **forward error**: $\tilde{f}(x) - f(x)$,
  - **round-off error**: $\tilde{f}(x) - \hat{f}(x)$
  - **method error**: $\hat{f}(x) - f(x)$

Remark: $\tilde{f}(x) - f(x) \simeq (\tilde{x} - x) \times \left| \frac{\partial f}{\partial x} \right|$.
In other words: \text{forward error} \simeq \text{backward error} \times \text{condition number}.
Numerical Analysis

Notations:
- a mathematical function $f(x)$,
- a floating-point program $\tilde{f}(x)$,
- the infinitely-precise evaluation $\hat{f}(x)$ of $\tilde{f}(x)$.

Definitions:
- **forward error**: $\tilde{f}(x) - f(x)$,
  - round-off error: $\tilde{f}(x) - \hat{f}(x)$
  - method error: $\hat{f}(x) - f(x)$
- **backward error**: $\tilde{x} - x$
  with $\tilde{x}$ closest from $x$ such that $f(\tilde{x}) = \tilde{f}(x)$
Numerical Analysis

Notations:
- a mathematical function $f(x)$,
- a floating-point program $\tilde{f}(x)$,
- the infinitely-precise evaluation $\hat{f}(x)$ of $\tilde{f}(x)$.

Definitions:
- **forward error**: $\tilde{f}(x) - f(x)$,
  - **round-off error**: $\tilde{f}(x) - \hat{f}(x)$
  - **method error**: $\hat{f}(x) - f(x)$

- **backward error**: $\tilde{x} - x$
  with $\tilde{x}$ closest from $x$ such that $f(\tilde{x}) = \tilde{f}(x)$

Remark: $\tilde{f}(x) - f(x) \simeq (\tilde{x} - x) \times \frac{\partial f}{\partial x}$.
In other words: **forward err** $\simeq$ **backward err** $\times$ condition num.
Evaluating $\sum_i a_i \cdot x^i$:

```plaintext
function Horner
    (a:map binary32, n:int, x:binary32)
body
    let ref y := binary32_cst(0.) in
    let ref i := n in
    for i = 0 to n - 1 do
        y := binary32_add(binary32_mul(y, x), a[i]);
    done;
y
```
Outline

Handling Machine Integers
Floating-Point Computations
Numerical Analysis
Automation
Numerical Algorithms
Using Ghost Variables for Model Values

```plaintext
function det(a b c d: binary32, aM bM cM dM: real):
    (binary32, real)
body
    let t1 := binary32_mul(a, d) in
    let t1M := aM * dM in
    let t2 := binary32_mul(b, c) in
    let t2M := bM * cM in
    let t3 := binary32_sub(t1, t2) in
    let t3M := t1M - t2M in
    (t3, t3M)

Forward error: property about t3 - t3M or t3/t3M - 1.
```
Implicit Model Values

function of_bin32(x: binary32): real
function model_of(x: binary32): real

function binary32_add(x y: binary32): binary32
  requires
    abs(rnd...(of_bin32(x) + of_bin32(y))) ≤ max_binary32
  ensures
    of_bin32(result) =
    rnd(of_bin32(x) + of_bin32(y)) ∧
    model_of(result) = model_of(x) + model_of(y)
Abstract Interpretation

Domains for floating-point variables:

- for the computed value \( x \),
- for the infinitely-precise value \( \hat{x} \),
- for the absolute error \( x - \hat{x} \),
- \( \ldots \)
Abstract Interpretation

Domains for floating-point variables:
- for the computed value \( x \),
- for the infinitely-precise value \( \hat{x} \),
- for the absolute error \( x - \hat{x} \),
- …

Naive domains:
- \([\underline{x}, \overline{x}]\) such that \( x \in [\underline{x}, \overline{x}] \),
  ex: \( \text{rnd}(x + y) \in [\text{rnd}(\underline{x} + \underline{y}), \text{rnd}(\overline{x} + \overline{y})] \),
- no domain for \( \hat{x} \),
- \( \delta_x \) such that \( |x - \hat{x}| \leq \delta_x \),
  ex: \( \delta_{x+y} = \delta_x + \delta_y + 2^{-p} \max(\overline{x} + \overline{y}, -(\underline{x} + \underline{y})) \)
Outline

Handling Machine Integers

Floating-Point Computations

Numerical Analysis

Automation

Numerical Algorithms
Newton’s Iterated Square Root

function fp_sqrt_init(x:binary64) : binary64
  requires 0.5 ≤ x ≤ 2;
  ensures abs(result - 1/sqrt(x)) ≤ 2^-6 * 1/sqrt(x);

function fp_sqrt(x:binary64) : binary64
  requires 0.5 ≤ x ≤ 2;
  ensures abs(result - sqrt(x)) ≤ 2^-43 * sqrt(x);

body
  let ref t := fp_sqrt_init(x) in
  for i = 1 to 3 do
    t := 0.5 * t * (3 - t * t * x);
  done;
  t * x
Quadratic Convergence

