Numeric Programs

Guillaume Melquiond

Cours MPRI 2-36-1 “Preuve de Programme”

13/20 février 2012

Computers and Number Representations

- 32-bit signed *integers* in two-complement:

Computers and Number Representations

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

Computers and Number Representations

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

Computers and Number Representations

- 32-bit signed *integers* in two-complement:
  - $1 + 1 \rightarrow 2$,
  - $2147483647 + 1 \rightarrow -2147483648$,
  - $100000^2 \rightarrow 1410065408$,

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 exception?!).

Computers and Number Representations

- 32-bit signed *integers* in two-complement:
  - $1 + 1 \to 2$,
  - $2147483647 + 1 \to -2147483648$,
  - $100000^2 \to 1410065408$,
  - -2147483648
    $\text{mod } -1 \to$ boom (floating-point exception?!).
- IEEE-754 binary64 *floating-point numbers*:

Computers and Number Representations

- 32-bit signed *integers* in two-complement:
  - $1 + 1 \to 2$,
  - $2147483647 + 1 \to -2147483648$,
  - $100000^2 \to 1410065408$,
  - -2147483648
    $\text{mod } -1 \to$ boom (floating-point exception?!).
- IEEE-754 binary64 *floating-point numbers*:
  - $2 \times 2 \times \cdots \times 2 \to +\infty$,

Computers and Number Representations

- 32-bit signed *integers* in two-complement:
  - $1 + 1 \to 2$,
  - $2147483647 + 1 \to -2147483648$,
  - $100000^2 \to 1410065408$,
  - -2147483648
    $\text{mod } -1 \to$ boom (floating-point exception?!).
- IEEE-754 binary64 *floating-point numbers*:
  - $2 \times 2 \times \cdots \times 2 \to +\infty$,
  - $1 \div 0 \to +\infty$,

Computers and Number Representations

- 32-bit signed *integers* in two-complement:
  - $1 + 1 \to 2$,
  - $2147483647 + 1 \to -2147483648$,
  - $100000^2 \to 1410065408$,
  - -2147483648
    $\text{mod } -1 \to$ boom (floating-point exception?!).
- IEEE-754 binary64 *floating-point numbers*:
  - $2 \times 2 \times \cdots \times 2 \to +\infty$,
  - $1 \div 0 \to +\infty$,
  - $1 \div -0 \to -\infty$,

Computers and Number Representations

- 32-bit signed *integers* in two-complement:
  - $1 + 1 \to 2$,
  - $2147483647 + 1 \to -2147483648$,
  - $100000^2 \to 1410065408$,
  - -2147483648
    mod $-1 \to$ boom (floating-point exception?!).
- IEEE-754 binary64 *floating-point numbers*:
  - $2 \times 2 \times \cdots \times 2 \to +\infty$,
  - $1 \div 0 \to +\infty$,
  - $1 \div -0 \to -\infty$,
  - $0 \div 0 \to$ NaN.

Some Other Numerical Failures

- 1983, truncation in computing an index of Vancouver Stock Exchange makes it to drop to half its value.

Some Other Numerical Failures

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

Some Other Numerical Failures

- 1983, truncation in computing an index of Vancouver Stock Exchange makes it to drop to half its value.
- 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.

Some Other Numerical Failures

- 1983, truncation in computing an index of Vancouver Stock Exchange makes it to drop to half its value.
- 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 Other Numerical Failures

- 1983, truncation in computing an index of Vancouver Stock Exchange makes it to drop to half its value.
- 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 Other Numerical Failures

- 1983, truncation in computing an index of Vancouver Stock Exchange makes it to drop to half its value.
- 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.

Verifying Numerical Algorithms

- 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.: $0.1 \simeq 209715 \cdot 2^{-24}$.
  Cumulated skew after 24h: -0.08s, distance: 160m.
  System was supposed to be rebooted periodically.

Verifying Numerical Algorithms

- 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.: $0.1 \simeq 209715 \cdot 2^{-24}$.
  Cumulated skew after 24h: -0.08s, distance: 160m.
  System was supposed to be rebooted periodically.

- 2007, Excel displays 77.1×850 as 100000.

  Bug in binary/decimal conversion.
  Failing inputs: 12 FP numbers.
  Probability to uncover them by random testing: 10^{-18}.

Outline

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.

Issues are similar for other arithmetic issues, e.g. division by 0.

Safety Checking with Hoare Logic

Idea: replace all arithmetic operations by abstract functions with preconditions. $x - y$ becomes int32_sub(x, y).

function int32_sub(x: int, y: int): int
 requires $-2^{31} \leq x - y < 2^{31}$
 ensures result = $x - y$

Safety Checking with Hoare Logic, 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*.

type int32
function of_int32(x: int32): int
axiom bounded_int32: **forall** x: int32. $-2^{31} \leq$ of_int32(x) $< 2^{31}$

function int32_sub(x: int32, y: int32): int32
  requires $-2^{31} \leq$ of_int32(x) + of_int32(y) $< 2^{31}$
  ensures of_int32(**result**) = of_int32(x) $-$ of_int32(y)

Exercises

1. How to handle int32 *constants* in programs?

Exercises

1. How to handle int32 *constants* in programs?
2. How to specify *saturating* arithmetic?

Outline

Floating-Point Arithmetic

- Limited range $\Rightarrow$ *exceptional* behaviors.
- Limited *precision* $\Rightarrow$ *inaccurate* results.

Floating-Point Data

IEEE-754 Binary Floating-Point Arithmetic
$1 + w_e + w_m = 32$, or 64, or 128.

A floating-point datum

sign s	biased exponent e' (w_e bits)	mantissa m (w_m bits)

represents

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

Bias: $2^{w_e - 1} - 1$. Precision: $p = w_m + 1$.

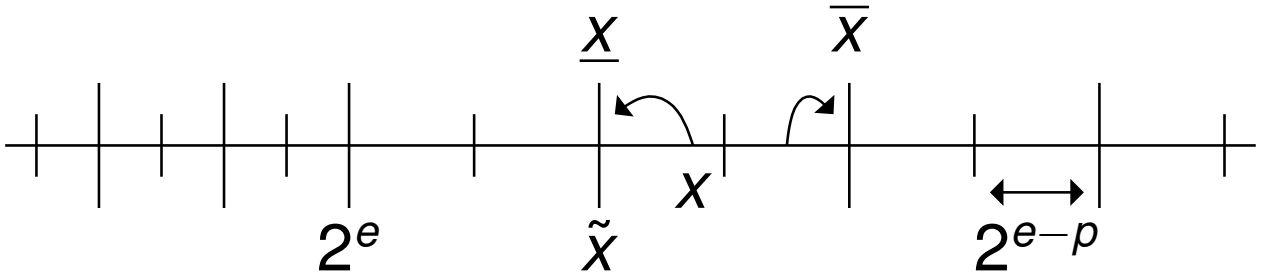
Semantics for the Finite Case

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.

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))

Usual Properties

- *idempotency*: $\forall x \in \mathbb{R},\ \mathrm{rnd}(\mathrm{rnd}(x)) = \mathrm{rnd}(x)$

Usual Properties

- *idempotency*: $\forall x \in \mathbb{R},\ \mathrm{rnd}(\mathrm{rnd}(x)) = \mathrm{rnd}(x)$
- *monotonicity*: $\forall x, y \in \mathbb{R},\ x \leq y \Rightarrow \mathrm{rnd}(x) \leq \mathrm{rnd}(y)$

Usual Properties

- *idempotency*: $\forall x \in \mathbb{R},\ \mathrm{rnd}(\mathrm{rnd}(x)) = \mathrm{rnd}(x)$
- *monotonicity*: $\forall x, y \in \mathbb{R},\ x \leq y \Rightarrow \mathrm{rnd}(x) \leq \mathrm{rnd}(y)$
- *successor*: if $m \cdot 2^e$ is a canonical representation, then $(m+1) \cdot 2^e$ is representable as FP; there is no representable number between $m \cdot 2^e$ and $(m+1) \cdot 2^e$, when $m \geq 0$

Usual Properties

- *idempotency*: $\forall x \in \mathbb{R},\ \mathrm{rnd}(\mathrm{rnd}(x)) = \mathrm{rnd}(x)$
- *monotonicity*: $\forall x, y \in \mathbb{R},\ x \leq y \Rightarrow \mathrm{rnd}(x) \leq \mathrm{rnd}(y)$
- *successor*: if $m \cdot 2^e$ is a canonical representation, then $(m+1) \cdot 2^e$ is representable as FP; there is no representable number between $m \cdot 2^e$ and $(m+1) \cdot 2^e$, when $m \geq 0$
- *round-off error*: $\forall x \in \mathbb{R},\ \mathrm{rnd}(x) = x \cdot (1+\varepsilon) + \delta$ with $|\varepsilon| \leq 2^{-p}$ and $|\delta| \leq 2^{-bias-p+1}$ when rounding to nearest