For all $u$ and $x$:

$$0.5u(3 - u^2 x)\sqrt{x} - 1 = -(1.5 + 0.5(u\sqrt{x} - 1)) \times (u\sqrt{x} - 1)^2$$

Loop iterations:

$$t_{n+1}\sqrt{x} - 1 \simeq 0.5t_n(3 - t_n^2 x)\sqrt{x} - 1 \simeq -1.5(t_n\sqrt{x} - 1)^2$$

Round-off error at step $n$ vanishes at step $n + 1$. 
Accurate Summation

Computing $\sum_i x_i$:

\[
\begin{align*}
  s & := x[0]; \\
  e & := 0.; \\
  \text{for } i = 1 \text{ to } n - 1 \text{ do} \\
  & \quad y := x[i]; \\
  & \quad t := s + y; \\
  & \quad u := t - y; \\
  & \quad r := (s - u) + (y - (t - u)); \\
  & \quad s := t; \\
  & \quad e := e + r; \\
  \text{done;}
\end{align*}
\]

\[
  s' := s + e;
\]
Accurate Summation

Computing $\sum_i x_i$:

```plaintext
s := x[0];
e := 0.;
for i = 1 to n - 1 do
    y := x[i];
    t := s + y;
    u := t - y;
    r := (s - u) + (y - (t - u));
    s := t;
    e := e + r;
done;
s' := s + e;
```

Naive sum
Accurate Summation

Computing $\sum_i x_i$:

\[
\begin{align*}
s & := x[0]; \\
e & := 0.; \\
for \ i = 1 \ to \ n - 1 \ do \\
\quad y & := x[i]; \\
\quad t & := s + y; \\
\quad u & := t - y; \\
\quad r & := (s - u) + (y - (t - u)); \\
\quad s & := t; \\
\quad e & := e + r; \\
done; \\
s' & := s + e;
\end{align*}
\]

Error-free addition: $t + r = s + y$
Error-Free Transformations

- Sterbenz: \( \forall x, y \in F, \ x/2 \leq y \leq 2x \Rightarrow \text{rnd}(x - y) = x - y \)
Error-Free Transformations

- **Sterbenz**: $\forall x, y \in F, \ x/2 \leq y \leq 2x \Rightarrow \text{rnd}(x - y) = x - y$

- **error of addition**: $\forall x, y \in F, \ \text{rnd}(x + y) - (x + y) \in F$
Error-Free Transformations

- **Sterbenz:** $\forall x, y \in F, \ x/2 \leq y \leq 2x \Rightarrow \text{rnd}(x - y) = x - y$

- **error of addition:** $\forall x, y \in F, \ \text{rnd}(x + y) - (x + y) \in F$

- **fast twosum:** $\forall x, y \in F, \ |x| \geq |y| \Rightarrow s + e = x + y$
  with $s = \text{rnd}(x + y)$ and $e = \text{rnd}(y - \text{rnd}(s - x))$
Error-Free Transformations

- **Sterbenz**: \( \forall x, y \in F, \ x/2 \leq y \leq 2x \Rightarrow \text{rnd}(x - y) = x - y \)

- **error of addition**: \( \forall x, y \in F, \ \text{rnd}(x + y) - (x + y) \in F \)

- **fast twosum**: \( \forall x, y \in F, \ |x| \geq |y| \Rightarrow s + e = x + y \)
  
  with \( s = \text{rnd}(x + y) \) and \( e = \text{rnd}(y - \text{rnd}(s - x)) \)

- **twosum**: \( \forall x, y \in F, \ s + e = x + y \)
  
  with \( s = \text{rnd}(x + y) \) and \( u = \text{rnd}(s - y) \) and
  
  \( e = \text{rnd}(\text{rnd}(x - u) + \text{rnd}(y - \text{rnd}(s - u))) \)
Payne & Hanek’s Argument Reduction

Reducing $x \geq 2^{31}$ to $0 \leq y \lesssim \pi/4$ for circular functions:

```plaintext
function reduce(x:binary32): (binary32, int)
  requires 2^31 \leq x
  ensures exists l:int. 
      abs((result + k * pi/4) - (x + l * 2*pi)) \leq 2^{-25}
body
  let x’ = binary64_of_binary32 x in
  let t = x’ * 1.273239545... in
  let k = trunc(t) in
  let y = (t - k) * 0.785398163... in
  (binary32_of_binary64(y), k)
```

Note: computations are performed with `binary64`. 
Payne & Hanek’s Argument Reduction

Reducing $x \geq 2^{31}$ to $0 \leq y \lesssim \pi/4$ for circular functions:

```haskell
function reduce(x:binary32): (binary32, int)
    requires 2^31 \leq x
    ensures exists l:int.
        abs((result + k * pi/4) - (x + l * 2*pi)) \leq 2^{-25}

body
    let x' = binary64_of_binary32 x in
    let t = x' * 0.02323954474... in
    let k = trunc(t) in
    let y = (t - k) * 0.785398163... in
    (binary32_of_binary64(y), k)

Note: computations are performed with binary64.
```