Usual Properties

- *idempotency*: $\forall x \in \mathbb{R},\ \mathrm{rnd}(\mathrm{rnd}(x)) = \mathrm{rnd}(x)$
- *monotonicity*: $\forall x, y \in \mathbb{R},\ x \leq y \Rightarrow \mathrm{rnd}(x) \leq \mathrm{rnd}(y)$
- *successor*: if $m \cdot 2^e$ is a canonical representation, then $(m+1) \cdot 2^e$ is representable as FP; there is no representable number between $m \cdot 2^e$ and $(m+1) \cdot 2^e$, when $m \geq 0$
- *round-off error*: $\forall x \in \mathbb{R},\ \mathrm{rnd}(x) = x \cdot (1+\varepsilon) + \delta$ with $|\varepsilon| \leq 2^{-p}$ and $|\delta| \leq 2^{-bias-p+1}$ when rounding to nearest
- *round-off error for addition*: $\delta = 0$

Outline

Numerical Errors

Given two real numbers u and v,

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

Numerical Errors

Given two real numbers u and v,

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

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*: $u/v - 1, \ldots$

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: $\mathrm{rnd}(u) - v = (\mathrm{rnd}(u) - u) + (u - v)$

Numerical Analysis

Notations:

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

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: $\hat{f}(x) - f(x)$

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*.

Numerical Analysis

Evaluating $\sum_i a_i \cdot x^i$:

```
function Horner(a:map int binary32, n:int, x:binary32)
 body
  let y := ref (binary32_cst 0.) in
  let i := ref n in
  while i ≥ 0 do
   y := binary32_add(binary32_mul(y, x), a[i]);
   i := i − 1;
  done;
  y
```

Outline

Using Ghost Variables for Model Values

```
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: binary32, y: binary32): binary32
  requires abs(rnd...(of_bin32(x) + of_bin32(y))) $\leq$ max_binary32
  ensures of_bin32(**result**) = rnd(of_bin32(x) + of_bin32(y)) $\wedge$
    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 $x - \hat{x}$,
- ...

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: $\mathrm{rnd}(x + y) \in [\mathrm{rnd}(\underline{x} + \underline{y}), \mathrm{rnd}(\overline{x} + \overline{y})]$,
- no domain for $\hat{x}$,
- δ_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

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 t := ref (fp_sqrt_init(x)) in
    let i := ref 0 in
    while i < 3 do
      t := 0.5 ∗ t ∗ (3 − t ∗ t ∗ x);
      i := i + 1;
    done;
    t ∗ x
```

Quadratic Convergence

For all u and x:

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

Loop iterations:

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

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

Accurate Summation

Computing $\sum_i x_i$:

```
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;
```

Accurate Summation

Computing $\sum_i x_i$:

```
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 $\sum_i x_i$:

```
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;
```

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

Error-Free Transformations

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

Error-Free Transformations

- Sterbenz: $\forall x, y \in F,\ x/2 \leq y \leq 2x \Rightarrow \mathrm{rnd}(x - y) = x - y$
- error of addition: $\forall x, y \in F,\ \mathrm{rnd}(x + y) - (x + y) \in F$

Error-Free Transformations

- Sterbenz: $\forall x, y \in F,\ x/2 \leq y \leq 2x \Rightarrow \mathrm{rnd}(x - y) = x - y$
- error of addition: $\forall x, y \in F,\ \mathrm{rnd}(x + y) - (x + y) \in F$
- fast twosum: $\forall x, y \in F,\ |x| \geq |y| \Rightarrow s + e = x + y$
  with $s = \mathrm{rnd}(x + y)$ and $e = \mathrm{rnd}(y - \mathrm{rnd}(s - x))$

Error-Free Transformations

- Sterbenz: $\forall x, y \in F,\ x/2 \leq y \leq 2x \Rightarrow \mathrm{rnd}(x - y) = x - y$
- error of addition: $\forall x, y \in F,\ \mathrm{rnd}(x + y) - (x + y) \in F$
- fast twosum: $\forall x, y \in F,\ |x| \geq |y| \Rightarrow s + e = x + y$
  with $s = \mathrm{rnd}(x + y)$ and $e = \mathrm{rnd}(y - \mathrm{rnd}(s - x))$
- twosum: $\forall x, y \in F,\ s + e = x + y$
  with $s = \mathrm{rnd}(x + y)$ and $u = \mathrm{rnd}(s - y)$ and
  $e = \mathrm{rnd}(\mathrm{rnd}(x - u) + \mathrm{rnd}(y - \mathrm{rnd}(s - u)))$

Payne & Hanek's Argument Reduction

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

```
function reduce(x:binary32): (binary32, int)
  requires 2^31 ≤ x
  ensures exists l:int. abs((result + k ∗ pi/4) − (x + l ∗ 2∗pi)) ≤ 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:

```
function reduce(x:binary32): (binary32, int)
  requires 2^31 ≤ x
  ensures exists l:int. abs((result + k ∗ pi/4) − (x + l ∗ 2∗pi)) ≤ 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.